Université de Montréal

Conditional Generative Modeling for Images, 3D Animations, and Video

par

Vikram Voleti

Département d’informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l’obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Informatique

July 24, 2023

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Conditional Generative Modeling for Images, 3D Animations, and Video

présentée par

Vikram Voleti

a été évaluée par un jury composé des personnes suivantes :

Aaron Courville
(président-rapporteur)

Christopher Pal
(directeur de recherche)

Ioannis Mitliagkas
(membre du jury)

Greg Mori
(examinateur externe)

Aaron Courville
(représentant du doyen de la FESP)

Résumé

La modélisation générative pour la vision par ordinateur a connu d'immenses progrès ces dernières années, révolutionnant notre façon de percevoir, comprendre et manipuler les données visuelles. Ce domaine en constante évolution a connu des avancées dans la génération d'images, l'animation 3D et la prédiction vidéo, débloquant ainsi diverses applications dans plusieurs domaines tels que le divertissement, le design, la santé et l'éducation. Alors que la demande de systèmes de vision par ordinateur sophistiqués ne cesse de croître, cette thèse s'efforce de stimuler l'innovation dans le domaine en explorant de nouvelles formulations de modèles génératifs conditionnels et des applications innovantes dans les images, les animations 3D et la vidéo.

Notre recherche se concentre sur des architectures offrant des transformations réversibles du bruit et des données visuelles, ainsi que sur l'application d'architectures encodeur-décodeur pour les tâches génératives et la manipulation de contenu 3D. Dans tous les cas, nous incorporons des informations conditionnelles pour améliorer la synthèse des données visuelles, améliorant ainsi l'efficacité du processus de génération ainsi que le contenu généré.

Les techniques génératives antérieures qui sont réversibles entre le bruit et les données et qui ont connu un certain succès comprennent les flux de normalisation et les modèles de diffusion de débruitage. La variante continue des flux de normalisation est alimentée par les équations différentielles ordinaires neuronales (Neural ODEs) et a montré une certaine réussite dans la modélisation de la distribution d'images réelles. Cependant, elles impliquent souvent un grand nombre de paramètres et un temps d'entraînement élevé. Les modèles de diffusion de débruitage ont récemment gagné énormément en popularité en raison de leurs capacités de généralisation, notamment dans les applications de texte vers image.

Dans cette thèse, nous introduisons l'utilisation des Neural ODEs pour modéliser la dynamique vidéo à l'aide d'une architecture encodeur-décodeur, démontrant leur capacité à prédire les images vidéo futures malgré le fait d'être entraînées uniquement à reconstruire les images actuelles. Dans notre prochaine contribution, nous proposons une variante conditionnelle des flux de normalisation continus qui permet une génération d'images à résolution supérieure à partir d'une entrée à résolution inférieure. Cela nous permet d'obtenir une qualité d'image comparable à celle des flux de normalisation réguliers, tout en réduisant considérablement le nombre de paramètres et le temps d'entraînement.

Notre prochaine contribution se concentre sur une architecture encodeur-décodeur flexible pour l'estimation et l'édition précises de la pose humaine en 3D. Nous présentons un pipeline complet qui prend des images de personnes en entrée, aligne automatiquement un personnage 3D humain/non humain spécifié par l'utilisateur sur la pose de la personne, et facilite l'édition de la pose en fonction d'informations partielles.

Nous utilisons ensuite des modèles de diffusion de débruitage pour la génération d'images et de vidéos. Les modèles de diffusion réguliers impliquent l'utilisation d'un processus gaussien pour ajouter du bruit aux images propres. Dans notre prochaine contribution, nous dérivons les détails mathématiques pertinents pour les modèles de diffusion de débruitage qui utilisent des processus gaussiens non isotropes, présentons du bruit non isotrope, et montrons que la qualité des images générées est comparable à la formulation d'origine. Dans notre dernière contribution, nous concevons un nouveau cadre basé sur les modèles de diffusion de débruitage, capable de résoudre les trois tâches vidéo de prédiction, de génération et d'interpolation. Nous réalisons des études d'ablation en utilisant ce cadre et montrons des résultats de pointe sur plusieurs ensembles de données.

Nos contributions sont des articles publiés dans des revues à comité de lecture. Dans l'ensemble, notre recherche vise à apporter une contribution significative à la poursuite de modèles génératifs plus efficaces et flexibles, avec le potentiel de façonner l'avenir de la vision par ordinateur.

Mots-clés: Apprentissage profond, vision par ordinateur, modèles génératifs, apprentissage de la représentation, modèles de diffusion

Summary

Generative modeling for computer vision has shown immense progress in the last few years, revolutionizing the way we perceive, understand, and manipulate visual data. This rapidly evolving field has witnessed advancements in image generation, 3D animation, and video prediction that unlock diverse applications across multiple fields including entertainment, design, healthcare, and education. As the demand for sophisticated computer vision systems continues to grow, this dissertation attempts to drive innovation in the field by exploring novel formulations of conditional generative models, and innovative applications in images, 3D animations, and video.

Our research focuses on architectures that offer reversible transformations of noise and visual data, and the application of encoder-decoder architectures for generative tasks and 3D content manipulation. In all instances, we incorporate conditional information to enhance the synthesis of visual data, improving the efficiency of the generation process as well as the generated content.

Prior successful generative techniques which are reversible between noise and data include normalizing flows and denoising diffusion models. The continuous variant of normalizing flows is powered by Neural Ordinary Differential Equations (Neural ODEs), and have shown some success in modeling the real image distribution. However, they often involve huge number of parameters, and high training time. Denoising diffusion models have recently gained huge popularity for their generalization capabilities especially in text-to-image applications.

In this dissertation, we introduce the use of Neural ODEs to model video dynamics using an encoder-decoder architecture, demonstrating their ability to predict future video frames despite being trained solely to reconstruct current frames. In our next contribution, we propose a conditional variant of continuous normalizing flows that enables higher-resolution image generation based on lower-resolution input. This allows us to achieve comparable image quality to regular normalizing flows, while significantly reducing the number of parameters and training time.

Our next contribution focuses on a flexible encoder-decoder architecture for accurate estimation and editing of full 3D human pose. We present a comprehensive pipeline that takes human images as input, automatically aligns a user-specified 3D human/non-human character with the pose of the human, and facilitates pose editing based on partial input information.

We then proceed to use denoising diffusion models for image and video generation. Regular diffusion models involve the use of a Gaussian process to add noise to clean images. In our next contribution, we derive the relevant mathematical details for denoising diffusion models that use non-isotropic Gaussian processes, present non-isotropic noise, and show that the quality of generated images is comparable with the original formulation. In our final contribution, devise a novel framework building on denoising diffusion models that is capable of solving all three video tasks of prediction, generation, and interpolation. We perform ablation studies using this framework, and show state-of-the-art results on multiple datasets.

Our contributions are published articles at peer-reviewed venues. Overall, our research aims to make a meaningful contribution to the pursuit of more efficient and flexible generative models, with the potential to shape the future of computer vision.

Keywords: Deep learning, computer vision, generative models, representation learning, diffusion models

Contents

Résumé 5

Summary 7

List of tables 17

List of figures 19

List of acronyms and abbreviations 23

Acknowledgements 25

Chapter 1. Introduction 27

1.1 Computer vision and machine learning 27

1.2 Generative modeling 28

1.2.1 Continuous normalizing flows 29

1.2.2 Denoising diffusion models 29

1.3 Conditional generative modeling 30

1.4 Applications 30

1.5 Thesis overview 31

Chapter 2. Background 35

2.1 Ordinary Differential Equations (ODEs) 35

2.1.1 Initial Value Problem (IVP) 35

2.1.2 Numerical integration 36

2.2 Neural Ordinary Differential Equations (Neural ODEs) 39

2.2.1 Adjoint method 39

2.3 Continuous Normalizing Flows (CNFs) 41

2.4 Denoising Diffusion Probabilistic Models (DDPM) 43
2.4.1 Forward (data to noise) for DDPM 43
2.4.2 Score for DDPM 43
2.4.3 Score-matching objective function for DDPM 44
2.4.4 Variance of score for DDPM 44
2.4.5 Overall objective function for DDPM 44
2.4.6 Noise-matching objective for DDPM 44
2.4.7 Reverse (noise to data) in DDPM 45
2.4.8 Sampling in DDPM 45
2.4.9 Sampling using DDIM 46
2.4.10 Expected Denoised Sample (EDS) for DDPM 47
2.4.11 SDE formulation : Variance Preserving (VP) SDE 48
2.4.11.1 Derivations 48

2.5 Score Matching Langevin Dynamics (SMLD) 50
2.5.1 Forward (data to noise) for SMLD 50
2.5.2 Score for SMLD 50
2.5.3 Score-matching objective function for SMLD 50
2.5.4 Variance of score for SMLD 51
2.5.5 Overall objective function for SMLD 51
2.5.6 Unconditional SMLD score estimation 51
2.5.7 Sampling in SMLD 51
2.5.8 Expected Denoised Sample (EDS) for SMLD 52
2.5.9 SDE formulation : Variance Exploding (VE) SDE 52
2.5.9.1 Derivations 52

Chapter 3. Simple video generation using Neural ODEs [Voleti et al., 2019] 55

3.0 Prologue to article 55
3.0.1 Article details 55
3.0.2 Context 55
3.0.3 Contributions 56
3.0.4 Research impact 56

3.1 Introduction 57

3.2 Related work 57

3.3 Neural Ordinary Differential Equations (Neural ODEs) 58

3.4 Our approach 59

3.5 Experiments on Moving MNIST 61
3.5.1 1-digit Moving MNIST with non-RNN Encoder 62
3.5.2 1-digit Moving MNIST with RNN Encoder 63
3.5.3 2-digit Moving MNIST with RNN Encoder 64
3.5.4 A note on the problem formulation 64

3.6 Future work 64

3.7 Conclusion 67

Chapter 4. Multi-Resolution Continuous Normalizing Flows [Voleti et al., 2021] 69

4.0 Prologue to article 69
4.0.1 Article details 69
4.0.2 Context 69
4.0.3 Contributions 70
4.0.4 Research impact 70

4.1 Introduction 71

4.2 Background 72
4.2.1 Normalizing flows 72
4.2.2 Continuous Normalizing Flows (CNF) 73

4.3 Our method 74
4.3.1 Multi-resolution image representation 74
4.3.2 Defining the high-level information $\mathbf{y}_s$ 75
4.3.3 Multi-Resolution Continuous Normalizing Flows (MRCNF) 77
4.3.4 Multi-resolution noise 78

4.3.5 8-bit to uniform 79
4.3.6 Simple example of density estimation in MRCNF 79
4.3.7 Simple example of generation using MRCNF 80

4.4 Related work 80
4.4.1 WaveletFlow 81

4.5 Experimental details 82
4.5.1 Models 82
4.5.2 Gradient norm 83

4.6 Experimental results 83
4.6.1 Super-resolution 85
4.6.2 Progressive training 85
4.6.3 Ablation study 85
4.6.4 Adversarial loss 86
4.6.5 FID v/s temperature 87

4.7 Examining Out-of-Distribution (OoD) behaviour 87
4.7.1 Shuffled in-distribution images 89

4.8 Qualitative samples 90

4.9 Conclusion 91

Chapter 5. Neural Inverse Kinematics with 3D Human Pose Prior [Voleti et al., 2022b] 93

5.0 Prologue to article 93
5.0.1 Article details 93
5.0.2 Context 94
5.0.3 Contributions 94
5.0.4 Research impact 94

5.1 Introduction 95

5.2 Background 96
5.2.1 Skinned Multi-Person Linear model (SMPL) 96

5.2.2 3D human pose datasets ... 97
5.2.3 Inverse Kinematics (IK) ... 97
5.2.4 Retargeting ... 97
5.2.5 SMPL and IK ... 97

5.3 SMPL Inverse Kinematics (SMPL-IK) ... 100

5.4 SMPL Shape Inversion (SMPL-SI) ... 100

5.5 Proposed workflow ... 101
5.5.1 Overall pipeline ... 103
5.5.2 Image to 3D pose ... 104
5.5.3 Custom character shape estimation using SMPL-SI ... 104
5.5.4 Retarget pose to custom character ... 104
5.5.5 Pose editing ... 104
5.5.6 Effector recovery ... 105

5.6 SMPL-IK details ... 105
5.6.1 SMPL-IK neural network diagram ... 105
5.6.2 SMPL-IK training details ... 106
5.6.3 SMPL-IK evaluation details ... 106

5.7 Empirical results ... 107

5.8 Limitations ... 107

5.9 Demos ... 108
5.9.1 Editing both shape and pose demo ... 108
5.9.2 Labeling tool demo ... 109
5.9.3 Pose authoring on a custom character ... 109
5.9.4 Effector recovery ... 109
5.9.5 Full pipeline demos ... 110
5.9.5.1 `Demo_Crouch_FineTuning.mp4` ... 110
5.9.5.2 `Demo_Sitting_Editing.mp4` ... 110

5.10 Conclusion ... 111

Chapter 6. Non-Isotropic Denoising Diffusion Models [Voleti et al., 2022c] 113

6.0 Prologue to article ... 113
6.0.1 Article details ... 113
6.0.2 Context ... 113
6.0.3 Contributions ... 114
6.0.4 Research impact ... 114

6.1 Introduction ... 115

6.2 Non-isotropic DDPM (NI-DDPM) ... 116
6.2.1 Forward (data to noise) for NI-DDPM ... 116
6.2.2 Score for NI-DDPM ... 116
6.2.3 Score-matching objective for NI-DDPM ... 116
6.2.4 Variance of score for NI-DDPM ... 117
6.2.5 Overall objective function for NI-DDPM ... 117
6.2.6 Noise-matching objective for NI-DDPM ... 117
6.2.7 Reverse (noise to data) for NI-DDPM ... 118
6.2.8 Sampling in NI-DDPM ... 119
6.2.9 Sampling in NI-DDPM using DDIM ... 120
6.2.10 Expected Denoised Sample (EDS) for NI-DDPM ... 120
6.2.11 SDE formulation : Non-Isotropic Variance Preserving (NIVP) SDE ... 120
6.2.11.1 Derivations ... 120

6.3 Non-isotropic SMLD (NI-SMLD) ... 122
6.3.1 Forward (data to noise) for NI-SMLD ... 122
6.3.2 Score for NI-SMLD ... 122
6.3.3 Score-matching objective function for NI-SMLD ... 122
6.3.4 Variance of score for NI-SMLD ... 122
6.3.5 Overall objective function for NI-SMLD ... 122
6.3.6 Unconditional NI-SMLD score estimation ... 123
6.3.7 Sampling i.e. Reverse (noise to data) for NI-SMLD ... 123
6.3.8 Expected Denoised Sample (EDS) for NI-SMLD ... 123
6.3.9 SDE formulation : Non-Isotropic Variance Exploding (NIVE) SDE ... 124
6.3.9.1 Derivations ... 124

6.3.10 Initial noise scale for NI-SMLD ... 125
6.3.11 Other noise scales ... 126
6.3.12 Configuring annealed Langevin dynamics ... 126
6.3.13 Optimal conditional score function ... 127

6.4 Isotropic DDPM v/s Non-isotropic DDPM ... 128
6.4.1 Isotropic Gaussian denoising diffusion models (DDPM) ... 128
6.4.2 Non-isotropic Gaussian denoising diffusion models (NI-DDPM) ... 130

6.5 Gaussian Free Field (GFF) images ... 132
6.5.1 Probability distribution of GFF ... 134
6.5.2 Log probability of transformation ... 135
6.5.3 Varying **K** ... 136

6.6 Experimental results ... 136

6.7 Conclusion ... 137

Chapter 7. MCVD: Masked Conditional Video Diffusion [Voleti et al., 2022a] 139

7.0 Prologue to article ... 139
7.0.1 Article details ... 139
7.0.2 Context ... 139
7.0.3 Contributions ... 140
7.0.4 Research impact ... 140

7.1 Introduction ... 141

7.2 Masked conditional diffusion for video ... 143
7.2.1 Video prediction ... 144
7.2.2 Video prediction + generation ... 145
7.2.3 Video prediction + generation + interpolation ... 146
7.2.4 Network architecture ... 146

7.3 Related work ... 147

7.4 Experiments ... 150
7.4.1 Tasks ... 150

7.4.2 Datasets 151
7.4.3 Training details 151
7.4.4 Metrics 152

7.5 Results 152
7.5.1 Video prediction 153
7.5.2 Video generation 155
7.5.3 Video interpolation 155
7.5.4 Ablation studies 156

7.6 Conclusion 157
7.6.1 Limitations 157
7.6.2 Broader Impacts 157

7.7 Qualitative results 158
7.7.1 Stochastic Moving MNIST 159
7.7.2 KTH 160
7.7.3 BAIR 161
7.7.4 UCF-101 162
7.7.5 Cityscapes 163

Chapter 8. Conclusion 165

References 169

List of tables

4.1 Number of parameters for different models with different total number of resolutions (res), and the number of channels (ch) and number of blocks (bl) per resolution. . 82

4.2 Unconditional image generation metrics (lower is better): parameters (P), bits-per-dimension, time (in hours). All our models were trained on only *one* NVIDIA V100 GPU. 84

4.3 Metrics for unconditional ImageNet128 generation. Param is number of parameters, Time is in hours. '-' indicates unreported values. 85

4.4 Ablation study across using Wavelet in Equation 4.3.2, and multi-resolution noise formulation in subsection 4.3.4. P is number of parameters, Time is in hours. Lower is better in all cases. '-' indicates unreported values. 'x' : Fails to train3. . 86

4.5 FID v/s temperature for Multi-Resolution Continuous Normalizing Flow (MRCNF) models trained on CIFAR10. 87

4.6 auROC for out-of-distribution (OoD) detection using -bpd and S[Serrà et al., 2020], for models trained on CIFAR10. 89

5.1 Table of 3D human pose datasets 98

5.2 SMPL-IK benchmark following the randomized effector scheme [Oreshkin et al., 2021] on AMASS and H36M datasets, based on MPJPE (Mean Per Joint Position Error), PA-MPJPE (Procrustes-Aligned MPJPE), and GE (Geodesic Error) metrics. 107

6.1 Image generation metrics FID, Precision (P), and Recall (R) for CIFAR10 using DDPM and NI-DDPM, with different generation steps. 136

7.1 Compute used. "steps" indicates the checkpoint with the best approximate FVD, "GPU hours" is the total training time up to "steps". 152

7.2 Video prediction on Cityscapes (128 × 128), 2 past frames, predicting 28. 153

7.3 Video prediction results on SMMNIST (64×64) for 10 predicted frames conditioned on 5 past frames. We predicted 10 trajectories per real video, and report the average FVD and maximum SSIM, averaged across 256 test videos. 153

7.4 Video prediction results on KTH (64×64), predicting 30/40 frames from 10 past frames using models trained to predict k frames at a time. All models test on 256 videos. 153

7.5 Video prediction results on BAIR (64×64) conditioning on p past frames and predicting $pred$ frames in the future, using models trained to predict k frames at at time. 154

7.6 Unconditional generation of BAIR video frames. 155

7.7 Unconditional generation of UCF-101 video frames. 155

7.8 Video Interpolation results (64×64). Given p past + f future frames $\rightarrow$ interpolate k frames. Reporting average of the best metrics out of n trajectories per test sample. $\downarrow(p+f)$ and $\uparrow k$ is harder. We used MCVD spatin past-mask for SMMNIST and KTH, and MCVD concat past-future-mask for BAIR. We also include results on SMMNIST for a "pure" model trained without any masking. 155

7.9 Results on the SMMNIST evaluation, conditioned on 5 past frames, predicting 10 frames using models trained to predict 5 frames at a time. 156

List of figures

2.1 Illustration of 1st-order Runge-Kutta / Euler's method of numerical integration. 37

2.2 Illustration of the solution to ODEs being flows (taken from Yan et al. [2020]). . 38

2.3 Illustration of the solution to ODEs being flows. 39

3.1 Architectures for Encoder-ODE-Decoder 62

3.2 Samples predicted at 20 time steps, conditioned on the first 10 time steps with frames from (a) train set and (b) validation set using Non-RNN Encoder — ODE — Decoder (Figure 3.1a). In each, top row are original samples, and bottom row are predicted samples. For this figure, we use the trained model to reconstruct the first 10 frames and then predict the next 10 frames 63

3.3 Predicted samples at 20 time steps conditioned on the first 10 time steps using RNN Encoder-ODE-Decoder (Figure 3.1b). Original (top row) and predicted (bottom row) samples are shown for both (a) train and (b) validation sets. The model is trained to reconstruct the first 10 frames and predict the next 10 frames. 65

3.4 (top) Original and (bottom) predicted samples of RNN Encoder — ODE — Decoder on 2-digit Moving MNIST 66

4.1 The architecture of our MRCNF method (best viewed in color). Continuous normalizing flows (CNFs) g_s are used to generate images $\mathbf{x}_s$ from noise $\mathbf{z}_s$ at each resolution, with those at finer resolutions conditioned (dashed lines) on the coarser image one level above $\mathbf{x}_{s+1}$, except at the coarsest level where it is unconditional. Every finer CNF produces an intermediate image $\mathbf{y}_s$, which is then combined with the immediate coarser image $\mathbf{x}_{s+1}$ using a linear map M from Equation 4.3.6 to form $\mathbf{x}_s$. The multiscale maps are defined by Equation 4.3.16. 71

4.2 Tetrahedron in 3D space with 4 corners. $c = 2^{2/3}$ 75

4.3 ImageNet: Example of super-resolving to 64x64 from ground truth 16x16. Row 1: ground truth 16x16, Row 2: generated 32x32, Row 3: generated 64x64 Row 4: ground truth 64x64. 85

4.4 Histogram of log likelihood per dimension i.e. −BPD (estimated using normalized empirical Kernel Density Estimation) of OoD datasets (TinyImageNet, SVHN, Constant) under (MR)CNF models trained on CIFAR10. As with other likelihood-based generative models such as Glow & PixelCNN, OoD datasets have higher likelihood under (MR)CNFs. 87

4.5 (a) Example of shuffling different-sized patches of a 32x32 image: (left to right, top to bottom) 1x1, 2x2, 4x4, 8x8, 16x16, 32x32 (unshuffled) (b) Histogram of log likelihood per dimension (normalized empirical Kernel Density Estimate) for MRCNF models at different resolutions, trained on CIFAR10. 89

4.6 Generated samples from MNIST at different resolutions. 90

4.7 Generated samples from CIFAR10 at different resolutions. 90

5.1 Our approach unlocks novel artistic workflows such as the one depicted above. An animator uses an image to initialize an editable 3D scene. Thus a multi-person 3D scene acquired from an RGB picture is populated with custom user-defined characters whose 3D poses are further edited with the state-of-the-art machine learning inverse kinematics tool integrated in a real-time 3D development software. 95

5.2 Pipeline for pose estimation and editing from a 2D image to a custom humanoid 3D character in the same pose as the human in the 2D image. Using any advanced animation tool, our approach enables an animator to apply poses extracted from an image to a custom user-defined 3D character and edit them further using SMPL-IK, our state-of-the-art machine learning inverse kinematics tool, integrated in real-time 3D development software (such as Unity as shown here). Given an RGB image of a human in a certain pose, a pose estimation algorithm is used to estimate the human's pose in terms of SMPL parameters. Our novel SMPL-SI then maps the skeleton of the user-defined 3D character onto its SMPL approximation, and the estimated human pose is retargeted onto the SMPL shape approximation obtained from SMPL-SI. This gives an SMPL mesh in the human pose with the morphology of the custom 3D character. From here, the animator can perform pose editing in the SMPL space of pose parameters, using our proposed SMPL-IK, while keeping the shape parameters fixed. To provide the user with effectors that control the pose of the 3D character (shown in purple in the demo videos), our proposed Effector Recovery method extracts only a few effectors to create an editable initial pose, while best preserving the estimated pose. 102

5.3 SMPL-IK neural network diagram. Note input conditioning on β and gender inputs. 106

5.4 Morphology-aware learned IK. Left: posing the average male SMPL character. Center: result of modifying only the SMPL gender parameter. Right: result of additionally modifying the SMPL β shape parameters. 108

5.5 Pipeline for 2D image labeling with accurate 3D pose. 109

5.6 Pose authoring with a custom humanoid character via SMPL-IK and SMPL-SI. . 110

6.1 Gaussian noise samples. 115

6.2 Isotropic v/s Non-Isotropic Denoising Diffusion Probabilistic Models 129

6.3 (left to right) 10 GFF images (each varying downwards) as a function of the power γ of the index (mentioned on the left). 132

7.1 Our approach generates high quality frames many steps into the future: Given two conditioning frames from the Cityscapes [Cordts et al., 2016] validation set (top left), we show 7 predicted future frames in row 2 below, then skip to frames 20-28, autoregressively predicted in row 4. Ground truth frames are shown in rows 1 and 3. Notice the initial large arrow advancing and passing under the car. In frame 20 (the far left of the 3rd and 4th row), the initially small and barely visible second arrow in the background of the conditioning frames has advanced into the foreground. Result generated by our **MCVD** method (concat variant). Note that some Cityscapes videos contain brightness changes, which may explain the brightness change in this sample. 141

7.2 Blockwise autoregressive prediction with our model. 144

7.3 This figure shows our block autoregressive strategy where the top row and third row are ground truth, and the second and fourth rows show the blockwise autoregressively generated frames using our approach. 145

7.4 Noisy current frames are given to a U-Net whose residual blocks receive conditional information from past/future frames and noise-level. The output is the predicted noise in the current frames, which is used to denoise the current frames. 147

7.5 Comparing future prediction methods on Cityscapes: SVG-LP (Top Row), Hier-vRNNs (Second Row), Our Method (Third Row), Ground Truth (Bottom Row). Frame 2, a ground truth conditioning frame is shown in first column, followed

by frames: 3, 5, 10 and 20 generated by each method vs the ground truth at the bottom. 149
7.6 Visualization of video (a) tasks and (b) our proposed masked solutions. 151
7.7 SMMNIST 5 $\rightarrow$ 10, trained on 5 (prediction). For each sample, top row is real ground truth, bottom is predicted by our MCVD model. 159
7.8 KTH 5 $\rightarrow$ 20, trained on 5 (prediction). For each sample, top row is real ground truth, bottom is predicted by our MCVD model. (We show only 2 conditional frames here) 160
7.9 BAIR 2 $\rightarrow$ 28, trained on 5 (prediction). For each sample, top row is real ground truth, bottom is predicted by our MCVD model. 161
7.10 UCF-101 4 $\rightarrow$ 16, trained on 4 (prediction). For each sample, top row is real ground truth, bottom is predicted by our MCVD model. 162
7.11 UCF-101 0 $\rightarrow$ 4 (generation) 162
7.12 Cityscapes: 2 $\rightarrow$ 7, trained on 5 (prediction); Conditioning on the two frames in the top left corner of each block of two rows of images, we generate the next 7 frames. The top row is the true frames, bottom row contains the generated frames. We use the MCVD concat model variant. 163

List of acronyms and abbreviations

BPD	Bits-per-dimension
CNF	Continuous Normalizing Flow
CNN	Convolutional Neural Network
DDPM	Denoising Diffusion Probabilistic Model
EDS	Expected Denoised Sample
FDP	Forward Diffusion Process
FID	Fréchet Inception Distance
FVD	Fréchet Video Distance
GAN	Generative Adversarial Network
GE	Geodesic Error
GFF	Gaussian Free Field
GPU	Graphics Processing Unit
ICML	International Conference on Machine Learning
IK	Inverse Kinematics
IS	Inception Score
IVP	Initial Value Problem
MCVD	Masked Conditional Video Diffusion
MLE	Maximum Likelihood Estimation
MPJPE	Mean Per Joint Position Error
MRCNF	Multi-Resolution Continuous Normalizing Flow
NeurIPS	Annual Conference on Neural Information Processing Systems
NI-DDPM	Non-Isotropic Denoising Diffusion Probabilistic Model
NI-SMLD	Non-Isotropic Score Matching Langevin Dynamics
NIVE	Non-Isotropic Variance Exploding
NIVP	Non-Isotropic Variance Preserving
ODE	Ordinary Differential Equations
OoD	Out-of-Distribution
PA-MPJPE	Procrustes-Aligned Mean Per Joint Position Error

PSNR	Peak Signal-to-Noise Ratio
RBN	Restricted Boltzmann Machine
RDP	Reverse Diffusion Process
RNN	Recurrent Neural Network
SDE	Stochastic Differential Equation
SGD	Stochastic Gradient Descent
SIGGRAPH	Special Interest Group on Computer Graphics and Interactive Techniques (annual conference on computer graphics organized by ACM SIGGRAPH)
SMLD	Score Matching Langevin Dynamics
SMPL	Skinned Multi-Person Linear model
SOTA	State-of-the-Art
SPADE	SPatially-Aaptive DE-normalization
SPATIN	SPAce-TIme-Adaptive Normalization
SSIM	Structural SIMilarity
VAE	Variational Autoencoder
VE	Variance Exploding
VP	Variance Preserving

Acknowledgements

I feel extremely privileged to have had the chance to pursue a PhD at Mila. I've learned an immense amount from amazing peers and mentors. I've made many wonderful friends over the course of my PhD, and I will always be grateful to everyone at this incredible lab.

Most importantly, I want to thank my advisor Chris for being the most supportive advisor one could ask for. His enthusiasm, positivity and optimism right from when I met him as a prospective student is something I hope to inculcate. He has been genuinely happy about my successes within and outside of Mila, and equally supportive during difficult times. I am grateful for his mentorship and guidance, which has been instrumental in shaping my career as a researcher. I would also like to thank Adam Oberman for his exceptional empathy and invaluable guidance. I am also grateful to Yoshua Bengio and Valerie Pisano (among so many others) for running a lab where one is surrounded by incredibly smart people, and providing us an environment where ideas are exchanged freely. Thank you Linda Peinthiere and Celine Bégin, without you, we'd be lost in an administrative maze.

I've also been fortunate to have been mentored by several amazing researchers during my internships. I'd like to thank Bryan Seybold and Sourish Chaudhuri at Google for being great mentors and allowing me to dip my toe in industry-applied research. I am grateful for their guidance during the internship, as well as their support beyond the program, including providing mock interview practice. I'd like to thank Boris Oreshkin and the DeepPose team including Florent Bocquelét, Louis-Simon Ménard, Félix Harvey, Jeremy Cowles and others for all their help and guidance during my internship at Unity Technologies. Boris has been a wonderful collaborator and mentor. I've immensely enjoyed all our conversations, and I feel privileged to have learned so much from him: his humility, his attitude of approaching problems with a can-do spirit, his skill of anticipating how our current research can be tied to future benefits for the organization. I'd like to thank Yashar Mehdad and Barlas Oğuz at Meta for giving me the resources and guidance to do large-scale text-to-3D research. It was inspiring to work with a team full of smart people who continued to perform excellent research despite only recently pivoting to computer vision. Their continual support throughout the internship helped me bridge the gap between academic research and industrial goals.

There were also several co-authors and friends at Mila who have contributed substantially to the articles in this thesis as well as to my own personal development, including but not limited to —- David Kanaa, Samira Kahou, Vincent Michalski for their humility and experienced insights; Sai Rajeshwar, Sandeep Subramanian for all the conversations over dosa; Tegan Maharaj for the parties and late night dinners; Anirudh Goyal, Krishna Murthy Jatavallabhula, Florian Golemo for their timely guidance and collaboration; Chris Finlay for his opportune collaboration and fun conversations; Roger Girgis, Martis Weiss, Mattie Tesfaldet, Chris Beckham for being the best friends one could ask for during this tumultuous journey; Alexia Jolicoeur-Martineau for being the best collaborator and co-author I have had the good fortune to work with. Thank you for giving me the opportunity to work with you, I've learned a lot from our interactions.

I thank Graham Taylor and Alexander Wong, who supported me during my PhD and made it easier to live and work outside of Montreal. I also thank Jason Deglint, Katie Thomas, and the BlueLion team for the wonderful conversations and fun meetups. I am grateful to Microsoft Research for granting me a Diversity Award so I could take care of important financial expenses.

Finally, I am indebted to my family, my father and mother for supporting me in all my endeavours in immeasurable and inconceivable ways throughout my life. I am also indebted to my wife, Kirti, who supported me through all my career aspirations, while excelling in her own PhD. Thank you for your patience and understanding, especially during times of uncertainty and self-doubt, and for being my partner in life.

Chapter 1

Introduction

1.1 Computer vision and machine learning

Computer vision is a field of study that aims to enable machines to understand and interpret visual information, including images and videos. The goal of computer vision systems is to replicate the capabilities of human vision, such as object detection, motion tracking, and understanding spatial relationships. This field has seen significant progress over the past several decades, especially due to machine learning in the recent past. In particular, generative modeling for computer vision has gained significant prominence in the past few years, and promotes a wide variety of applications.

Early work in computer vision focused on simple image processing techniques, such as edge detection and thresholding. In the 1960s and 1970s, researchers began to develop more sophisticated algorithms for image analysis, including pattern recognition and feature extraction. One of the most influential works from this era was the book "Digital Picture Processing" by A. Rosenfeld and A. C. Kak (1976) [Rosenfeld and Kak, 1969], which provided a comprehensive overview of image processing techniques.

In the 1980s and 1990s, computer vision research shifted towards more advanced techniques, such as machine learning and neural networks. The seminal book "Parallel Distributed Processing" by D. E. Rumelhart and J. L. McClelland (1986) [McClelland et al., 1986] introduced the idea of using neural networks for image analysis, which paved the way for the development of deep learning techniques in the 21st century.

Today, computer vision systems are used in a wide range of applications, from autonomous vehicles to medical imaging to facial recognition systems. Recent advances in deep learning have enabled machines to achieve human-level performance on many computer vision tasks, including object detection and recognition, image segmentation, and pose estimation.

The following sections introduce generative modeling in computer vision and applications. Relevant generative modeling techniques are discussed, with a special focus on two methods relevant to this thesis : continuous normalizing flows, and denoising diffusion models.

1.2 Generative modeling

Generative modeling is a sub-field of machine learning that aims to learn the underlying probability distribution of a given dataset, and use this knowledge to generate new, realistic samples that are similar to the original data. This has many applications, such as image and video synthesis, data augmentation, style transfer, etc.

One of the earliest and most popular generative models is the Restricted Boltzmann Machine (RBM), which was introduced by Smolensky in 1986 [Smolensky, 1986]. RBMs are a type of neural network that learn to model the joint probability distribution of the input data. RBMs have been used in a variety of applications, including image and speech recognition, recommendation systems, and collaborative filtering.

Another important early generative model is the Autoencoder, which was first introduced by Rumelhart et al. in 1986 [Rumelhart et al., 1986]. Autoencoders are neural networks that learn to encode input data into a lower-dimensional representation, and then decode this representation back into the original data. Autoencoders can be used for a variety of tasks, such as data compression, denoising, and anomaly detection.

In recent years, generative modeling has emerged as an exciting area of research. In the context of computer vision, generative models can be used to create realistic images, videos, and other visual media. Generative modeling for computer vision is typically based on deep learning architectures such as Generative Adversarial Networks (GANs) [Goodfellow et al., 2014], Variational Auto-Encoders (VAEs) [Kingma and Welling, 2013], Normalizing flows [Dinh et al., 2015, 2017], Continuous normalizing flows [Chen et al., 2018a], and Denoising diffusion models [Song and Ermon, 2019, Ho et al., 2020].

GANs were introduced in 2014 by Goodfellow et al. and have since become one of the most popular and widely used generative models for computer vision. GANs consist of two neural networks, a generator network and a discriminator network, that are trained together in a game-like setting. The generator network learns to create realistic images that can fool the discriminator network into thinking they are real, while the discriminator network learns to distinguish between real and fake images.

VAEs, on the other hand, are based on the idea of learning a low-dimensional representation of the data that can be used to generate new samples. They were introduced in 2014 by Kingma and Welling, and have since become a popular choice for generative modeling in computer vision. VAEs consist of an encoder network that learns to map input images to a latent space, and a decoder network that learns to generate new images from the latent space.

Normalizing flows are a family of generative models that learn to transform a simple distribution, such as a standard normal distribution, into a complex distribution that resembles the target distribution of the data. Normalizing flows have shown impressive results in image generation, and have been used to create high-resolution images.

The upcoming sections introduce two specific generative modeling techniques: Continuous normalizing flows, and Denoising diffusion Models. These techniques have been chosen due to their direct relevance to the contributions presented in the following chapters. Furthermore, conditional variants, and the diverse applications of generative models will be discussed.

1.2.1 Continuous normalizing flows

Continuous Normalizing Flows (CNFs) are another class of generative models that have shown great promise in recent years that leverage Neural Ordinary Differential Equations (Neural ODEs). Instead of specifying a fixed sequence of transformations like Normalizing Flows, Neural ODEs learn a continuous-time dynamics model that evolves over time by using a neural network to approximate the solution to an ordinary differential equation. This approach provides a flexible and adaptive way of modeling complex systems. Neural ODEs have been successfully applied to a range of applications, including image and speech generation, data imputation, and scientific simulations.

Continuous Normalizing Flows go a step further, and assume that on one end of these Neural ODE-based transformations is a simple distribution, such as a standard normal distribution, resulting in a flexible and expressive generative model. CNFs have been shown to be effective in a range of applications, including image and speech generation, and have the potential to be used in a wide range of scientific and engineering applications. However, they can be challenging to train, and there is ongoing research to develop more efficient and effective training algorithms for these models.

1.2.2 Denoising diffusion models

Denoising diffusion models are a relatively new class of generative models that learn a stochastic process that gradually transforms a known distribution, such as a standard normal distribution, into the target distribution of the data. Unlike normalizing flows which learn a series of invertible transformations, diffusion models leverage a series of noise processes that act as a stochastic diffusion process. This makes diffusion models well-suited for modeling complex high-dimensional distributions, and they have shown great promise in a range of applications, including image synthesis and denoising tasks.

The ability of diffusion models to generate high-quality data with realistic textures and fine details has been demonstrated in several recent studies [Ho et al., 2020, Chen et al., 2021a, Voleti et al., 2022a, Poole et al., 2023], where they have been used to generate images, audio, videos, and 3D objects that are virtually indistinguishable from real-world data. Furthermore, diffusion models have been shown to have a number of advantages over other generative models, including improved training stability, better likelihood estimation, and the ability to handle missing data. Despite these successes, there are still many challenges to be addressed in the development and application of diffusion models, and ongoing research is focused on improving their scalability, robustness, and efficiency.

1.3 Conditional generative modeling

Conditional generative modeling involves generating new data samples based on a given set of conditions. The conditioning aspect of the modeling refers to providing additional information to the generative model in the form of inputs or labels, which are used to guide the generation process. In other words, the model is trained to generate data samples that are conditioned on specific inputs or labels. This approach is particularly useful when the generated data needs to meet certain criteria, or follow specific patterns. For example, in image generation, the conditional inputs may include the desired object category, color, or size. The conditioning information can be incorporated into the model architecture in various ways, such as by adding an additional input layer, or by modifying the loss function to account for the conditioning information.

Text-to-image diffusion models are a type of conditional generative model that involves generating high-quality images based on textual descriptions [Rombach et al., 2022]. The conditioning information in this case includes the textual description of the image to be generated, such as the color, shape, and position of various objects. The generation process in text-to-image diffusion models involves iteratively refining the image based on the given textual input. One of the key advantages of text-to-image diffusion models is their ability to generate highly customized images based on specific textual inputs. This makes them particularly useful in applications such as e-commerce, where personalized images can be generated based on the customer's description of the product they want to purchase. This can be expanded to other modalities such as audio, video, etc.

1.4 Applications

Generative modeling for computer vision has many exciting applications, such as image and video synthesis, data augmentation, and style transfer. These models can be used to create new, realistic images and videos from scratch, and to manipulate existing visual media in creative ways. For example, generative models can be used to translate images into different styles [Gatys et al., 2016] such as converting a photo into a painting or a sketch, or to create realistic animations of objects and scenes [Unterthiner et al., 2018, Voleti et al., 2022a].

Another important application of generative modeling in computer vision is in data augmentation. Data augmentation refers to the process of creating new training samples by applying random transformations to existing samples. This technique enhances deep learning models by expanding the training set and boosting performance. Generative models can generate new, realistic training samples to augment the training set [Zhang et al., 2021], potentially improving the accuracy and robustness of the model.

One of the most prominent applications of image, 3D, and video generation is in the entertainment industry. The ability to generate realistic images and animations has enabled filmmakers and game developers to create stunning visual effects and immersive worlds that were previously impossible to achieve using traditional methods [Mildenhall et al., 2020].

In advertising, image and video generation enables realistic product simulations and virtual try-on experiences for customers [Jiang et al., 2022]. This can enhance sales and reduce returns by allowing customers to visualize the product's fit before making a purchase.

In healthcare, image and video generation can be used for a variety of applications, such as training medical professionals and developing new treatments. For example, medical simulations can be generated to train surgeons and other healthcare professionals in complex procedures, while virtual environments can be created to simulate and test new medical treatments before they are used on patients [Mirchi et al., 2020].

Overall, conditional generative modeling is a powerful technique for generating highly customized and realistic data samples. It has broad applications across various industries and fields. The ability to create realistic images, 3D objects, and videos based on specific conditions brings numerous benefits. As generative modeling techniques advance, we anticipate even more exciting applications in image, 3D, and video generation in the future.

1.5 Thesis overview

This thesis aims to explore various aspects of conditional generative modeling. The following is a list of peer-reviewed publications based on these contributions:

- *NeurIPS 2022* - "MCVD: Masked Conditional Video Diffusion for Prediction, Generation, and Interpolation", V. Voleti, A. Jolicoeur-Martineau, C. Pal arXiv
- *NeurIPS 2022 Workshop* - "Score-based Denoising Diffusion with Non-Isotropic Gaussian Noise Models", V. Voleti, C. Pal, A. Oberman arXiv
- *SIGGRAPH Asia 2022* - "SMPL-IK: Learned Morphology-Aware Inverse Kinematics for AI-Driven Artistic Workflows", V. Voleti, B. N. Oreshkin, F. Bocquelet, F. G. Harvey, L. Ménard, C. Pal arXiv
- *ICML 2021 Workshop* - "Improving Continuous Normalizing Flows using a Multi-Resolution Framework", V. Voleti, C. Finlay, A. Oberman, C. Pal
- *NeurIPS 2019 Workshop* - "Simple Video Generation using Neural ODEs", V. Voleti*, D. Kanaa*, S. E. Kahou, C. Pal arXiv

Through a comprehensive literature review, original research, and analysis, this thesis provides an understanding of the application of (conditional) generative modeling in the domains of:

- images: Chapters 4 and 6,

- 3D animations: Chapter 5, and
- video: Chapters 3 and 7.

The chapters are arranged in the chronological order of publication of the respective articles. Chapter 2 provides essential background information that lays the foundation for subsequent chapters. Chapter 3 employs Neural Ordinary Differential Equations (Neural ODEs) in an encoder-decoder framework. Chapter 4 builds on Continuous Normalizing Flows (CNFs), a development on Neural ODEs. Chapter 5 utilizes an encoder-decoder framework for 3D pose estimation. Chapters 6 and 7 further develop and employ denoising diffusion models.

Chapter 2 consists of relevant background information that is useful for the later chapters. Section 2.2 introduces Neural ODEs, and Section 2.3 presents Continuous Normalizing Flows (CNFs). These concepts shall be helpful to understand Chapters 3 and 4. Section 2.4 introduces denoising diffusion models, including relevant mathematical details. It involves the derivations of two broad streams of diffusion models called Denoising Diffusion Probabilistic Models (DDPM) (Section 2.4) and Score-Matching Langevin Dynamics (SMLD) (Section 2.5). These concepts shall be helpful to understand Chapters 6 and 7.

Chapter 3 describes our novel application of Neural ODEs to video prediction. Section 3.2 details relevant prior work. Section 3.3 provides an overview of Neural ODEs. Section 3.4 explains our method of using Neural ODEs to model the dynamics of a video. Section 3.5 demonstrates the application of our method on the MovingMNIST dataset. Section 3.6 discusses the limitations of our method and future work. This work was published at a workshop at NeurIPS 2019.

Chapter 4 derives a conditional variant of continuous normalizing flows called Multi-Resolution Continuous Normalizing Flows (MRCNF), and applies it to image generation. Section 4.2 details background information, and Section 4.3 builds on them to derive MRCNF. Section 4.4 reviews prior related works, with a focus on WaveletFlow in Section 4.4.1. Section 4.5 presents experimental results on image generation, showcasing MRCNF's efficiency in terms of number of parameters and training time. It also includes various conclusions from ablation studies. Section 4.7 analyzes the out-of-distribution (OoD) properties of MRCNFs, and shows that they are similar to those of other normalizing flows. This work was published as a workshop paper at ICML 2021, and a version is currently under review for a journal.

Chapter 5 focuses on incorporating a 3D human pose prior in inverse kinematics i.e. estimation of full 3D pose from partial inputs. Section 5.2 expands on relevant background information. Section 5.3 presents our learned morphology-aware inverse kinematics module involving an encoder-decoder architecture that is flexible in input space. Section 5.4 details our proposed method to apply the estimated pose on non-human 3D characters. Section 5.5 presents our proposed artistic workflow for 3D scene authoring from an image. Section 5.6 provides details on the neural network architecture, training, and evaluation. Sections 5.7

and 5.8 report experimental results and limitations of our method, and Section 5.9 provides demo videos. This work was published at SIGGRAPH Asia 2022.

Chapter 6 presents a novel mathematical derivation of Non-Isotropic Denoising Diffusion Models, a variant of denoising diffusion models that uses an underlying non-isotropic Gaussian noise model. Section 6.2 derives Non-Isotropic DDPM (NI-DDPM), and Section 6.3 derives Non-Isotropic SMLD (NI-SMLD). Figure 6.2 is dedicated to a direct comparison between DDPM and NI-DDPM. Section 6.5 presents an instantiation of a non-isotropic noise process called Gaussian Free Fields (GFF), and Section 6.6 reports results from experiments on image generation from GFFs using NIDDPM. This work was published as a workshop paper at NeurIPS 2022.

Chapter 7 presents a novel application of denoising diffusion models to modeling video in a conditional fashion. Section 7.2 derives the use of diffusion models to solve three video-related tasks : video prediction, video generation, video interpolation, all using a single model. Section 7.2.4 explains the model architecture used. Section 7.3 provides a direct comparison with prior relevant methods, and discusses other prior methods. Section 7.4 details state-of-the-art results on all tasks, and Section 7.5.4 mentions various ablation studies conducted. Section 7.7 provides qualitative results of generated videos. This work was published as a conference paper at NeurIPS 2022.

Overall, I hope this thesis provides a comprehensive presentation of these key modern methods for constructing conditional generative models for images, 3D animations, and video, and their potential impact on the field of computer vision and deep learning.

Chapter 2

Background

This chapter introduces the relevant details of ODEs (Section 2.1), Neural ODEs (Section 2.2), and Continuous Normalizing Flows (Section 2.3). These concepts will be helpful for Chapters 3 and 4. It then introduces the main mathematical details of DDPM [Ho et al., 2020] (Section 2.4) and SMLD [Song and Ermon, 2019] (Section 2.5). These concepts will be helpful for Chapters 6 and 7.

2.1 Ordinary Differential Equations (ODEs)

Ordinary Differential Equations (ODEs) are mathematical tools used to model dynamic systems that evolve over time [Courant et al., 1965, Apostol, 1991, Strang, 1991]. They are used to describe phenomena such as population growth, chemical reactions, motion of particles, heat transfer, and electrical circuits, to name a few, and have a wide range of applications in various fields of science and engineering, including physics, chemistry, biology, economics, and engineering. ODEs come in various forms, from simple linear equations to complex non-linear systems, and can be solved analytically or numerically. Many numerical methods have been developed to solve ODEs, making them an essential tool for modeling and simulating various dynamical systems in a variety of fields.

2.1.1 Initial Value Problem (IVP)

Typically, the context of ODEs begins with the Initial Value Problem. For a state $x(t)$ that changes dynamically with time t, if the rate of change is described by a function f, and if the initial value of the state x at time t_0 is given, then what is the value of $x(t)$ at some other time step t_1?

$$\frac{dx(t)}{dt} = f(x(t), t); \quad x(t_0) \text{ is given}; \quad x(t_1) = \,? \tag{2.1.1}$$

Here, f is called the differential of x with respect to time t. Many physical processes follow this template of the Initial Value Problem (IVP).

The solution to the Initial Value Problem is:

$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} f(x(t), t)\ dt. \tag{2.1.2}$$

As an example,

$$\frac{dx}{dt} = 2t;\ x(0) = 2;\ x(1) = \ ? \tag{2.1.3}$$

$$\begin{aligned} \implies x(1) &= x(0) + \int_0^1 2t\ dt, \\ &= x(0) + (t^2|_{t=1} - t^2|_{t=0}), \\ &= 2 + 1^2 - 0^2, \\ &= 3. \end{aligned} \tag{2.1.4}$$

2.1.2 Numerical integration

The above example involved the use of analytical integration for $\int_{t_0}^{t_1} f(x(t), t)\ dt$. However, in some cases, it may not be possible to use simple integration to estimate the solution. For example, the solution to the following IVP requires some non-trivial simplification:

$$\frac{dx}{dt} = 2xt\ ;\ x(0) = 3;\ x(1) = \ ? \tag{2.1.5}$$

$$\implies \int \frac{1}{2x}\ dx = \int t\ dt,$$

$$\implies \frac{1}{2}\log x = \frac{1}{2}t^2 + c_0,$$

$$\implies x(t) = ce^{t^2}. \tag{2.1.6}$$

$$\text{We know that } x(0) = 3 \implies c = 2,$$

$$\implies x(t) = 2e^{t^2},$$

$$\implies x(1) = 5.436. \tag{2.1.7}$$

Hence, the analytic solution is:

$$\underline{\text{Analytic solution}}:\ x(t) = 2e^{t^2} \implies x(1) = 5.436. \tag{2.1.8}$$

In such cases, in order to automate the integration process in a computer, the integration is approximated using Numerical Integration methods, or ODE Solvers. There are several ODE Solvers, the simplest for them being the Euler method. This is illustrated in Figure 2.1, and is described below:

$$\begin{cases} t_{n+1} &= t_n + h, \\ x(t_{n+1}) &= x(t_n) + h\ f(x(t_n), t_n). \end{cases} \tag{2.1.9}$$

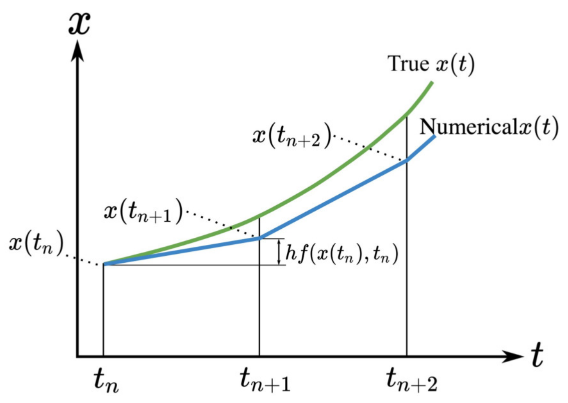


Fig. 2.1. Illustration of 1st-order Runge-Kutta / Euler's method of numerical integration.

Hence, using Euler's method in Equation (2.1.9), the numerical solution to the above problem is:

<u>Numerical solution</u>:

$$\text{Let } h = 0.25. \tag{2.1.10}$$

$$\begin{aligned} x(0.25) &= x(0) + 0.25 * f(x(0), 0), \\ &= 3 + 0.25 * (2 * 3 * 0), \\ &= 3. \end{aligned} \tag{2.1.11}$$

$$\begin{aligned} x(0.5) &= x(0.25) + 0.25 * f(x(0.25), 0.25), \\ &= 3 + 0.25 * (2 * 3 * 0.25), \\ &= 3.375. \end{aligned} \tag{2.1.12}$$

$$\begin{aligned} x(0.75) &= x(0.5) + 0.25 * f(x(0.5), 0.5), \\ &= 3.375 + 0.25 * (2 * 3.375 * 0.5), \\ &= 4.21875. \end{aligned} \tag{2.1.13}$$

$$\begin{aligned} x(1) &= x(0.75) + 0.25 * f(x(0.75), 0.75), \\ &= 4.21875 + 0.25 * (2 * 4.21875 * 0.75), \\ &= 5.8008. \end{aligned} \tag{2.1.14}$$

We can see that the analytic solution 5.436 and the numerical solution 5.8008 are not equal. Indeed, the numerical solution is only an approximation of the analytical solution. Better approximations are obtained using higher order methods such as Runge-Kutta methods [Pontyagin et al., 1962, Kutta, 1901, Hairer et al., 2000], multi-step algorithms such as the Adams-Moulton-Bashforth methods [Butcher, 2016, Hairer et al., 2000, Quarteroni et al., 2000], etc. More advanced algorithms provide better control over the approximation error and the accuracy [Press et al., 2007]. For example, the 4th order Runge-Kutta method is used as the default ODE solver in many applications:

$$\begin{cases} t_{n+1} &= t_n + h, \\ s_1 &= f(x(t_n),\, t_n), \\ s_2 &= f(x(t_n) + \frac{h}{2}s_1,\, t_n + \frac{h}{2}), \\ s_3 &= f(x(t_n) + \frac{h}{2}s_2,\, t_n + \frac{h}{2}), \\ s_4 &= f(x(t_n) + hs_3,\, t_n + h), \\ x(t_{n+1}) &= x(t_n) + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4). \end{cases} \tag{2.1.15}$$

Hence, we shall now use "ODESolve" as a placeholder for the user's choice of ODE Solver. The solution to the Initial Value Problem is thus:

$$x(t_1) = \text{ODESolve}(\ f(x(t), t),\ x(t_0),\ t_0,\ t_1\ ), \tag{2.1.16}$$

where $x(t_1)$ is the value of the state x to be estimated at time step t_1, $x(t_0)$ is the initial value of x at initial time step t_0, and f is the differential function of x.

Suppose f is continuously differentiable. Then, considering the solution curves as plotted on a plane with time t on one axis and state x on the other axis/axes,

(1) the solution curves for this differential equation completely fill the plane, and

(2) the solution curves of different solutions do not intersect.

This means the solution of an ODE is a *flow*, it involves non-intersecting solution curves. This is illustrated in Figure 2.2, as provided in Yan et al. [2020].

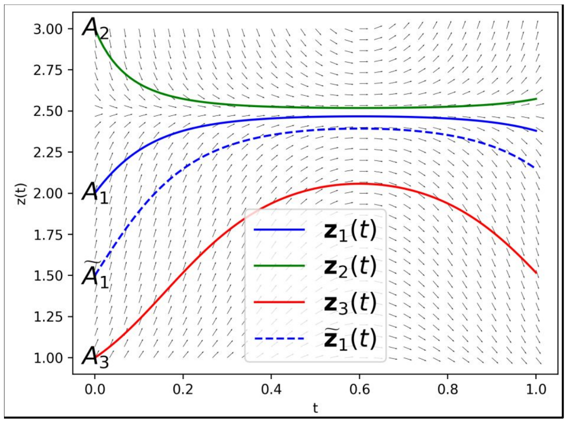


Fig. 2.2. Illustration of the solution to ODEs being flows (taken from Yan et al. [2020]).

2.2 Neural Ordinary Differential Equations (Neural ODEs)

Neural ODEs formulate ODEs such that the differential function is a neural network [Chen et al., 2018a]. This represents a paradigm shift in the way ODEs were typically solved. Whereas earlier ODEs governing natural phenomena were hand-designed, now using the framework of Neural ODEs, the ODE could be learnt through backpropagation into a parameterized neural network.

Suppose the problem of classification is taken up. Given an input data point $\mathbf{x}(t_0)$, it is transformed into a feature $\mathbf{x}(t_1)$ using a Neural ODE f_θ parameterized by θ. Then, a classification loss function is applied on the feature. In general, any objective function L can be applied on $\mathbf{x}(t_1)$. Then, the neural network is trained through backpropagation to update its parameters θ to minimize the loss L.

There are parallels that could be drawn with a residual network, if the residual network shares its parameters across all layers. This is illustrated in Figure 2.3.

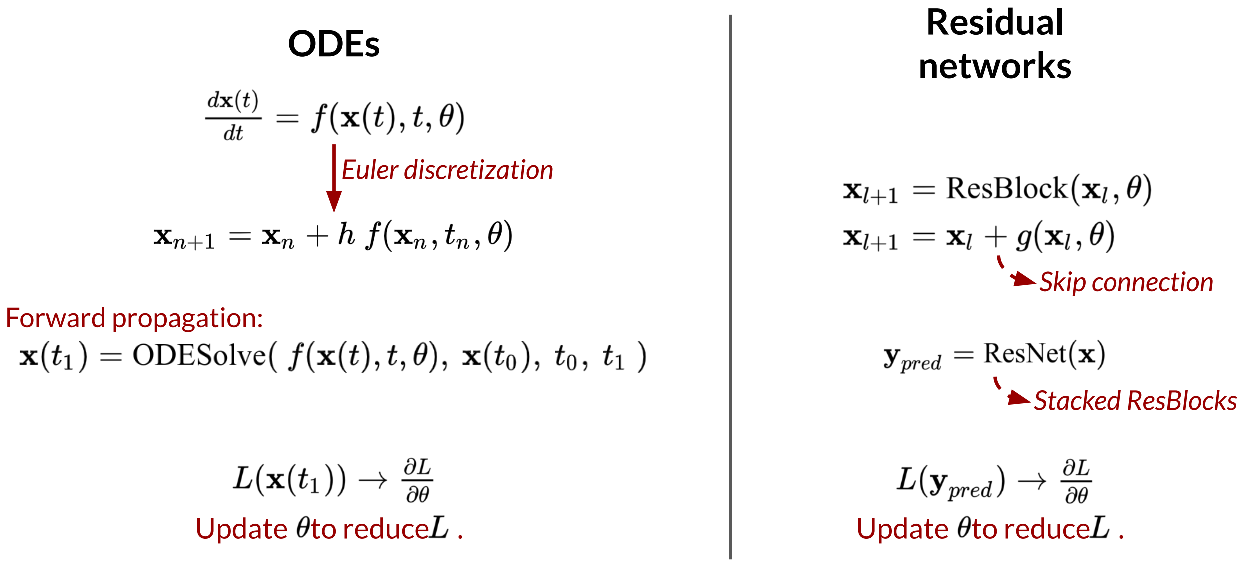

ODEs
$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), t, \theta)$
Euler discretization
$\mathbf{x}_{n+1} = \mathbf{x}_n + h\, f(\mathbf{x}_n, t_n, \theta)$
Forward propagation:
$\mathbf{x}(t_1) = \text{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ )$
$L(\mathbf{x}(t_1)) \rightarrow \frac{\partial L}{\partial \theta}$
Update θ to reduce L .
Residual networks
$\mathbf{x}_{l+1} = \text{ResBlock}(\mathbf{x}_l, \theta)$
$\mathbf{x}_{l+1} = \mathbf{x}_l + g(\mathbf{x}_l, \theta)$
Skip connection
$\mathbf{y}_{pred} = \text{ResNet}(\mathbf{x})$
Stacked ResBlocks
$L(\mathbf{y}_{pred}) \rightarrow \frac{\partial L}{\partial \theta}$
Update θ to reduce L .

Fig. 2.3. Illustration of the solution to ODEs being flows.

2.2.1 Adjoint method

However, computing the gradient of the loss L with respect to the parameters θ i.e. $\partial L/\partial\theta$, incurs a high memory cost, since all activations of all iterations of ODESolve need to be stored in memory to complete backpropagation. This is sub-optimal, can we do better?

It turns out that there is a way to compute $\partial L/\partial\theta$ without having to save all activations from t_0 to t_1. This is made possible by what is called the "adjoint method" [Pontyagin et al., 1962]. The adjoint $\mathbf{a}(t)$ of a state $\mathbf{x}(t)$ is defined as:

$$\text{adjoint } \mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{x}}, \tag{2.2.1}$$

$$\frac{d\mathbf{a}}{dt} = -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t),\, t,\, \theta)}{\partial \mathbf{x}}. \tag{2.2.2}$$

The adjoint $a(t_1)$ can be computed from the loss L and the final state $\mathbf{x}(t_1)$ after one forward propagation pass through the ODESolve function:

$$\mathbf{x}(t_1) = \text{ODESolve}(\ f(\mathbf{x}(t), t, \theta),\ \mathbf{x}(t_0),\ t_0,\ t_1\ ), \tag{2.2.3}$$

$$\implies \mathbf{a}(t_1) = \frac{\partial L}{\partial \mathbf{x}(t_1)}. \tag{2.2.4}$$

Then, Pontyagin et al. [1962] showed that $\partial L / \partial \theta$ can be computed by solving another ODE involving the adjoint, in the reverse direction from t_1 to t_0:

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t),\, t,\, \theta)}{\partial \theta}\ dt. \tag{2.2.5}$$

This can be computed using our ODESolve function from t_1 to t_0, with the initial value as $\mathbf{0}$, and the differential as defined in eq. (2.2.5):

$$\frac{\partial L}{\partial \theta} = \text{ODESolve}(-\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t),\, t,\, \theta)}{\partial \theta},\quad \mathbf{0}_{|\theta|}\ \ ,\ t_1,\ t_0). \tag{2.2.6}$$

However, for this ODESolve to work, the values of $\mathbf{a}(t)$ and $\mathbf{x}(t)$ at all intermediate steps of numerical integration are required. Hence, two other ODESolves are performed from t_1 to t_0: one to compute $\mathbf{a}(t)$, and the other for $\mathbf{x}(t)$. The initial values for these ODESolves are $\mathbf{a}(t_1)$ and $\mathbf{x}(t_1)$, which were computed in the forward pass in Equations (2.2.3) and (2.2.4):

$$\mathbf{x}(t_0) = \text{ODESolve}(\qquad f(\mathbf{x}(t), t, \theta) \qquad,\ \mathbf{x}(t_1),\ t_1,\ t_0). \tag{2.2.7}$$

$$\mathbf{a}(t_0) = \text{ODESolve}(-\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t),\, t,\, \theta)}{\partial \mathbf{x}},\ \mathbf{a}(t_1),\ t_1,\ t_0). \tag{2.2.8}$$

These three ODESolves can be combined into a single ODESolve:

$$\begin{bmatrix} \mathbf{x}(t_0) \\ \mathbf{a}(t_0) \\ \frac{\partial L}{\partial \theta} \end{bmatrix} = \text{ODESolve}\left(\begin{bmatrix} f(\mathbf{x}(t), t, \theta) \\ -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t),\, t,\, \theta)}{\partial \mathbf{x}} \\ -\mathbf{a}(t)^\top \frac{\partial f(\mathbf{x}(t),\, t,\, \theta)}{\partial \theta} \end{bmatrix},\ \begin{bmatrix} \mathbf{x}(t_1) \\ \mathbf{a}(t_1) \\ \mathbf{0}_{|\theta|} \end{bmatrix},t_1,t_0 \right). \tag{2.2.9}$$

Thus, at the end of ODESolve, $\partial L / \partial \theta$ is directly obtained in the third part of the augmented state. $\partial L / \partial \theta$ is then used to update θ using gradient descent, thus training the Neural ODE.

Because a Neural ODE ultimately describes an ODE, the fundamental theorem of ODEs applies to the solution f: Neural ODEs describe a homeomorphism/*flow* i.e. they preserve dimensionality, and they form non-intersecting solution trajectories. Moreover, Neural ODEs are reversible architectures : the same ODE can be solved forwards or backwards.

2.3 Continuous Normalizing Flows (CNFs)

Neural ODEs can then be used in a generative modeling framework by setting the final state to be a sample from a known distribution, typically a noise distribution such as the standard normal. Thus, a Neural ODE can then be trained to map between a data distribution and the normal distribution. Since a Neural ODE describes a (geometric) flow, and in this framework is used to map to the normal distribution, this framework is a *normalizing flow*. To distinguish it from the usual normalizing flow in literature [Dinh et al., 2017], considering the fact that it operates in continuous space while normalizing flows operate in discrete steps, this framework is called "Continuous Normalizing Flow" (CNF).

Suppose a CNF g transforms its state $\mathbf{v}(t)$ using a Neural ODE, with the differential defined by the neural network f parameterized by θ. $\mathbf{v}(t_0) = \mathbf{x}$ is, say, an image, and at the final time step $\mathbf{v}(t_1) = \mathbf{z}$ is a sample from a known noise distribution.

$$\frac{\mathrm{d}\mathbf{v}(t)}{\mathrm{d}t} = f(\mathbf{v}(t), t, \theta), \tag{2.3.1}$$

$$\implies \mathbf{v}(t_1) = g(\mathbf{v}(t_0)) = \mathbf{v}(t_0) + \int_{t_0}^{t_1} f(\mathbf{v}(t), t, \theta)\ \mathrm{d}t, \tag{2.3.2}$$

$$\implies \mathbf{z} = g(\mathbf{x}) = \mathbf{x} + \int_{t_0}^{t_1} f(\mathbf{v}(t), t, \theta)\ \mathrm{d}t. \tag{2.3.3}$$

This integration is typically performed by an ODE solver, as shown in Equation (2.1.16). Since this integration can be run backwards as well to obtain the same $\mathbf{v}(t_0)$ from $\mathbf{v}(t_1)$, a Continuous Normalizing Flows (CNF) is a reversible model.

CNFs can now be trained by maximizing the likelihood of real data points under the model. This is equivalent to transforming the real data point $\mathbf{v}(t_0)$ to the the final state $\mathbf{v}(t_1)$, and maximizing the likelihood of $\mathbf{v}(t_1)$ under the standard normal distribution. Given this maximum likelihood objective function, the neural network f in the CNF can be optimized as described in Section 2.2.1.

The change-of-variables formula describes how the likelihood of the real data point $\mathbf{x}$ can be computed as a combination of the likelihood of the final point $\mathbf{z}$ and the change in the likelihood under the CNF transformation g:

$$\log p(\mathbf{x}) = \log \left| \det \frac{\partial g}{\partial \mathbf{v}} \right| + \log p(\mathbf{z}) = \Delta \log p_{\mathbf{v}(t_0) \to \mathbf{v}(t_1)} + \log p(\mathbf{z}). \tag{2.3.4}$$

The second term, $\log p(\mathbf{z})$, is computed as the log probability of $\mathbf{z}$ under a known noise distribution, typically the standard normal $\mathcal{N}(\mathbf{0}, \mathbf{I})$. However, the log determinant of the Jacobian (first term on the right) is often intractable. Previous works on normalizing flows have found some ways to estimate this efficiently.

Chen et al. [2018a] and Grathwohl et al. [2019] instead proposed a more efficient variant in the CNF context, the instantaneous change-of-variables formula:

$$\frac{\partial \log p(\mathbf{v}(t))}{\partial t} = -\mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{v}(t)}\right), \tag{2.3.5}$$

$$\implies \Delta \log p_{\mathbf{v}(t_0)\to\mathbf{v}(t_1)} = \int_{t_0}^{t_1} -\mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{v}(t)}\right) \mathrm{d}t. \tag{2.3.6}$$

Here, "Tr" implies the trace operation i.e. the sum of the diagonal elements in the matrix.

Hence, the change in log-probability of the state of the Neural ODE i.e. $\Delta \log p_{\mathbf{v}(t_0)\to\mathbf{v}(t_1)}$ is expressed as another differential equation. The ODE solver now solves both differential equations Equations (2.3.3) and (2.3.6) by augmenting the original state. Thus, a CNF forward pass provides both the final state $\mathbf{v}(t_1)$ as well as the change in log probability $\Delta \log p_{\mathbf{v}(t_0)\to\mathbf{v}(t_1)}$ together:

$$\begin{bmatrix} \mathbf{z} \\ \Delta \log p_{\mathbf{v}(t_0)\to\mathbf{v}(t_1)} \end{bmatrix} = \mathrm{ODESolve}\left(\begin{bmatrix} f(\mathbf{v}(t), t, \theta) \\ -\mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{v}(t)}\right) \end{bmatrix}, \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}, t_0, t_1\right). \tag{2.3.7}$$

Thus, having estimated $\mathbf{z}$ and $\Delta \log p_{\mathbf{v}(t_0)\to\mathbf{v}(t_1)}$, $\log p(\mathbf{x})$ can be computed using Equation (2.3.4). Taking the objective function to be maximizing $\log p(\mathbf{x})$ i.e. maximizing the likelihood of real data under the model, the CNF can be trained as described in Section 2.2.1.

2.4 Denoising Diffusion Probabilistic Models (DDPM)

The formulations related to DDPM [Ho et al., 2020] are introduced here. Chapter 6 builds on this and derives a formulation with non-isotropic DDPM. Chapter 7 applies DDPM to video prediction, generation and interpolation.

DDPM focuses on modeling the diffusion process of noisy samples to approximate the clean data distribution. In its practical implementation, DDPM uses a neural network to estimate the noise to be subtracted from the noisy data to make it cleaner. At training time, a noise sample is added to a clean data sample, and the neural network is trained to predict the noise sample from the noisy data. At sample time, a noise sample is iteratively cleaned little by little, by estimating the noise to be subtracted at each step.

2.4.1 Forward (data to noise) for DDPM

In DDPM, for a fixed sequence of positive scales $0 < \beta_1 < \cdots < \beta_L < 1$, $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a noise sample from a standard normal distribution $\mathcal{N}$ with zero mean $\mathbf{0}$ and identity covariance matrix $\mathbf{I}$, and $\mathbf{x}_0$ is a clean data point, the **transition "forward"** noising process is:

$$p_{\beta_t}^{\mathrm{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}), \tag{2.4.1}$$

$$\implies \mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t \tag{2.4.2}$$

Then, the **cumulative "forward"** noising process can be derived as:

$$p_t^{\mathrm{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = p_{\beta_t}^{\mathrm{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_{t-1})\ p_{\beta_t}^{\mathrm{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_{t-2})\ \cdots\ p_{\beta_t}^{\mathrm{DDPM}}(\mathbf{x}_1 \mid \mathbf{x}_0). \tag{2.4.3}$$

Using $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$, the cumulative "forward" noising process can be simplified to:

$$p_t^{\mathrm{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}), \tag{2.4.4}$$

$$\implies \mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{2.4.5}$$

$$\implies \boldsymbol{\epsilon} = \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}. \tag{2.4.6}$$

2.4.2 Score for DDPM

Then, the **score** i.e. $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)$ can be calculated as:

$$\log p_t^{\mathrm{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = \log(\mathrm{const}) - \frac{1}{2(1-\bar{\alpha}_t)}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^T(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0), \tag{2.4.7}$$

$$\implies \text{Score } \ \mathbf{s} = \nabla_{\mathbf{x}_t} \log p_t^{\mathrm{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = -\frac{1}{(1-\bar{\alpha}_t)}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0), \tag{2.4.8}$$

$$= -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\left[\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}\right] = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}. \tag{2.4.9}$$

2.4.3 Score-matching objective function for DDPM

The score-matching objective for DDPM at noise level t is the expected Mean Square Error (MSE) between the true score in eq. (2.4.8), and the predicted score from a neural network $\mathbf{s}_{\boldsymbol{\theta}}$:

$$\ell^{\mathrm{DDPM}}(\boldsymbol{\theta}; \bar{\alpha}_t) \triangleq \frac{1}{2}\mathbb{E}_{p_t(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\|\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t) + \frac{1}{(1-\bar{\alpha}_t)}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)\right\|_2^2\right]. \tag{2.4.10}$$

$\mathbf{s}_{\boldsymbol{\theta}}$ predicts the score from the noisy image $\mathbf{x}_t$, and the noise level $\bar{\alpha}_t$ (or just the time step t).

The overall loss is the weighted sum of the losses at each step:

$$\mathcal{L}^{\mathrm{DDPM}}(\boldsymbol{\theta}; \{\bar{\alpha}_t\}_{t=1}^{L}) \triangleq \mathbb{E}_t\ \lambda(\bar{\alpha}_t)\ \ell(\boldsymbol{\theta}; \bar{\alpha}_t). \tag{2.4.11}$$

To let the loss have equal weight across all noise levels, the weight $\lambda(\bar{\alpha}_t)$ is the inverse of the variance of the true score at that noise level.

2.4.4 Variance of score for DDPM

$$\begin{aligned}\mathbb{E}\left[\left\|\nabla_{\mathbf{x}_t}\log p_t^{\mathrm{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0)\right\|_2^2\right] &= \mathbb{E}\left[\left\|-\frac{1}{(1-\bar{\alpha}_t)}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)\right\|_2^2\right], \\ = \mathbb{E}\left[\left\|\frac{\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}{(1-\bar{\alpha}_t)}\right\|_2^2\right] &= \frac{1}{1-\bar{\alpha}_t}\mathbb{E}\left[\|\boldsymbol{\epsilon}\|_2^2\right] = \frac{1}{1-\bar{\alpha}_t}.\end{aligned} \tag{2.4.12}$$

2.4.5 Overall objective function for DDPM

The overall objective function in Ho et al. [2020] used the inverse of the variance of the score at each time step (from eq. (2.4.12)) as the weight $\lambda(\bar{\alpha}_t)$:

$$\lambda^{\mathrm{DDPM}}(\bar{\alpha}_t) \propto 1/\mathbb{E}\Big[\left\|\nabla_{\mathbf{x}_t}\log p_t^{\mathrm{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0)\right\|_2^2\Big], \tag{2.4.13}$$

$$\implies \lambda^{\mathrm{DDPM}}(\bar{\alpha}_t) = 1 - \bar{\alpha}_t. \tag{2.4.14}$$

Then, the overall objective in eq. (2.4.11) changes to:

$$\mathcal{L}^{\mathrm{DDPM}}(\boldsymbol{\theta}; \{\bar{\alpha}_t\}_{t=1}^{L}) \triangleq \mathbb{E}_{t,p_t(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\|\sqrt{1-\bar{\alpha}_t}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t) + \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)}{\sqrt{1-\bar{\alpha}_t}}\right\|_2^2\right], \tag{2.4.15}$$

$$= \mathbb{E}_{t,p_t(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\|\sqrt{1-\bar{\alpha}_t}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t) + \boldsymbol{\epsilon}\right\|_2^2\right]. \tag{2.4.16}$$

2.4.6 Noise-matching objective for DDPM

Upon inspection of eq. (2.4.9), one can recognize that the score is a $-1/\sqrt{1-\bar{\alpha}_t}$ factor of $\boldsymbol{\epsilon}$, hence only $\boldsymbol{\epsilon}$ needs to be estimated:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t). \tag{2.4.17}$$

In this case, the overall objective function changes to the noise-matching objective:

$$\begin{aligned}\mathcal{L}^{\text{DDPM}}(\boldsymbol{\theta}; \{\bar{\alpha}_t\}_{t=1}^{L}) &\triangleq \mathbb{E}_{t,\boldsymbol{\epsilon},\mathbf{x}_0}\left[\|-\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t) + \boldsymbol{\epsilon}\|_2^2\right], \\ &= \mathbb{E}_{t,\boldsymbol{\epsilon},\mathbf{x}_0}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \bar{\alpha}_t)\right\|_2^2\right]. \qquad (2.4.18)\end{aligned}$$

This eq. (2.4.18) is Equation 14 in the DDPM paper [Ho et al., 2020]. The DDPM paper [Ho et al., 2020] retains conditioning of $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ on $\bar{\alpha}_t$ (or just t), but the SMLD paper [Song and Ermon, 2019] omits it.

2.4.7 Reverse (noise to data) in DDPM

The goal is to estimate the **reverse** transition probability $q_t^{\text{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$. However, this is intractable to compute, but it is possible to estimate it conditioned on $\mathbf{x}_0$, using Bayes' theorem:

$$\begin{aligned}q_t^{\text{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) &= \frac{q_t^{\text{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \quad q_t^{\text{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q_t^{\text{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0)}, \qquad (2.4.19) \\ &= \frac{\mathcal{N}(\mathbf{x}_t \mid \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad \mathcal{N}(\mathbf{x}_{t-1} \mid \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\mathbf{I})}{\mathcal{N}(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})}. \qquad (2.4.20)\end{aligned}$$

This can be simplified to:

$$\begin{aligned}&\implies q_t^{\text{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1} \mid \tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_{t-1}\mathbf{I}), \text{ where} \\ &\quad \tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t; \quad \tilde{\beta}_{t-1} = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \qquad (2.4.21) \\ &\implies \mathbf{x}_{t-1} = \tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \mathbf{x}_0) + \tilde{\beta}_{t-1}\mathbf{z} \quad \text{where } \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ is a noise sample.} \qquad (2.4.22)\end{aligned}$$

Given $\mathbf{x}_t$, eq. (2.4.5) is used to estimate $\mathbf{x}_0$ by first estimating noise $\boldsymbol{\epsilon}$ using a **neural network** $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$:

$$\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)). \qquad (2.4.23)$$

[$\because$ $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$ from eq. (2.4.5), and loss is minimized when $\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t) = \boldsymbol{\epsilon}$.]

Hence, using $\hat{\mathbf{x}}_0$ estimated from $\mathbf{x}_t$ using eq. (2.4.23), $\mathbf{x}_{t-1}$ is computed from eq. (2.4.22) as:

$$\mathbf{x}_{t-1} = \left[\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\mathbf{x}}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t\right] + \sqrt{\tilde{\beta}_{t-1}}\mathbf{z}. \qquad (2.4.24)$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is a noise sample.

2.4.8 Sampling in DDPM

Ho et al. [2020] splits sampling at each time step into 2 steps:

$$\textit{Step 1 (from eq. (2.4.23)):}\ \hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t)). \qquad (2.4.25)$$

$$\textit{Step 2 (from eq. (2.4.24)):}\ \mathbf{x}_{t-1} = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\mathbf{x}}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \sqrt{\tilde{\beta}_{t-1}}\mathbf{z}_{t-1}. \tag{2.4.26}$$

This can be simplified as:

$$\begin{aligned}
\implies \tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \hat{\mathbf{x}}_0) &= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\left(\frac{1}{\sqrt{\bar{\alpha}_t}}\Big(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)\Big)\right) + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t, \\
&= \frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}}\frac{\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_t - \frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}}\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t) + \frac{\sqrt{1-\beta_t}}{1-\bar{\alpha}_t}\Big(1-\bar{\alpha}_{t-1}\Big)\mathbf{x}_t, \\
&= \frac{1}{\sqrt{1-\beta_t}}\frac{\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{1-\beta_t}}{1-\bar{\alpha}_t}\left(1-\frac{\bar{\alpha}_t}{1-\beta_t}\right)\mathbf{x}_t - \frac{1}{\sqrt{1-\beta_t}}\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t), \\
&= \frac{1}{\sqrt{1-\beta_t}}\left[\frac{\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{(1-\beta_t)}{1-\bar{\alpha}_t}\left(1-\frac{\bar{\alpha}_t}{1-\beta_t}\right)\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)\right], \\
&= \frac{1}{\sqrt{1-\beta_t}}\left[\frac{\beta_t+1-\beta_t-\bar{\alpha}_t}{1-\bar{\alpha}_t}\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)\right], \\
&= \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)\right).
\end{aligned}$$

$$\implies \mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)\right) + \sqrt{\tilde{\beta}_{t-1}}\ \mathbf{z}_{t-1}, \tag{2.4.27}$$

$$= \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t + \beta_t\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t)\right) + \sqrt{\tilde{\beta}_{t-1}}\ \mathbf{z}_{t-1}. \tag{2.4.28}$$

However, an alternative sampler mentioned in Song et al. [2021b] contains β_{t-1} instead of $\tilde{\beta}_{t-1}$:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)\right) + \sqrt{\beta_{t-1}}\mathbf{z}_{t-1}, \tag{2.4.29}$$

$$= \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t + \beta_t\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t)\right) + \sqrt{\beta_{t-1}}\mathbf{z}_{t-1}. \tag{2.4.30}$$

Another alternative sampling technique called Denoising Diffusion Implicit Models (DDIM) was introduced by Song et al. [2021a], as discussed below.

2.4.9 Sampling using DDIM

DDIM [Song et al., 2021a] replaces *Step 2* in eq. (2.4.26) with the DDPM forward process eq. (2.4.5):

$$\textit{Step 1:}\ \hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)). \tag{2.4.31}$$

$$\textit{Step 2:}\ \mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t). \tag{2.4.32}$$

This is derived from the following distributions from Song et al. [2021a]:

$$p_L^{\text{DDIM}}(\mathbf{x}_L \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_L \mid \sqrt{\bar{\alpha}_L}\mathbf{x}_0, (1-\bar{\alpha}_L)\mathbf{I}), \tag{2.4.33}$$

$$q_{t-1}^{\text{DDIM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1} \mid \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}, \mathbf{0}\right), \tag{2.4.34}$$

$$\implies p_t^{\text{DDIM}}(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}\right). \tag{2.4.35}$$

Proof by induction: From 2.115 in Bishop and Nasrabadi [2006]:

For a random variable $\mathbf{u}$ distributed as a normal with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Lambda}^{-1}$, and a dependent variable $\mathbf{v}$ conditionally distributed as a normal with mean $\mathbf{A}\mathbf{u} + \mathbf{b}$ and covariance matrix $\mathbf{L}^{-1}$:

$$p(\mathbf{u}) = \mathcal{N}(\mathbf{u} \mid \boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}), \tag{2.4.36}$$

$$p(\mathbf{v} \mid \mathbf{u}) = \mathcal{N}(\mathbf{v} \mid \mathbf{A}\mathbf{u} + \mathbf{b}, \mathbf{L}^{-1}), \tag{2.4.37}$$

the marginal probability of $\mathbf{v}$ is distributed as:

$$\implies p(\mathbf{v}) = \mathcal{N}(\mathbf{v} \mid \mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^T). \tag{2.4.38}$$

In the case of DDPM, considering $p(\mathbf{u}) = p_t^{\text{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0)$, and $p(\mathbf{v} \mid \mathbf{u}) = q_{t-1}^{\text{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_0)$:

$$p_t^{\text{DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \text{ from eq. (2.4.4), and}$$

$$q_{t-1}^{\text{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1} \mid \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}, \mathbf{0}\right) \text{ from eq. (2.4.21),}$$

then the marginal $p(\mathbf{v}) = q_{t-1}^{\text{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_0)$ is computed using eq. (2.4.38) as:

$$\begin{aligned}\implies q_{t-1}^{\text{DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_0) = \mathcal{N}\Bigg(\mathbf{x}_{t-1} \mid \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\frac{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}},& \\ \mathbf{0} + \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}(1-\bar{\alpha}_t)\mathbf{I}\Bigg),& \quad (2.4.39) \\ = \mathcal{N}\left(\mathbf{x}_{t-1} \mid \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\mathbf{I}\right).& \quad (2.4.40)\end{aligned}$$

Hence $\mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)$ in eq. (2.4.32).

2.4.10 Expected Denoised Sample (EDS) for DDPM

From Saremi and Hyvarinen [2019], for isotropic Gaussian noise, we know that the expected denoised sample $\mathbf{x}_0^*(\mathbf{x}_t, \bar{\alpha}_t) \triangleq \mathbb{E}_{\mathbf{x}_0 \sim q_t(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}_0]$ and the optimal score $\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t)$ are related as:

$$\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t) = \mathbb{E}\left[\|\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)\|_2^2\right](\mathbf{x}_0^*(\mathbf{x}_t, \bar{\alpha}_t) - \mathbf{x}_t). \tag{2.4.41}$$

For DDPM, using eq. (2.4.12), this simplifies to:

$$\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t) = \frac{1}{1-\bar{\alpha}_t}\left(\mathbf{x}_0^*(\mathbf{x}_t, \bar{\alpha}_t) - \mathbf{x}_t\right), \tag{2.4.42}$$

$$\implies \mathbf{x}_0^*(\mathbf{x}_t, \bar{\alpha}_t) = \mathbf{x}_t + (1-\bar{\alpha}_t)\ \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t) = \mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\ \boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t). \tag{2.4.43}$$

While so far the noising/denoising processes have been considered discrete, the following section takes up the continuous formulation of DDPM using Stochastic Differential Equations (SDEs).

2.4.11 SDE formulation : Variance Preserving (VP) SDE

In the continuous formulation, the discrete β_t is now a predefined continuous $\beta(t)$. For DDPM i.e. Variance Preserving (VP) SDE, given $\mathbf{w}$ is a Weiner process i.e. standard Brownian motion, the **forward equation** and **transition probability** are (derived below):

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\ \mathrm{d}t + \sqrt{\beta(t)}\ \mathrm{d}\mathbf{w}, \tag{2.4.44}$$

$$p_{0t}^{\mathrm{VP}}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}\left(\mathbf{x}(t) \mid \mathbf{x}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, \mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s}\right). \tag{2.4.45}$$

2.4.11.1 Derivations :

Forward process: We know from eq. (2.4.2) that:

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_{t-1}. \tag{2.4.46}$$

$$\implies \mathbf{x}(t+\Delta t) = \sqrt{1-\beta(t+\Delta t)\Delta t}\ \mathbf{x}(t) + \sqrt{\beta(t+\Delta t)\Delta t}\ \boldsymbol{\epsilon}(t), \tag{2.4.47}$$

$$\approx \left(1 - \frac{1}{2}\beta(t+\Delta t)\Delta t\right)\mathbf{x}(t) + \sqrt{\beta(t+\Delta t)\Delta t}\ \boldsymbol{\epsilon}(t), \tag{2.4.48}$$

$$\approx \mathbf{x}(t) - \frac{1}{2}\beta(t)\Delta t\ \mathbf{x}(t) + \sqrt{\beta(t)\Delta t}\ \boldsymbol{\epsilon}(t), \tag{2.4.49}$$

$$\implies \mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\ \mathrm{d}t + \sqrt{\beta(t)}\ \mathrm{d}\mathbf{w}. \tag{2.4.50}$$

We know from eq. 5.50 and 5.51 in Särkkä and Solin [2019] that, given that a random variable $\mathbf{x}$ follows a stochastic process with drift coefficient $\mathbf{f}(\mathbf{x}, t)$ and diffusion coefficient $\mathbf{G}(\mathbf{x}, t)$:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + \mathbf{G}(\mathbf{x}, t)\mathrm{d}\mathbf{w}, \tag{2.4.51}$$

the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}_{\mathrm{cov}}$ follow the following differential processes:

$$\frac{\mathrm{d}\boldsymbol{\mu}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}(\mathbf{x}, t)], \tag{2.4.52}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_{\mathrm{cov}}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}(\mathbf{x}, t)(\mathbf{x}-\boldsymbol{\mu})^T] + \mathbb{E}_{\mathbf{x}}[(\mathbf{x}-\boldsymbol{\mu})\mathbf{f}(\mathbf{x}, t)^T] + \mathbb{E}_{\mathbf{x}}[\mathbf{G}(\mathbf{x}, t)\mathbf{Q}\mathbf{G}^T(\mathbf{x}, t)], \tag{2.4.53}$$

where $\mathbf{w}$ is Brownian motion, $\mathbf{Q}$ is the PSD of $\mathbf{w}$. For Gaussian noise, $\mathbf{Q} = \mathbf{I}$.

Hence, **Mean** (from eq. 5.50 in Särkkä and Solin [2019] i.e. eq. (2.4.52) above):

$$\mathrm{d}\mathbf{x} = \mathbf{f}\ \mathrm{d}t + \mathbf{G}\ \mathrm{d}\mathbf{w} \implies \frac{\mathrm{d}\boldsymbol{\mu}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}]. \tag{2.4.54}$$

For DDPM, $\mathbf{f} = -\frac{1}{2}\beta(t)\mathbf{x}$.

$$\therefore \frac{\mathrm{d}\boldsymbol{\mu}_{\text{DDPM}}(t)}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[-\frac{1}{2}\beta(t)\mathbf{x}] = -\frac{1}{2}\beta(t)\mathbb{E}_{\mathbf{x}}(\mathbf{x}) = -\frac{1}{2}\beta(t)\boldsymbol{\mu}_{\text{DDPM}}(t), \tag{2.4.55}$$

$$\implies \frac{\mathrm{d}\boldsymbol{\mu}_{\text{DDPM}}(t)}{\boldsymbol{\mu}_{\text{DDPM}}(t)} = -\frac{1}{2}\beta(t)\mathrm{d}t \implies \log \boldsymbol{\mu}_{\text{DDPM}}(t)|_0^t = -\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s, \tag{2.4.56}$$

$$\implies \log \frac{\boldsymbol{\mu}_{\text{DDPM}}(t)}{\boldsymbol{\mu}(0)} = -\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s, \tag{2.4.57}$$

$$\implies \boldsymbol{\mu}_{\text{DDPM}}(t) = \boldsymbol{\mu}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}. \tag{2.4.58}$$

Covariance (from eq. 5.51 in Särkkä and Solin [2019] i.e. eq. (2.4.53) above):

$$\mathrm{d}\mathbf{x} = \mathbf{f}\ \mathrm{d}t + \mathbf{G}\ \mathrm{d}\mathbf{w},$$

$$\implies \frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{cov}}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}(\mathbf{x}-\boldsymbol{\mu})^T] + \mathbb{E}_{\mathbf{x}}[(\mathbf{x}-\boldsymbol{\mu})\mathbf{f}^T] + \mathbb{E}_{\mathbf{x}}[\mathbf{G}\mathbf{G}^T]. \tag{2.4.59}$$

For DDPM, $\mathbf{f} = -\frac{1}{2}\beta(t)\mathbf{x}, \boldsymbol{\mu} = \mathbf{0}, \mathbf{G} = \sqrt{\beta(t)}\mathbf{I}$.

$$\therefore \frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{DDPM}}(t)}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[-\frac{1}{2}\beta(t)\mathbf{x}\mathbf{x}^T] + \mathbb{E}_{\mathbf{x}}[\mathbf{x}(-\frac{1}{2}\beta(t)\mathbf{x})^T] + \mathbb{E}_{\mathbf{x}}[\sqrt{\beta(t)}\mathbf{I}\sqrt{\beta(t)}\mathbf{I}], \tag{2.4.60}$$

$$= -\beta(t)\boldsymbol{\Sigma}_{\text{DDPM}}(t) + \beta(t)\mathbf{I} = \beta(t)(\mathbf{I} - \boldsymbol{\Sigma}_{\text{DDPM}}(t)), \tag{2.4.61}$$

$$\implies \frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{DDPM}}(t)}{\mathbf{I} - \boldsymbol{\Sigma}_{\text{DDPM}}(t)} = \beta(t)\mathrm{d}t \implies -\log(\mathbf{I} - \boldsymbol{\Sigma}_{\text{DDPM}}(t))|_0^t = \int_0^t \beta(s)\mathrm{d}s, \tag{2.4.62}$$

$$\implies -\log(\mathbf{I} - \boldsymbol{\Sigma}_{\text{DDPM}}(t)) + \log(\mathbf{I} - \boldsymbol{\Sigma}_{\mathbf{x}}(0)) = \int_0^t \beta(s)\mathrm{d}s, \tag{2.4.63}$$

$$\implies \frac{\mathbf{I} - \boldsymbol{\Sigma}_{\text{DDPM}}(t)}{\mathbf{I} - \boldsymbol{\Sigma}_{\mathbf{x}}(0)} = e^{-\int_0^t \beta(s)\mathrm{d}s} \implies \boldsymbol{\Sigma}_{\text{DDPM}}(t) = \mathbf{I} - e^{-\int_0^t \beta(s)\mathrm{d}s}(\mathbf{I} - \boldsymbol{\Sigma}_{\mathbf{x}}(0)), \tag{2.4.64}$$

$$\implies \boldsymbol{\Sigma}_{\text{DDPM}}(t) = \mathbf{I} + e^{-\int_0^t \beta(s)\mathrm{d}s}(\boldsymbol{\Sigma}_{\mathbf{x}}(0) - \mathbf{I}). \tag{2.4.65}$$

For each data point $\mathbf{x}(0)$, $\boldsymbol{\mu}(0) = \mathbf{x}(0)$, $\boldsymbol{\Sigma}_{\mathbf{x}}(0) = \mathbf{0}$:

$$\implies \boldsymbol{\mu}_{\text{DDPM}}(t) = \mathbf{x}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, \tag{2.4.66}$$

$$\boldsymbol{\Sigma}_{\text{DDPM}}(t) = \mathbf{I} + e^{-\int_0^t \beta(s)\mathrm{d}s}(\mathbf{0} - \mathbf{I}) = \mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s}. \tag{2.4.67}$$

$\therefore$ DDPM i.e. $p_{0t}^{\text{VP}}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}\left(\mathbf{x}(t) \mid \mathbf{x}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, \mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s}\right)$ in eq. (2.4.45).

Calculating $\int_0^t \beta(s)\mathrm{d}s$ using a linear beta schedule:

$$\beta(t) = \beta_{\min} + t(\beta_{\max} - \beta_{\min}) \implies \int_0^t \beta(s)\mathrm{d}s = t\beta_{\min} + \frac{t^2}{2}(\beta_{\max} - \beta_{\min}). \tag{2.4.68}$$

2.5 Score Matching Langevin Dynamics (SMLD)

The formulations related to SMLD [Song and Ermon, 2019, 2020] are introduced here. Chapter 6 builds on this and derives a formulation for non-isotropic SMLD.

While DDPM and SMLD both provide excellent generative sample quality, there are key differences in their approaches. In the following sections, the same mathematical derivations as for DDPM are repeated for SMLD so that the differences are made clear. DDPM focuses on modeling the diffusion process of noisy samples to approximate the clean data distribution. In contrast, SMLD directly estimates the gradient from noisy to clean samples i.e. the score function, and employs Langevin dynamics to traverse from noise to data. While DDPM defines a variance preserving SDE, SMLD defines a variance exploding SDE, as will be detailed below.

In their practical implementations, DDPM and SMLD employ different approaches to achieve their objectives. DDPM uses a neural network to estimate the noise to be subtracted from the noisy data to make it cleaner. SMLD uses a neural network to estimate the score i.e. the gradient from noisy to cleaner data. As shall be seen, the score and noise are inter-related, so training to predict one is equivalent to predicting the other.

2.5.1 Forward (data to noise) for SMLD

In SMLD, for a fixed sequence of positive scales $0 < \sigma_1 < \cdots < \sigma_L < 1$, and a noise sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and a clean data point $\mathbf{x}_0$, the **cumulative** "**forward**" process is:

$$q_{\sigma_t}^{\text{SMLD}}(\mathbf{x}_i \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_i \mid \mathbf{x}, \sigma_i^2 \mathbf{I}) \implies \mathbf{x}_i = \mathbf{x}_0 + \sigma_i \boldsymbol{\epsilon}. \tag{2.5.1}$$

The **transition** "**forward**" process can be derived as:

$$q_{\sigma_i}^{\text{SMLD}}(\mathbf{x}_{i+1} \mid \mathbf{x}_i) = \mathcal{N}(\mathbf{x}_{i+1} \mid \mathbf{x}_i, (\sigma_{i+1}^2 - \sigma_i^2)\mathbf{I}) \implies \mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2} \boldsymbol{\epsilon}_{i-1}. \tag{2.5.2}$$

2.5.2 Score for SMLD

For isotropic Gaussian noise as in SMLD,

$$q_{\sigma_t}^{\text{SMLD}}(\mathbf{x}_i \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_i \mid \mathbf{x}_0, \sigma_i^2 \mathbf{I}), \tag{2.5.3}$$

$$\implies \nabla_{\mathbf{x}_i} \log q_{\sigma_i}^{\text{SMLD}}(\mathbf{x}_i \mid \mathbf{x}_0) = -\frac{1}{\sigma_i^2}(\mathbf{x}_i - \mathbf{x}_0) = -\frac{1}{\sigma_i} \boldsymbol{\epsilon}. \tag{2.5.4}$$

2.5.3 Score-matching objective function for SMLD

The objective function for SMLD at noise level σ_i is:

$$\ell^{\text{SMLD}}(\boldsymbol{\theta}; \sigma_i) \triangleq \frac{1}{2} \mathbb{E}_{q_{\sigma_i}^{\text{SMLD}}(\mathbf{x}_i|\mathbf{x}_0) p(\mathbf{x}_0)} \left[\left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) + \frac{1}{\sigma_i^2}(\mathbf{x}_i - \mathbf{x}_0) \right\|_2^2 \right]. \tag{2.5.5}$$

2.5.4 Variance of score for SMLD

$$\mathbb{E}\left[\left\|\nabla_{\mathbf{x}_i} \log q_{\sigma_i}^{\mathrm{SMLD}}(\mathbf{x}_i \mid \mathbf{x}_0)\right\|_2^2\right] = \mathbb{E}\left[\left\|-\frac{(\mathbf{x}_i - \mathbf{x}_0)}{\sigma_i^2}\right\|_2^2\right], = \frac{1}{\sigma_i^2}\mathbb{E}\left[\|\boldsymbol{\epsilon}\|_2^2\right] = \frac{1}{\sigma_i^2}. \tag{2.5.6}$$

2.5.5 Overall objective function for SMLD

Song and Ermon [2019, 2020] chose a geometric series of σ_i's, i.e. $\sigma_{i-1}/\sigma_i = \gamma$. The overall objective function was a weighted combination of the objectives at different noise levels, the weight $\lambda(\sigma_i)$ being the inverse of the variance of the score from eq. (2.5.6) i.e. $\lambda(\sigma_i) = \sigma_i^2$:

$$\begin{aligned}\mathcal{L}^{\mathrm{SMLD}}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^L) &\triangleq \frac{1}{2L}\sum_{i=1}^{L} \mathbb{E}_{q_{\sigma_i}^{\mathrm{SMLD}}(\mathbf{x}_i|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\|\sigma_i \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) + \frac{(\mathbf{x}_i - \mathbf{x}_0)}{\sigma_i}\right\|_2^2\right], \\ &= \frac{1}{2L}\sum_{i=1}^{L} \mathbb{E}_{q_{\sigma_i}^{\mathrm{SMLD}}(\mathbf{x}_i|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\|\sigma_i \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) + \boldsymbol{\epsilon}\|_2^2\right].\end{aligned} \tag{2.5.7}$$

2.5.6 Unconditional SMLD score estimation

Song and Ermon [2020] discovered that empirically the estimated score was proportional to $\frac{1}{\sigma}$. So an unconditional score model is:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) = -\frac{1}{\sigma_i}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_i). \tag{2.5.8}$$

In this case, the overall objective function changes to:

$$\begin{aligned}\mathcal{L}^{\mathrm{SMLD}}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^L) &\triangleq \frac{1}{2L}\sum_{i=1}^{L} \mathbb{E}_{q_{\sigma_i}^{\mathrm{SMLD}}(\mathbf{x}_i|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_i)\|_2^2\right], \\ &= \frac{1}{2L}\sum_{i=1}^{L} \mathbb{E}_{q_{\sigma_i}^{\mathrm{SMLD}}(\mathbf{x}_i|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_0 + \sigma_i\boldsymbol{\epsilon})\|_2^2\right].\end{aligned} \tag{2.5.9}$$

2.5.7 Sampling in SMLD

Unlike DDPM, SMLD does not explicitly define a reverse process. Instead, Song and Ermon [2019, 2020] use an iterative variant of Langevin Sampling called Annealed Langevin Sampling to transform from noise to data. $i = 0$ corresponds to data, and $i = L$ corresponds to noise, hence time order for noise to data is L to 0.

Forward : $\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\boldsymbol{\epsilon}_{i-1}$.

Reverse: Using Annealed Langevin Sampling from Song and Ermon [2019, 2020]:

$\mathbf{x}_L^M \sim \mathcal{N}(\mathbf{0}, \sigma_{\max}\mathbf{I})$.

$$\left.\begin{aligned}
&\mathbf{x}_i^M = \mathbf{x}_{i+1}^0. \\
&\alpha_i = \epsilon\sigma_i^2/\sigma_{\min}^2. \\
&\mathbf{x}_i^{m-1} \leftarrow \mathbf{x}_i^m + \alpha_i \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i^m, \sigma_i) + \sqrt{2\alpha_i}\boldsymbol{\epsilon}_i^{m-1}, m = M, \cdots, 0. \\
\implies\quad &\mathbf{x}_i^{m-1} \leftarrow \mathbf{x}_i^m - \tfrac{\alpha_i}{\sigma_i}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_i^m) + \sqrt{2\alpha_i}\boldsymbol{\epsilon}_i^{m-1}, m = M, \cdots, 0.
\end{aligned}\right\} i = L, \cdots, 1 \tag{2.5.10}$$

Using Consistent Annealed Sampling from Jolicoeur-Martineau et al. [2021b]:

$$\begin{aligned}
&\alpha_i = \epsilon\sigma_i^2/\sigma_{\min}^2 = \eta\sigma_i^2;\ \beta = \sqrt{1-\gamma^2(1-\epsilon/\sigma_{\min}^2)^2};\ \gamma = \sigma_i/\sigma_{i-1}; \sigma_i > \sigma_{i-1}. \\
&\mathbf{x}_{i-1} \leftarrow \mathbf{x}_i + \alpha_i \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma_i) + \beta\sigma_{i-1}\boldsymbol{\epsilon}_{i-1}, i = L, \cdots, 1. \\
\implies &\mathbf{x}_{i-1} \leftarrow \mathbf{x}_i - \eta\sigma_i\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_i) + \beta\sigma_{i-1}\boldsymbol{\epsilon}_{i-1}, i = L, \cdots, 1.
\end{aligned} \tag{2.5.11}$$

2.5.8 Expected Denoised Sample (EDS) for SMLD

From Saremi and Hyvarinen [2019], for isotropic Gaussian noise, we know that the expected denoised sample $\mathbf{x}_0^*(\mathbf{x}_i, \sigma_i) \triangleq \mathbb{E}_{\mathbf{x}_0 \sim q_{\sigma_i}(\mathbf{x}_0|\mathbf{x}_i)}[\mathbf{x}_0]$ and the optimal score $\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma_i)$ are related as:

$$\begin{aligned}
\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma_i) &= \frac{1}{\sigma_i^2}(\mathbf{x}_0^*(\mathbf{x}_i, \sigma_i) - \mathbf{x}_i), \\
\implies \mathbf{x}_0^*(\mathbf{x}_i, \sigma_i) &= \mathbf{x}_i + \sigma_i^2 \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma_i) = \mathbf{x}_i - \sigma_i\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_i).
\end{aligned} \tag{2.5.12}$$

2.5.9 SDE formulation : Variance Exploding (VE) SDE

For SMLD i.e. Variance Exploding (VE) SDE, the forward equation and transition probability are derived (below) as:

$$\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\,\mathrm{d}\mathbf{w}, \tag{2.5.13}$$

$$p_{0t}^{\mathrm{VE}}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}\left(\mathbf{x}(t) \mid \mathbf{x}(0), \sigma^2(t)\mathbf{I}\right). \tag{2.5.14}$$

2.5.9.1 Derivations :

Forward process: We know from eq. (2.5.2) that:

$$\begin{aligned}
\mathbf{x}_i &= \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\boldsymbol{\epsilon}_{i-1}. \\
\implies \mathbf{x}(t+\Delta t) &= \mathbf{x}(t) + \sqrt{(\sigma^2(t+\Delta t) - \sigma^2(t))\Delta t}\ \boldsymbol{\epsilon}(t), \\
&\approx \mathbf{x}(t) + \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}\Delta t}\ \mathbf{w}(t). \\
\implies \mathrm{d}\mathbf{x} &= \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\,\mathrm{d}\mathbf{w}.
\end{aligned} \tag{2.5.15}$$

Thus, eq. (2.5.13) is derived.

Mean $\boldsymbol{\mu}$ and **Covariance $\boldsymbol{\Sigma}_{\text{cov}}$** (from eq. 5.50 and eq.5.51 in Särkkä and Solin [2019]) for a random variable $\mathbf{x}$ that changes according to a stochastic process with drift and diffusion coefficients $\mathbf{f}$ and $\mathbf{G}$, change as:

$$\mathrm{d}\mathbf{x} = \mathbf{f}\ \mathrm{d}t + \mathbf{G}\ \mathrm{d}\mathbf{w} \implies \frac{\mathrm{d}\boldsymbol{\mu}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}], \tag{2.5.16}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{cov}}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}(\mathbf{x}-\boldsymbol{\mu})^T] + \mathbb{E}_{\mathbf{x}}[(\mathbf{x}-\boldsymbol{\mu})\mathbf{f}^T] + \mathbb{E}_{\mathbf{x}}[\mathbf{G}\mathbf{G}^T]. \tag{2.5.17}$$

For SMLD i.e. VE SDE, $\mathbf{f} = \mathbf{0}, \boldsymbol{\mu} = \mathbf{0}, \mathbf{G} = \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\mathbf{I}$.

$$\begin{aligned}
&\frac{\mathrm{d}\boldsymbol{\mu}_{\text{SMLD}}(t)}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{0}] = \mathbf{0},\\
\implies &\boldsymbol{\mu}_{\text{SMLD}}(t) = \boldsymbol{\mu}(0) = \mathbf{x}(0).\\
&\frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{SMLD}}(t)}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}\left[\mathbf{0} + \mathbf{0} + \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\sqrt{\frac{\mathrm{d}[\sigma^2(t)]\mathbf{I}}{\mathrm{d}t}}\right] = \frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}\mathbf{I},\\
\implies &\boldsymbol{\Sigma}_{\text{SMLD}}(t) = \sigma^2(t)\mathbf{I}.
\end{aligned}$$

$\therefore$ SMLD i.e. $p_{0t}^{\text{VE}}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}\left(\mathbf{x}(t) \mid \mathbf{x}(0), \sigma^2(t)\mathbf{I}\right)$. Thus, eq. (2.5.14) is derived.

Chapter 3

Simple video generation using Neural ODEs [Voleti et al., 2019]

3.0 Prologue to article

3.0.1 Article details

Simple video generation using Neural ODEs. Vikram Voleti*, David Kanaa*, Samira Ebrahimi Kahou, Christopher Pal (*denotes equal contribution). *Advances in Neural Information Processing Systems (NeurIPS) 2019 Workshop*

Personal contribution: The project began with discussions between the authors at Mila during Christopher Pal's research group meetings. The idea was to try to see if Neural ODEs were capable of modeling the dynamics of a video, to such an extent as to predict future frames. Vikram Voleti proposed the preliminary experiments to test the hypothesis, and wrote the code. Vikram Voleti and David Kanaa wrote further code, performed several experiments with various settings, discussed the mathematical foundations of the method, proposed further ideas about the architecture of the model, coded two variants in the architecture, performed experiments using the Moving MNIST dataset, and wrote parts of the paper. Samira Ebrahimi Kahou and Christopher Pal provided advice and guidance throughout the project and wrote parts of the paper.

3.0.2 Context

Despite having been studied to a great extent, the task of conditional generation of sequences of frames—or videos—remains extremely challenging. It is a common belief that a key step towards solving this task resides in modelling accurately both spatial and temporal information in video signals. A promising direction to do so has been to learn latent variable models that predict the future in latent space and project back to pixels, as suggested in literature. Recently, Neural ODEs were proposed as a way to model continuous dynamics using neural networks. However, whether they were capable of modeling the dynamics in videos had not been explored yet.

3.0.3 Contributions

Building on top of a family of models introduced in prior works, Neural ODE, this work investigates an approach that models time-continuous dynamics over a continuous latent space with a differential equation with respect to time. The intuition behind this approach is that these trajectories in latent space could then be extrapolated to generate video frames beyond the time steps for which the model is trained. We show that our approach yields promising results in the task of future frame prediction on the Moving MNIST dataset with 1 and 2 digits. To the best of our knowledge, this is the first work that explores the use of Neural ODEs in the space of video generation.

3.0.4 Research impact

Our main hypothesis that Neural ODEs are capable of capturing the dynamics of a video has been re-validated and improved upon by future works. This work influenced several future publications that used Neural ODEs as part of generative models for images [Finlay et al., 2020, Ghosh et al., 2020, Wang et al., 2021, Voleti et al., 2021], as well as improved video generation [Çagatay Yildiz et al., 2019, Park et al., 2021, Xu et al., 2023, Auzina et al., 2023]. Some of these works expanded upon topics we introduced in this paper, such as interpolation of video by oversampling the ODE trajectory.

3.1 Introduction

Conditional frame generation in videos (interpolation and/or extrapolation) remains a challenging task despite having been well studied in the literature. It involves encoding the first few frames of a video into a good representation that could be used for subsequent tasks, i.e. prediction of the next few frames. Solving the task of conditional frame generation in videos requires one to identify, extract and understand latent concepts in images, as well as adequately model both spatial and temporal factors of variation. Typically, an encoder-decoder architecture is used to first encode conditioning frames into latent space, and then recurrently predict future latent points and decode them into pixel space to render future frames.

In this paper, we investigate the use of Neural Ordinary Differential Equations [Chen et al., 2018a] (Neural ODEs) for video generation. The intuition behind this is that we would like to enforce the latent representations of video frames to follow continuous dynamics. Following a dynamic means that frames close to each other in the space-time domain (for example, any video of a natural scene) are close in the latent space. This implies that if we connect the latent embeddings of contiguous video frames, we should be able to obtain trajectories that can be solved for with the help of ordinary differential equations.

Since these trajectories follow certain dynamics in latent space, it should also be possible to extrapolate these trajectories in latent space to future time steps, and decode those latent points to predict future frames. In this paper, we explore this possibility by experimenting on a simple video dataset — Moving MNIST [Srivastava et al., 2015] — and show that Neural ODEs do offer the advantages described above in predicting future video frames.

Our main contributions are:

- we repurpose the encoder-decoder architecture for video generation with Neural ODEs,
- we show promising results on 1-digit and 2-digit Moving MNIST,
- we discuss the future directions of work and relevant challenges.

To the best of our knowledge, this is the first work that explores the use of Neural ODEs in the space of video generation.

3.2 Related work

Early work on using deep learning algorithms to perform video generation has tackled the problematic in various ways, from using stacked regular LSTM layers [Srivastava et al., 2015] to combining convolution with LSTM modules in order to extract local spatial information which correlates with long-term temporal dependencies [Xingjian et al., 2015]. Prabhat et al. [2017] show 3D convolution can be effectively used to extract spatio-temporal information from sequences of images for extreme weather detection. Wang et al. [2018] use a generative model guided with segmentation maps to generate single step future frame. While their results may be interesting, the model might rely too much on segmentation due to the conditioning.

Other work in the recent literature [Babaeizadeh et al., 2018, Denton and Fergus, 2018, Lee et al., 2018a] incorporate stochastic components to their model that encodes the conditioning frames into latent space similar to a prior distribution, which is then sampled from to predict the next frames. This ensures the uncertainty over possible futures is taken into account.

More recently, [Castrejón et al., 2019] show that using a hierarchy of latent variables to improve the expressiveness of their generative model can lead to noticeably better performance on the task of video generation. [Clark et al., 2019] use a Generative Adversarial Network combined with separable spatial and temporal attention models applied on latent feature maps in order to handle spatial and temporal consistency separately.

While some of the above methods have yielded state-of-the-art results, some still struggle to produce smooth motions and for those who do produce continuously smooth ones, they enforce it through temporal regularisation in the optimisation objective, or through a specific training procedure. Drawing from recent work on using parameterised ODE estimators [Chen et al., 2018a] to model continuous-time dynamics, we choose to approach this problem with the intuition that we would like the video frames to be smoothly connected in latent space according to some continuous dynamics which we would learn. Unlike recurrent neural networks and other purely auto-regressive approaches which require observations to occur at uniform intervals, and may require three different models to extrapolate forward and backwards, or interpolate, continuously-defined dynamics should naturally allow to process observations occurring at non-uniform intervals and generate at any time, thus reducing the number of models required to perform extrapolation and interpolation to one.

3.3 Neural Ordinary Differential Equations (Neural ODEs)

Neural Ordinary Differential Equations [Chen et al., 2018a] (Neural ODEs) represent a family of parameterised algorithms designed to model the evolution across time of any system, of state $\boldsymbol{\xi}(t)$ at an arbitrary time t, governed by continuous-time dynamics satisfying a Cauchy (or initial value) problem

$$\begin{cases} \boldsymbol{\xi}(t_0) & = \boldsymbol{\xi}_0, \\ \\ \dfrac{\partial \boldsymbol{\xi}}{\partial t}(t) & = f(\boldsymbol{\xi}(t), t). \end{cases}$$

By approximating the differential with an estimator $f_\theta \simeq f$ parameterized by θ, such as a neural network, these methods allow to learn such dynamics (or, trajectories) from relevant data. Thus formalised, the state $\xi(t)$ of such a system is defined at all times, and can be computed at any desired time using a numerical ODE solver, which will evaluate the dynamics f_θ to determine the solution.

$$(\boldsymbol{\xi}_0, \boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_n) = \texttt{ODEsolver}(f_\theta, \boldsymbol{\xi}_0, (t_0, t_1, \ldots, t_n)).$$

For any single arbitrary time value t_i, a call to the `ODEsolver` computes a numerical approximation of the integral of the dynamics from the initial time value t_0 to t_i.

$$\boldsymbol{\xi}_i = \texttt{ODEsolver}(f_\theta, \boldsymbol{\xi}_0, (t_0, t_i)) \simeq \boldsymbol{\xi}_0 + \int_{t_0}^{t_i} f_\theta\left(\boldsymbol{\xi}(s), s\right)\, ds \,=\, \boldsymbol{\xi}(t_i).$$

There exist in the literature a plethora of algorithms to perform numerical integration of differential equations. Amongst the most common are : the simplest, Euler's method; higher order methods such as Runge-Kutta methods [Pontyagin et al., 1962, Kutta, 1901, Hairer et al., 2000]; as well as multistep algorithms like the Adams-Moulton-Bashforth methods [Butcher, 2016, Hairer et al., 2000, Quarteroni et al., 2000]. More advanced algorithms have been proposed to provide better control over the approximation error and the accuracy [Press et al., 2007]. In their implementation[1] [Chen et al., 2018a] use a variant of the fifth-order Runge-Kutta with adaptive stepsize and local truncation error monitoring to ensure accuracy.

The optimisation of the Neural ODE is performed through the framework of adjoint sensitivity [Pontyagin et al., 1962] which can be formalised as follows. Provided a scalar-valued objective function:

$$L(\theta) = L\left(\boldsymbol{\xi}_0 + \int_{t_0}^{t_i} f_\theta\left(\boldsymbol{\xi}(s), s\right)\, ds\right),$$

the gradient of the objective with respect to the model's parameters follows the differential system:

$$\begin{aligned}\frac{d\boldsymbol{a}(t)}{dt} &= -\,\boldsymbol{a}(t)^\top \frac{\partial f(\boldsymbol{\xi}(t), t, \theta)}{\partial \boldsymbol{\xi}},\\ \frac{dL}{d\theta} &= -\int_{t_i}^{t_0} \boldsymbol{a}(s)^\top \frac{\partial f(\boldsymbol{\xi}(s), s, \theta)}{\partial \theta} ds,\end{aligned}$$

where the $\boldsymbol{a}(t) = \partial L / \partial \boldsymbol{\xi}$ is the adjoint.

3.4 Our approach

Our approach combines the familiar encoder-decoder architecture of neural network models with a Neural ODE that works in the latent space.

(1) We encode the conditioning frames into a point in latent space

(2) We feed this latent embedding to a Neural ODE as the "initial value" at time $t = 0$, and use it to predict latent points corresponding to future time steps.

(3) We decode each of these latent points into frames in pixel space at different time steps

More formally, in accordance with established formulations of the task of video prediction, let us assume a setting in which we have a set of m contextual frames $\mathcal{C} = \{(x_i, t_i)\}_{i \in [\![0,\, m]\!]}$. We seek to learn a predictive model such that, provided $\mathcal{C}$, we can make predictions $\mathcal{P}(\mathcal{C}) =$

[1] https://github.com/rtqichen/torchdiffeq

$\{(x_j, t_j)\}_{j\in[\![m,\, m+n]\!]}$ about the evolution of the video across time, arbitrarily in the future or past (extrapolation) or even in between observed frames (interpolation).

Let $x(t)$ denote the continuous signal representing the video stream from which $\mathcal{C}$ is sampled, that is :

$$\forall (x_i, t_i) \in \mathcal{C},\, x(t_i) = x_i.$$

The temporal changes in the raw signal $x(t)$ can be interpreted as effects of temporal variations in the latent concepts embedded within it. For example, suppose we have a video of a ball moving, any temporal change in the video will be observed only on pixels related to the latent notion of "moving ball". Because the concept "ball" follows some motion, the related pixels will change accordingly. From this statement it follows the intuition to model dynamics in latent space and capture spatial characteristics separately. Thus we learn a predictor $\mathcal{P}$ which

- learns a latent representation of the observed discrete sequence of frames that captures spatial factors of variation, as well as
- infers plausible latent continuous dynamics from which the aforementioned discrete sequence may be sampled i.e. which better explains the temporal variations within the sequence.

The proposed model follows the formalism of latent variable model proposed by [Chen et al., 2018a] in which the latent at the current time value $z(t_m)$ is sampled from a distribution $\mathbb{P}_Z$, the latent generative process is defined by an ODE that determines the trajectory followed in latent space from the initial condition $z(t_m)$, and a conditional $\mathbb{P}_{X|Z}$ with respect the latent vectors predicted along the trajectory at provided times is used to independently sample predicted images:

$$\begin{aligned}
&z_m \sim \mathbb{P}_Z\left(\cdot\right), \\
&z(t_i) = \mathcal{I}(f_\theta, z_m, t_m, t_i) = z_m + \int_{t_m}^{t_i} f\left(z(s), s; \theta\right)\, ds && \forall t_i \in [\![t_m\,,\, t_{m+n}]\!], \\
&x(t_i) \sim \mathbb{P}_{X|Z}\left(\cdot \,|\, z(t_i)\right), \quad x(t_i) \perp\!\!\!\perp x(t_j) && \forall t_i, t_j \in [\![t_m\,,\, t_{m+n}]\!].
\end{aligned}$$

In practice, we use an approximate posterior $q_\phi(\cdot \,|\, \mathcal{C})$ instead of $\mathbb{P}_Z$, and similarly, instead of $\mathbb{P}_{X|Z}$, we use an estimator $p_\psi(\cdot \,|\, z(t_m))$. Together, these estimators function as an *encoder-decoder* pair between the space of image pixels and that of latent representations.

We investigate a deterministic setting where a unique and non-recurrent pair encoder-decoder is used to process every frame. The encoder projects a frame (x_i, t_i) onto an embedding $z_{t_i} = z_i = q_\phi(x_{t_i})$, then the ODE defining the latent dynamics is integrated to produce the value of the latent embedding $z_{t_i} = \mathcal{I}(f_\theta, z_0, t_0, t_i)$. Finally, the decoder is used to project $z(t_j)$ back into an image $\hat{x}_{t_j} = p_\psi(z_{t_j})$. In terms of objective function used to optimise the parameters of the model, we use a combination of an L_2 reconstruction in pixel space, and an L_2 distance between the latent points predicted by the NeuralODE and the embeddings of each frame:

$$\mathcal{L}(\phi, \theta, \psi) = \sum_{[\![t_m\,,\,t_{m+n}]\!]} \|x_{t_i} - p_\psi \circ \mathcal{I}(f_\theta, q_\phi(x_{t_0}), t_0, t)\|_2^2 \\ + \|q_\phi(x_{t_i}) - \mathcal{I}(f_\theta, q_\phi(x_{t_0}), t_0, t_i)\|_2^2. \tag{3.4.1}$$

The latter component of the objective function is meant to ensure that we learn a compact latent subspace to which both the learnt dynamics and the encoder project. More precisely, it enforces the latent representation predicted by the Neural ODE to match that estimated for each time step by the encoder.

We also inquire into the sequence-to-sequence architecture [Chen et al., 2018a], where

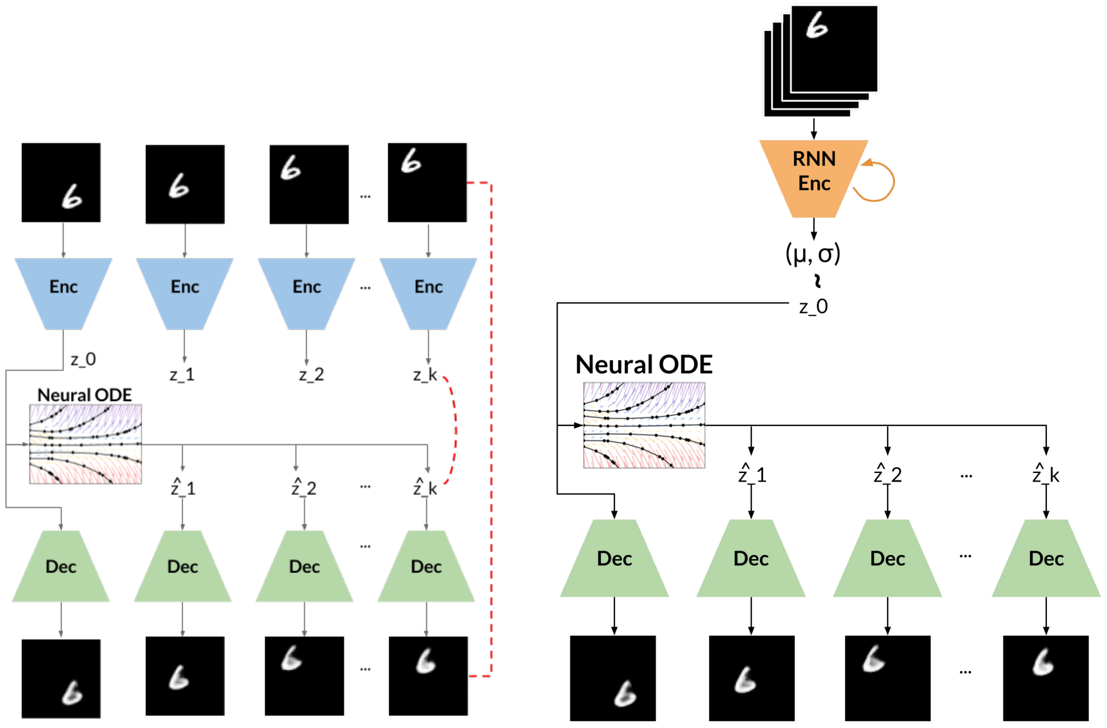
$$\mathbb{P}_Z = \mathcal{N}(\mu(\mathcal{C}), \sigma^2(\mathcal{C})), \quad \mathbb{P}_{X|Z} = \mathcal{N}(\mu(z(t_m)), \sigma^2(z(t_m))),$$
$$\text{thus,}$$
$$q_\phi(\cdot\,|\,\mathcal{C}) = (\mu_\phi(\mathcal{C}),\sigma_\phi^2(\mathcal{C})), \quad p_\psi(\cdot\,|\,z(t_m)) = (\mu_\psi(z(t_m)), \sigma_\psi^2(z(t_m))).$$

In practice, $\sigma_\psi(z(t_m))$ is set to a constant value $\sigma = 1$ and $\mu_\psi(z(t_m)) = x_m$, the true frame observed at time t_m. In this setting, the variational encoder q_ϕ used is based on an RNN model over the context $\mathcal{C} = \{(x_i, t_i)\}_{i \in [\![0,\,m]\!]}$, whereas the decoder p_ψ is non-recurrent—hypothesis of independence between generated frames; the temporal dependencies are modelled by the ODE. At training time, the entire estimator is optimised as a variational auto-encoders [Kingma and Welling, 2013, Rezende et al., 2014] through the maximisation of the Evidence Lower Bound (ELBO):

$$\mathcal{E}(\phi, \theta, \psi) = \underbrace{\sum_{[\![t_m\,,\,t_{m+n}]\!]} - \mathbb{E}_{z_m \sim q_\phi(\cdot\,|\,\mathcal{C})}[\log p_\psi(\hat{x}_t\,|\,\mathcal{I}(f_\theta, z_m, t_m, t)]}_{\text{reconstruction term}} + \mathrm{D}_{KL}(q_\phi(\cdot\,|\,\mathcal{C})\,\|\,\mathcal{N}(0, \boldsymbol{I})). \tag{3.4.2}$$

3.5 Experiments on Moving MNIST

We explore two different methods of combining an encoder-decoder framework with ODEs for 1-digit and 2-digit Moving MNIST [Srivastava et al., 2015]. In each case, we use the first 10 frames as both input to the model and as ground truth for reconstruction, which is the output of the model. We then check how the model performs on the subsequent 10 frames.



(a) (b)

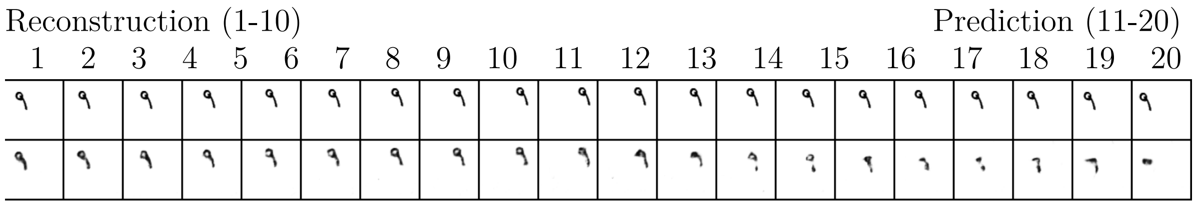
Fig. 3.1. Architectures for Encoder-ODE-Decoder

3.5.1 1-digit Moving MNIST with non-RNN Encoder

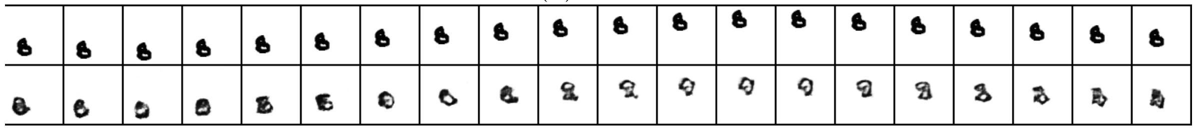
This method, corresponding to Equation 3.4.1, involves an encoder and a decoder that each act on a single frame to embed and decode, respectively, a latent representation. Figure 3.1 (a) shows this architecture. Here, we try to enforce this representation to follow a continuous dynamics in latent space such that there is a one-to-one mapping between the raw pixel space and the latent space from both the encoder side as well as the decoder side.

This model takes one frame as the conditioning input, encodes it, feeds it to the ODE which then predicts the latent representations of the first 10 time steps (including the one which was fed to it), each of which is then decoded to pixel space. We then compute a loss between the reconstructed output and the original input. In addition, each frame of the original video is also encoded separately, and we compute another loss on the encoded latent representations and those predicted by the ODE. This is to enforce the latent representations provided by the encoder to follow the dynamics implicit in the Neural ODE.

We used 1000 video sequences of length 10 as conditioning input (as well as reconstruction output), and a batch size of 100. The encoder and decoder have inverted architectures with the same number of channels in their respective orders. Figure 3.2 shows samples from using this architecture.



(a) Train



(b) Validation

Fig. 3.2. Samples predicted at 20 time steps, conditioned on the first 10 time steps with frames from (a) train set and (b) validation set using Non-RNN Encoder — ODE — Decoder (Figure 3.1a). In each, top row are original samples, and bottom row are predicted samples. For this figure, we use the trained model to reconstruct the first 10 frames and then predict the next 10 frames

3.5.2 1-digit Moving MNIST with RNN Encoder

While the previous architecture works pretty well on the training samples, We see that it does not work very well on validation data. We believe that there are two things that must be corrected:

- The encoder must be conditioned on multiple frames.
- The latent representation provided by the encoder must be stochastic in nature

Since the Neural ODE is only seeing the first frame of the video to base its latent dynamic trajectories on, it is a highly constrained problem. However, we would like to relax this constraint by conditioning the Neural ODE on multiple frames, which is commonly practiced in video prediction/generation.

We would also like to make the model stochastic. The previous model is deterministic, so there is a high chance it simply memorizes the training data. So, given an input frame, there is exactly 1 trajectory the Neural ODE is able to generate for it, so there is no scope of any variation in the generated videos. We would like to generate different videos given the same conditioning input, since it matches with real world data.

Figure 3.1 (b) shows such an architecture that solves both the above issues, corresponding to Equation 3.4.2. It is similar to a Variational Recurrent Neural Network [Chung et al., 2015], except here a Neural ODE handles the latent space.

We feed the first 10 frames as conditioning input to a Recurrent Neural Network (RNN). This network outputs the mean and variance of a multivariate Gaussian distribution. We sample a latent point from this distribution, and feed this to a Neural ODE as the initial value of the latent variable at $t = 0$. The Neural ODE then predicts latent representations at the first 10 time steps, which we then decode independently to raw pixels. We compute a reconstruction loss between the predicted frames and the original frames in the first 10 time steps. We also add a KL-divergence loss between the predicted Gaussian distribution and the standard normal distribution, to constrain the latent representation to follow a standard normal prior.

The model architectures of the encoder (except the recurrent part) and the decoder are the same as in the previous model. We provide 10000 videos as training input, and use a batch size of 128. Figure 3.3 shows the results using this architecture. We can see that the model has been able to capture both structural information and temporal information.

3.5.3 2-digit Moving MNIST with RNN Encoder

We use the same architecture as for 1-digit Moving MNIST (Figure 3.1 (b)) to try to reconstruct 2-digit Moving MNIST. We used the same model settings (number of layers, number of channels, etc.) and the same optimization settings. At the time of writing this paper, we stopped the training at 2000 epochs, same as that for 1-digit Moving MNIST.

Figure 3.4 shows some samples from a model trained on 2-digit Moving MNIST. We believe the spatial trajectories of each individual digit are being recorded very well by the Neural ODE. However it would take many more epochs for the encoder and decoder to reconstruct the images better. This phenomenon of the Neural ODE training earlier than the encoder-decoder was observed while training 1-digit Moving MNIST as well.

3.5.4 A note on the problem formulation

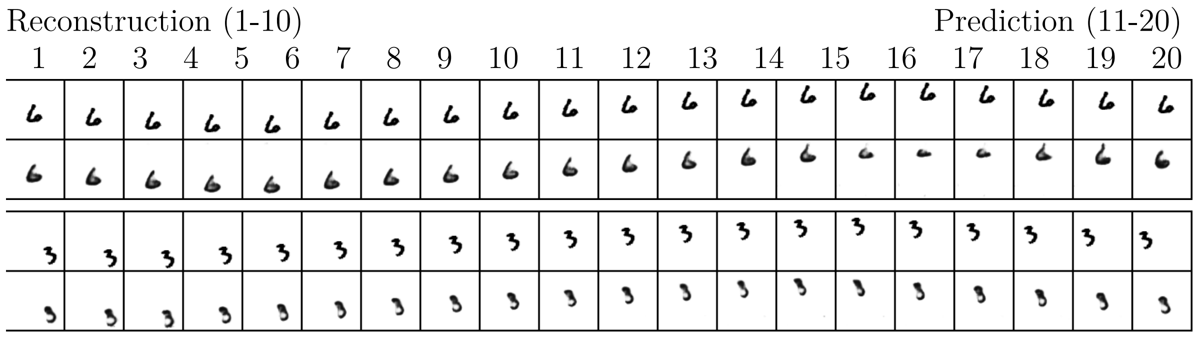
Note the difference in our training formulation compared to the typical approach for generating videos. The typical training procedure involves conditioning the model on initial frames, and training it to predict future steps. Then in evaluation, conditioned on initial frames of unseen videos, the model is used to predict for time steps including and beyond the trained steps.

However, we formulate our training problem as *reconstruction* instead of prediction i.e. conditioned on initial frames, our model is only trained to reconstruct the same frames during training. Despite this, we can still predict future frames effectively by leveraging the preserved dynamics of the training set, by following the trajectory in latent space and decoding.

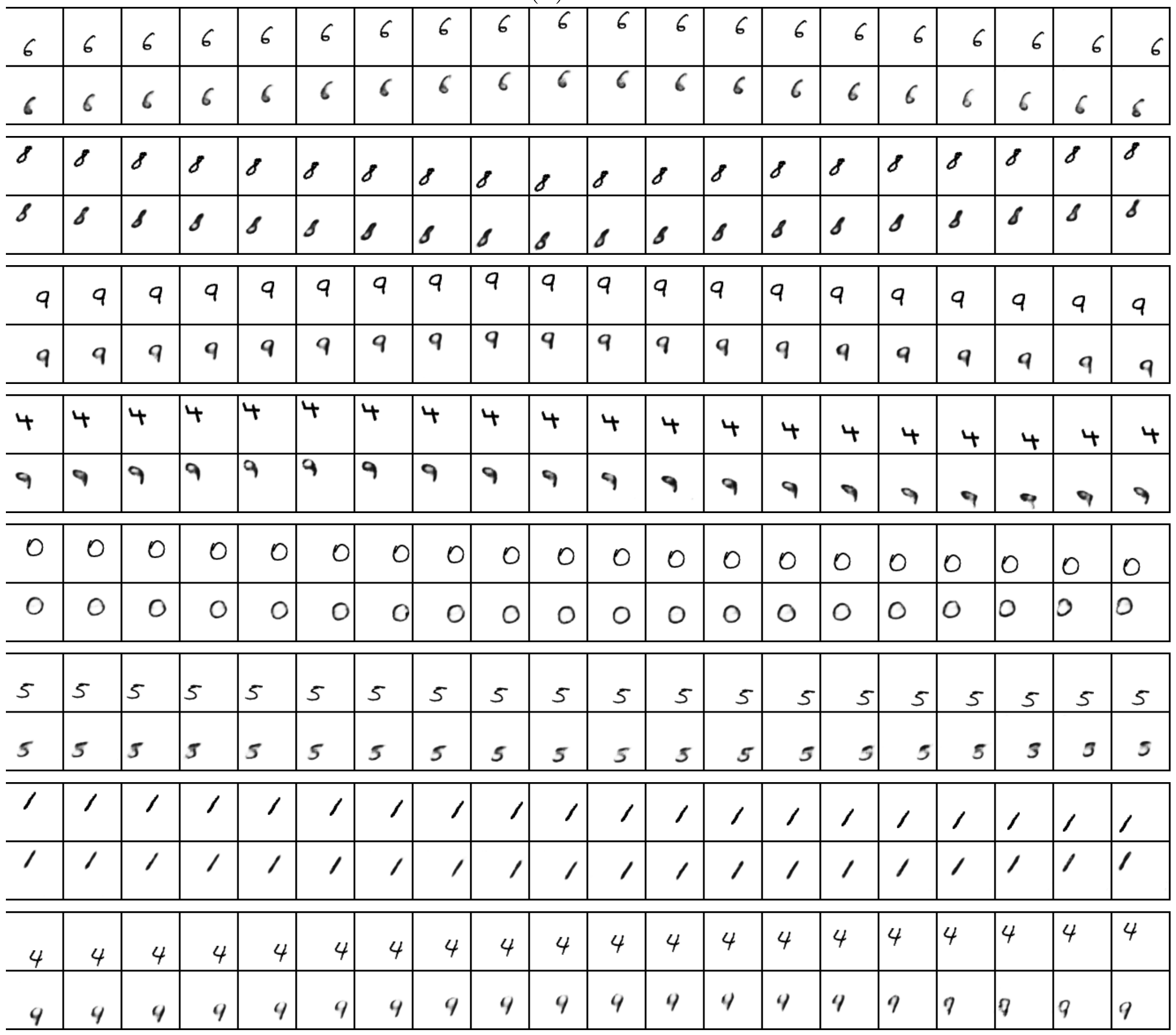
3.6 Future work

There are several future directions we are looking at:

- We would first like to improve the results for 2-digit Moving MNIST. As of the time of writing this paper, we are already making progress in this direction.
- **Scaling up**: We would like to scale this up to bigger datasets such as KTH [Schuldt et al., 2004], the Kinetics dataset [Kay et al., 2017], etc. As of the time of writing this paper, we are already making progress in this direction.

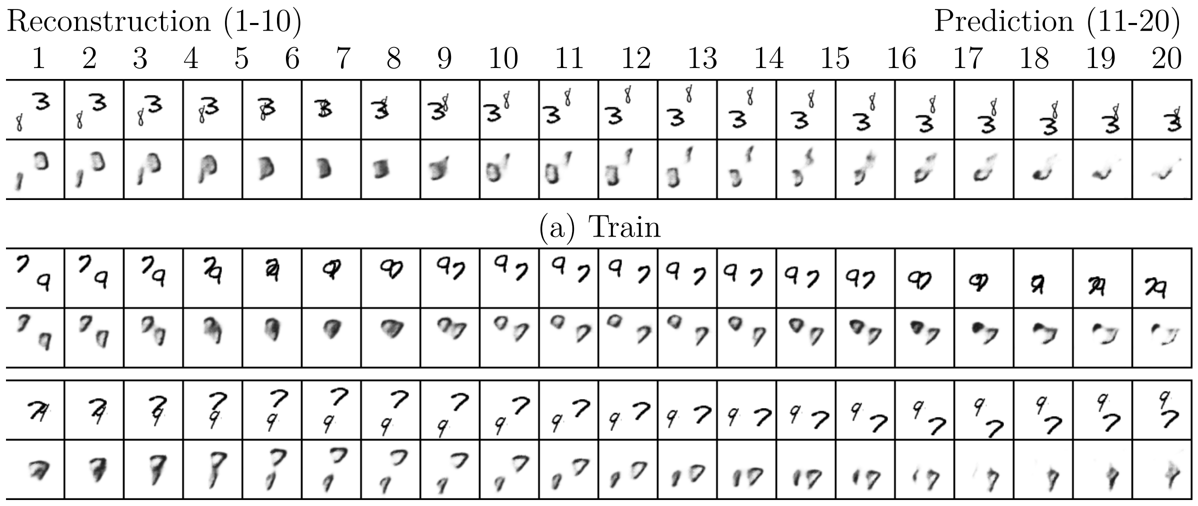


(a) Train



(b) Validation

Fig. 3.3. Predicted samples at 20 time steps conditioned on the first 10 time steps using RNN Encoder-ODE-Decoder (Figure 3.1b). Original (top row) and predicted (bottom row) samples are shown for both (a) train and (b) validation sets. The model is trained to reconstruct the first 10 frames and predict the next 10 frames.



(a) Train

(b) Validation

Fig. 3.4. (top) Original and (bottom) predicted samples of RNN Encoder — ODE — Decoder on 2-digit Moving MNIST

- **Fair comparison**: We would like to explore how well it performs when conditioning on some frames and training to predict the subsequent frames, as it is typically tackled by many of the recent papers on video generation [Denton and Fergus, 2018, Babaeizadeh et al., 2018, Lee et al., 2018a], so we can make a fair comparison of our approach with these methods.
- **Disentanglement**: We would like to examine the latent representation created by the Neural ODE in this domain in more detail. We would like to explore whether it implicitly disentangles spatial and temporal information, which it seems to be doing so from the evidence so far.
- **Visualization**: We would like to visualize the latent representation in lower dimensional space, to check the evidence of trajectories as being enforced by the Neural ODE. The best reason to use Neural ODE is so that the latent representation is now more interpretable i.e. consecutive time steps lie on a lower-dimensional trajectory. We would like to prove this, and show how exploring in latent space maps to intuitive changes in the decoded frames. We plan on using tools such as t-SNE [van der Maaten and Hinton, 2008] and UMAP [McInnes et al., 2018] for visualization.
- **Temporal interpolation of videos**: Since videos follow continuous dynamics in latent space, it is possible to sample latent points for fractional time steps, i.e. time steps that are in the continuous range *between* the time steps of the original video, and decode them using the same decoder. Hence, it should be possible to increase the frame rate of any given video without additional effort. This also opens the door to the use of learned representation for other downstream tasks in video.

- **Better metric for evaluation**: We would also like to have a better metric to estimate the quality of generated videos. The shortcomings of the current metrics of PSNR and SSIM have been mentioned [Lee et al., 2018a]. More recently, [Clark et al., 2019] use the popular image quality metrics of Inception Score [Salimans et al., 2016] and FID [Heusel et al., 2017], however, these metrics do not necessarily account for consistency in temporal information. Since many recent models have a stochastic component, it is all the more important to be able to effectively measure the difference in the real and predict distributions.

3.7 Conclusion

In this paper, we explored the use of Neural ODEs for video prediction. We showed very promising results on the 1-digit and 2-digit Moving MNIST dataset. We investigated two different architectures, with and without a recurrent component, respectively stochastic and deterministic. Even though we formulated the training problem as reconstruction, we were able to use our model for prediction of future frames because the Neural ODE learns the temporal evolution of continuous-time dynamics governing the latent features. We discussed in detail many future directions that would be useful to support our current approach, as well as help the space of video generation. We also discussed how our approach could be directly applied to other tasks such as temporal interpolation of videos. We hope that the research community uses our approach and takes advantage of the implicit feature of Neural ODEs to model continuous dynamics.

Chapter 4

Multi-Resolution Continuous Normalizing Flows [Voleti et al., 2021]

4.0 Prologue to article

4.0.1 Article details

Improving Continuous Normalizing Flows using a Multi-Resolution Framework. Vikram Voleti, Chris Finlay, Adam Oberman, Christopher Pal. *International Conference on Machine Learning 2021 Workshop (also under review at a journal).*

Personal contribution: This project began when Vikram Voleti contacted Chris Finlay, then a PhD candidate at McGill University with Prof. Adam Oberman. Chris Finlay had earlier worked on a publication that improved the dynamics of Neural ODEs for image generation, and Vikram had worked on a project that used Neural ODEs for video generation. Vikram Voleti and Chris Finlay brainstormed over ideas for improving image generation using the continuous normalizing flows framework of Neural ODEs. Adam Oberman and Christopher Pal provided advice and guidance throughout the project and wrote parts of the paper. With help from Adam Oberman and Christopher Pal, Vikram derived the mathematical framework. With help from Chris Finlay, Vikram designed the experiments, wrote the code, ran experiments, proposed and executed on out-of-distribution analysis, and wrote the paper.

4.0.2 Context

Neural Ordinary Differential Equations (Neural ODEs) can serve as generative models of images using the perspective of Continuous Normalizing Flows (CNFs). Such models offer exact likelihood calculation, and invertible generation/density estimation. However, they had not been used in a multi-resolution framework yet, and most implementations using normalizing flows had very high number of parameters as well as high training time. A concurrent work called WaveletFlow also used a multi-resolution framework, however it incurred high parameter cost and training time.

4.0.3 Contributions

In this work we introduce a Multi-Resolution variant of CNFs called Multi-Resolution CNF (MRCNF), by characterizing the conditional distribution over the additional information required to generate a fine image that is consistent with the coarse image. We introduce a transformation between resolutions that allows for no change in the log likelihood. We show that this approach yields comparable likelihood values for various image datasets, with improved performance at higher resolutions, with fewer parameters, using only one GPU. Further, we examine the out-of-distribution properties of MRCNFs, and find that they are similar to those of other likelihood-based generative models.

4.0.4 Research impact

This work derived the use of continuous normalizing flows in the context of image generation in a multi-resolution framework. Although there are concurrent and follow-up works that also show the validity of our approach [Yu et al., 2020], the adoption of our proposed continuous normalizing flows framework however hasn't been as widespread as we had hoped.

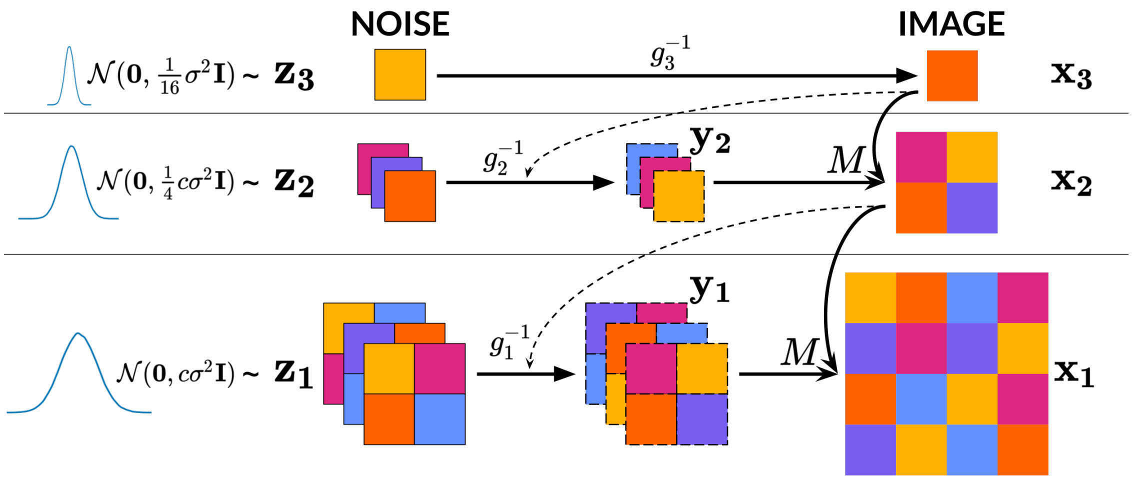

NOISE
IMAGE
$\mathcal{N}(\mathbf{0}, \frac{1}{16}\sigma^2\mathbf{I}) \sim \mathbf{z_3}$
g_3^{-1}
$\mathbf{x_3}$
$\mathcal{N}(\mathbf{0}, \frac{1}{4}c\sigma^2\mathbf{I}) \sim \mathbf{z_2}$
g_2^{-1}
$\mathbf{y_2}$
M
$\mathbf{x_2}$
$\mathcal{N}(\mathbf{0}, c\sigma^2\mathbf{I}) \sim \mathbf{z_1}$
g_1^{-1}
$\mathbf{y_1}$
M
$\mathbf{x_1}$

Fig. 4.1. The architecture of our MRCNF method (best viewed in color). Continuous normalizing flows (CNFs) g_s are used to generate images $\mathbf{x}_s$ from noise $\mathbf{z}_s$ at each resolution, with those at finer resolutions conditioned (dashed lines) on the coarser image one level above $\mathbf{x}_{s+1}$, except at the coarsest level where it is unconditional. Every finer CNF produces an intermediate image $\mathbf{y}_s$, which is then combined with the immediate coarser image $\mathbf{x}_{s+1}$ using a linear map M from Equation 4.3.6 to form $\mathbf{x}_s$. The multiscale maps are defined by Equation 4.3.16.

4.1 Introduction

Reversible generative models derived through the use of the change of variables technique [Dinh et al., 2017, Kingma and Dhariwal, 2018, Ho et al., 2019a, Yu et al., 2020] are growing in interest as alternatives to generative models based on Generative Adversarial Networks (GANs) [Goodfellow et al., 2016] and Variational Autoencoders (VAEs) [Kingma and Welling, 2013]. While GANs and VAEs have been able to produce visually impressive samples of images, they have a number of limitations. A change of variables approach facilitates the transformation of a simple base probability distribution into a more complex model distribution. Reversible generative models using this technique are attractive because they enable efficient density estimation, efficient sampling, and computation of exact likelihoods.

A promising variation of the change-of-variable approach is based on the use of a continuous time variant of normalizing flows [Chen et al., 2018a, Grathwohl et al., 2019, Finlay et al., 2020], which uses an integral over continuous time dynamics to transform a base distribution into the model distribution, called CNF. This approach uses ordinary differential equations (ODEs) specified by a neural network, or Neural ODEs. CNFs have been shown to be capable of modelling complex distributions such as those associated with images.

While this new paradigm for the generative modelling of images is not as mature as GANs or VAEs in terms of the generated image quality, it is a promising direction of research as it does not have some key shortcomings associated with GANs and VAEs. Specifically, GANs are known to suffer from mode-collapse [Lin et al., 2018], and are notoriously difficult to train [Arjovsky and Bottou, 2017] compared to feed forward networks because their adversarial loss seeks a saddle point instead of a local minimum [Berard et al., 2020]. CNFs are trained by mapping images to noise, and their reversible architecture allows images to be generated by going in reverse, from noise to images. This leads to fewer issues related to mode collapse, since any input example in the dataset can be recovered from the flow using the reverse of the transformation learned during training. VAEs only provide a lower bound on the marginal likelihood whereas CNFs provide exact likelihoods. Despite the many advantages of reversible generative models built with CNFs, quantitatively such methods still do not match the widely used Fréchet Inception Distance (FID) scores of GANs or VAEs. However their other advantages motivate us to explore them further.

Furthermore, state-of-the art GANs and VAEs exploit the multi-resolution properties of images, and recent top-performing methods also inject noise at each resolution [Brock et al., 2019, Shaham et al., 2019, Karras et al., 2020, Vahdat and Kautz, 2020]. While shaping noise is fundamental to normalizing flows, only recently have normalizing flows exploited the multi-resolution properties of images. For example, WaveletFlow [Yu et al., 2020] splits an image into multiple resolutions using the Discrete Wavelet Transform, and models the average image at each resolution using a normalizing flow. While this method has advantages, it suffers from many issues such as high parameter count and long training time.

In this work, we consider a non-trivial multi-resolution approach to continuous normalizing flows, which fixes many of these issues. A high-level view of our approach is shown in Figure 4.1. Our main contributions are:

(1) We propose a multi-resolution transformation that does not add cost in terms of likelihood.
(2) We introduce **Multi-Resolution Continuous Normalizing Flows (MRCNF)**.
(3) We achieve comparable Bits-per-dimension (BPD) (negative log likelihood per pixel) on image datasets using fewer model parameters and significantly less training time with only one GPU.
(4) We explore the out-of-distribution properties of (MR)CNF, and find that they are similar to non-continuous normalizing flows.

4.2 Background

4.2.1 Normalizing flows

Normalizing flows [Tabak and Turner, 2013, Jimenez Rezende and Mohamed, 2015, Dinh et al., 2017, Papamakarios et al., 2019, Kobyzev et al., 2020] are generative models that map a complex data distribution, such as real images, to a known noise distribution. They are trained by maximizing the log likelihood of their input images. Suppose a normalizing flow g

produces output $\mathbf{z}$ from an input $\mathbf{x}$ i.e. $\mathbf{z} = g(\mathbf{x})$. The change-of-variables formula provides the likelihood of the image under this transformation as:

$$\log p(\mathbf{x}) = \log \left| \det \frac{\mathrm{d}g}{\mathrm{d}\mathbf{x}} \right| + \log p(\mathbf{z}). \tag{4.2.1}$$

The first term on the right (log determinant of the Jacobian) is often intractable, however, previous works on normalizing flows have found ways to estimate this efficiently. The second term, $\log p(\mathbf{z})$, is computed as the log probability of $\mathbf{z}$ under a known noise distribution, typically the standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The normalizing flow is trained by maximizing the log-likelilhood of the data $\mathbf{x}$ in the real distribution i.e. $\log p(\mathbf{x})$, using Equation 4.2.1.

4.2.2 Continuous Normalizing Flows (CNF)

Continuous Normalizing Flows (CNF) [Chen et al., 2018a, Grathwohl et al., 2019, Finlay et al., 2020] are a variant of normalizing flows that operate in the continuous domain. A CNF creates a geometric flow between the input and target (noise) distributions, by assuming that the state transition is governed by an Ordinary Differential Equation (ODE). It further assumes that the differential function is parameterized by a neural network, this model is called a Neural ODE [Chen et al., 2018a]. Suppose CNF g transforms its state $\mathbf{v}(t)$ using a Neural ODE, with the differential defined by the neural network f parameterized by θ. $\mathbf{v}(t_0) = \mathbf{x}$ is, say, an image, and at the final time step $\mathbf{v}(t_1) = \mathbf{z}$ is a sample from a known noise distribution.

$$\frac{\mathrm{d}\mathbf{v}(t)}{\mathrm{d}t} = f(\mathbf{v}(t), t, \theta) \implies \mathbf{v}(t_1) = g(\mathbf{v}(t_0)) = \mathbf{v}(t_0) + \int_{t_0}^{t_1} f(\mathbf{v}(t), t, \theta)\ \mathrm{d}t. \tag{4.2.2}$$

This integration is typically performed by an ODE solver. Since this integration can be run backwards as well to obtain the same $\mathbf{v}(t_0)$ from $\mathbf{v}(t_1)$, a CNF is a reversible model. Equation 4.2.1 can be used to compute the change in log-probability induced by the CNF. However, Chen et al. [2018a] and Grathwohl et al. [2019] proposed a more efficient variant in the CNF context, the instantaneous change-of-variables formula:

$$\frac{\partial \log p(\mathbf{v}(t))}{\partial t} = -\mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{v}(t)}\right), \tag{4.2.3}$$

$$\implies \Delta \log p_{\mathbf{v}(t_0) \to \mathbf{v}(t_1)} = -\int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f_\theta}{\partial \mathbf{v}(t)}\right) \mathrm{d}t. \tag{4.2.4}$$

Hence, the change in log-probability of the state of the Neural ODE i.e. $\Delta \log p_{\mathbf{v}}$ is expressed as another differential equation. The ODE solver now solves both differential equations Equation 4.2.2 and Equation 4.2.4 by augmenting the original state with the above. Thus, a CNF provides both the final state $\mathbf{v}(t_1)$ as well as the change in log probability $\Delta \log p_{\mathbf{v}(t_0) \to \mathbf{v}(t_1)}$ together.

Prior works [Grathwohl et al., 2019, Finlay et al., 2020, Ghosh et al., 2020, Onken et al., 2021, Huang and Yeh, 2021] have trained CNFs as reversible generative models of images by maximizing the image likelihood:

$$\mathbf{z} = g(\mathbf{x}) \qquad ; \qquad \log p(\mathbf{x}) = \Delta \log p_{\mathbf{x}\to\mathbf{z}} + \log p(\mathbf{z}). \tag{4.2.5}$$

where $\mathbf{x}$ is an image, $\mathbf{z}$ and $\Delta \log p_{\mathbf{x}\to\mathbf{z}}$ are computed by the CNF using Equation 4.2.2 and Equation 4.2.4, and $\log p(\mathbf{z})$ is the likelihood of $\mathbf{z}$ under a known noise distribution, typically the standard Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Novel images are generated by sampling $\mathbf{z}$ from the noise distribution, and running the CNF in reverse.

4.3 Our method

Our method is a reversible generative model of images that builds on top of CNFs. We introduce the notion of multiple resolutions in images, and connect the different resolutions in an autoregressive fashion. This helps generate images faster, with better likelihood values at higher resolutions, using only one GPU in all our experiments. We call this model Multi-Resolution Continuous Normalizing Flow (MRCNF).

4.3.1 Multi-resolution image representation

Multi-resolution representations of images have been explored in computer vision for decades [Burt, 1981, Marr, 2010, Witkin, 1987, Burt and Adelson, 1983, Mallat, 1989, Lindeberg, 1990]. Much of the content of an image at a resolution is a composition of low-level information captured at coarser resolutions, and high-level information not present in the coarser images. We take advantage of this by first decomposing an image in *resolution space* i.e. by expressing it as a series of S images at decreasing resolutions: $\mathbf{x} \to (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_S)$, where $\mathbf{x}_1 = \mathbf{x}$ is the finest image, $\mathbf{x}_S$ is the coarsest, and every $\mathbf{x}_{s+1}$ is the average image of $\mathbf{x}_s$. Thus, each $\mathbf{x}_j, j > i$ is deterministic if $\mathbf{x}_i$ is given. This called an image pyramid, or a Gaussian Pyramid if the upsampling-downsampling operations include a Gaussian filter [Burt, 1981, Burt and Adelson, 1983, Adelson et al., 1984, Witkin, 1987, Lindeberg, 1990]. In this work, we obtain a coarser image simply by averaging pixels in every 2x2 patch, thereby halving the width and height. , i.e. we express a 32×32 image into, say, 3 images of resolutions: (8×8, 16×16, 32×32), each image being an average of its immediate finer image. However, this representation is redundant since much of the information in $\mathbf{x}_1$ is contained in $\mathbf{x}_{s>1}$. Instead, We then express $\mathbf{x}$ as a series of high-level information $\mathbf{y}_s$ not present in the immediate coarser images $\mathbf{x}_{s+1}$, and a final coarse image $\mathbf{x}_S$, and our overall method is to map these S terms to S noise samples using S CNFs.:

$$\mathbf{x} \to (\mathbf{y}_1, \mathbf{x}_2) \to (\mathbf{y}_1, \mathbf{y}_2, \mathbf{x}_3) \to \cdots \to (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{S-1}, \mathbf{x}_S). \tag{4.3.1}$$

4.3.2 Defining the high-level information $\mathbf{y}_s$

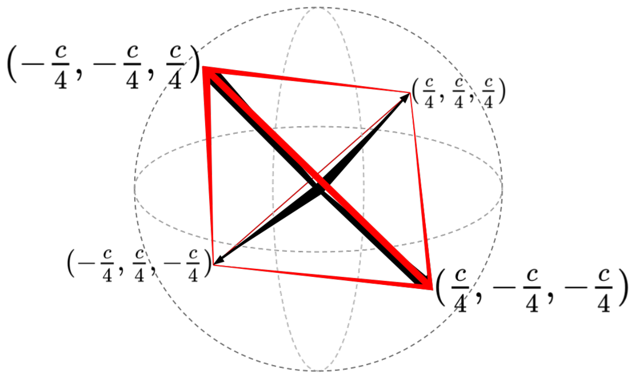

$(-\frac{c}{4}, -\frac{c}{4}, \frac{c}{4})$
$(\frac{c}{4}, \frac{c}{4}, \frac{c}{4})$
$(-\frac{c}{4}, \frac{c}{4}, -\frac{c}{4})$
$(\frac{c}{4}, -\frac{c}{4}, -\frac{c}{4})$

Fig. 4.2. Tetrahedron in 3D space with 4 corners. $c = 2^{2/3}$

We choose to design a linear transformation with the following properties: 1) invertible i.e. it should be possible to deterministically obtain $\mathbf{x}_s$ from $\mathbf{y}_s$ and $\mathbf{x}_{s+1}$, and vice versa ; 2) volume preserving i.e. determinant is 1, change in log-likelihood is 0 ; 3) angle preserving ; and 4) range preserving.

Consider the simplest case of 2 resolutions where $\mathbf{x}_1$ is a 2x2 image with pixel values x_1, x_2, x_3, x_4, and $\mathbf{x}_2$ is a 1x1 image with pixel value $\bar{x} = \frac{1}{4}(x_1 + x_2 + x_3 + x_4)$. We require three values $(y_1, y_2, y_3) = \mathbf{y}_1$ that contain information not present in $\mathbf{x}_2$, such that $\mathbf{x}_1$ is obtained when $\mathbf{y}_1$ and $\mathbf{x}_2$ are combined.

This could be viewed as a problem of finding a matrix $\mathbf{M}$ such that: $[x_1, x_2, x_3, x_4]^\top = \mathbf{M}\ [y_1, y_2, y_3, \bar{x}]^\top$. We fix the last column of $\mathbf{M}$ as $[1, 1, 1, 1]^\top$, since every pixel value in $\mathbf{x}_1$ depends on $\bar{x}$. Finding the rest of the parameters can be viewed as requiring four 3D vectors that are spaced such that they do not degenerate the number of dimensions of their span. These can be considered as the four corners of a tetrahedron in 3D space, under any configuration (rotated in 3D space), and any scaling of the vectors (see Figure 4.2).

Out of the many possibilities for this tetrahedron is the matrix that performs the Discrete Haar Wavelet Transform [Mallat, 1989, Mallat and Peyré, 2009]:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 1 \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & 1 \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \bar{x} \end{bmatrix} \iff \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \bar{x} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}. \tag{4.3.2}$$

However, this transformation incurs a cost in terms of log-likelihood:

$$\Delta \log p_{(x_1,x_2,x_3,x_4)\to(y_1,y_2,y_3,\bar{x})} = \log \left|\det(\mathbf{M}^{-1})\right| = \log(1/2). \tag{4.3.3}$$

and is therefore not volume preserving. Other simple scaling of Equation 4.3.2 has been used in the past, for example multiplying the last row by 2, yielding an orthogonal transformation, such as in WaveletFlow [Yu et al., 2020]. However, this transformation neither preserves the volume i.e. the log determinant is not 0, nor the maximum i.e. the range of $\mathbf{x}_s$ changes.

We wish to find a transformation $\mathbf{M}$ where: one of the results is the average of the inputs, $\bar{x}$; it is unit determinant; the columns are orthogonal; and it preserves the range of $\bar{x}$. Fortunately such a matrix exists – although we have not seen it discussed in prior literature. It can be seen as a variant of the Discrete Haar Wavelet Transformation matrix that is unimodular, i.e. has a determinant of 1 (and is therefore volume preserving). It is also preserving the range of the images for the input and its average, , i.e. it is range preserving such that $\min(x) = \min(\bar{x})$ and $\max(x) = \max(\bar{x})$. It is obtained by multiplying the 3 rows corresponding to the non-average terms with $2^{2/3}$. This is shown in Figure 4.2):

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \frac{1}{a} \begin{bmatrix} c & c & c & a \\ c & -c & -c & a \\ -c & c & -c & a \\ -c & -c & c & a \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \bar{x} \end{bmatrix} \iff \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \bar{x} \end{bmatrix} = \begin{bmatrix} c^{-1} & c^{-1} & -c^{-1} & -c^{-1} \\ c^{-1} & -c^{-1} & c^{-1} & -c^{-1} \\ c^{-1} & -c^{-1} & -c^{-1} & c^{-1} \\ a^{-1} & a^{-1} & a^{-1} & a^{-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \tag{4.3.4}$$

where $c = 2^{2/3}$, $a = 4$. Hence, there is no cost to the log-likelihood due to the transformation:

$$\Delta \log p_{(x_1,x_2,x_3,x_4)\to(y_1,y_2,y_3,\bar{x})} = \log \left|\det(\mathbf{M}^{-1})\right| = \log(1) = 0. \tag{4.3.5}$$

This can be scaled up to larger spatial regions by performing the same calculation for each 2x2 patch. Let M be the function that uses matrix $\mathbf{M}$ from above and combines every pixel in $\mathbf{x}_{s+1}$ with the three corresponding pixels in $\mathbf{y}_s$ to make the 2x2 patch at that location in $\mathbf{x}_s$ using Equation 4.3.4:

$$\mathbf{x}_s = M(\mathbf{y}_s, \mathbf{x}_{s+1}) \iff \mathbf{y}_s, \mathbf{x}_{s+1} = M^{-1}(\mathbf{x}_s). \tag{4.3.6}$$

Equation 4.2.1 can be used to compute the change in log likelihood from this transformation $\mathbf{x}_s \to (\mathbf{y}_s, \mathbf{x}_{s+1})$:

$$\begin{aligned} \log p(\mathbf{x}_s) &= \Delta \log p_{\mathbf{x}_s \to (\mathbf{y}_s, \mathbf{x}_{s+1})} + \log p(\mathbf{y}_s, \mathbf{x}_{s+1}), \\ &= \log \left|\det(M^{-1})\right| + \log p(\mathbf{y}_s, \mathbf{x}_{s+1}), \end{aligned} \tag{4.3.7}$$

where $\log\left|\det(M^{-1})\right|$ can be determined for the Wavelet transform in Equation 4.3.2 using:

$$\log \left|\det(M^{-1})\right| = \mathrm{dims}(\mathbf{x}_{s+1}) \log(1/2), \tag{4.3.8}$$

where "dims" is the number of pixels times the number of channels (typically 3) in the image, and for our unimodular transform in Equation 4.3.4 using Equation 4.3.5 as :

$$\log \left|\det(M^{-1})\right| = 0. \tag{4.3.9}$$

4.3.3 Multi-Resolution Continuous Normalizing Flows (MRCNF)

Using the multi-resolution image representation in Equation 4.3.1, we characterize the conditional distribution over the additional degrees of freedom ($\mathbf{y}_s$) required to generate a higher resolution image ($\mathbf{x}_s$) that is consistent with the average ($\mathbf{x}_{s+1}$) over the equivalent pixel space. At each resolution s, we use a CNF to reversibly map between $\mathbf{y}_s$ (or $\mathbf{x}_S$ when $s{=}S$) and a sample $\mathbf{z}_s$ from a known noise distribution. For generation, $\mathbf{y}_s$ only adds information missing in $\mathbf{x}_{s+1}$, but conditional on it.

This framework ensures a coarse image could generate several potential fine images, but these fine images share the same coarse image as their average. This fact is preserved across resolutions. Note that the 3 additional pixels in $\mathbf{y}_s$ per pixel in $\mathbf{x}_{s+1}$ are generated conditioned on the entire coarser image $\mathbf{x}_{s+1}$, thus maintaining consistency using full context.

In principle, any generative model could be used to map between the multi-resolution image and noise. Normalizing flows are good candidates for this as they are probabilistic generative models that perform exact likelihood estimates, and can be run in reverse to generate novel data from the model's distribution. This allows model comparison and measurement of generalization to unseen data. We choose to use the CNF variant of normalizing flows at each resolution. CNFs have recently been shown to be effective in modeling image distributions using a fraction of the number of parameters typically used in normalizing flows (and non flow-based approaches), and their underlying framework of Neural ODEs have been shown to be more robust than convolutional layers [Yan et al., 2020].

Training: We train an MRCNF by maximizing the average log-likelihood of the images in the training dataset under the model. The log probability of each image $\log p(\mathbf{x})$ can be estimated recursively using the sequence of variables in Equation 4.3.1, and the corresponding simplification of the log-probability using Equation 4.3.7 as (here, $\mathbf{x}_1 = \mathbf{x}$):

$$\begin{aligned}
\log p(\mathbf{x}) &= \Delta \log p_{\mathbf{x}_1 \to (\mathbf{y}_1, \mathbf{x}_2)} + \log p(\mathbf{y}_1, \mathbf{x}_2), \\
&= \Delta \log p_{\mathbf{x}_1 \to (\mathbf{y}_1, \mathbf{x}_2)} + \log p(\mathbf{y}_1 \mid \mathbf{x}_2) + \log p(\mathbf{x}_2), \\
&= \Delta \log p_{\mathbf{x}_1 \to (\mathbf{y}_1, \mathbf{x}_2)} + \log p(\mathbf{y}_1 \mid \mathbf{x}_2) \\
&\qquad + \Delta \log p_{\mathbf{x}_2 \to (\mathbf{y}_2, \mathbf{x}_3)} + \log p(\mathbf{y}_2 \mid \mathbf{x}_3) + \log p(\mathbf{x}_3), \\
&\vdots \\
&= \sum_{s=1}^{S-1} \Big(\Delta \log p_{\mathbf{x}_s \to (\mathbf{y}_s, \mathbf{x}_{s+1})} + \log p(\mathbf{y}_s \mid \mathbf{x}_{s+1}) \Big) + \log p(\mathbf{x}_S). \qquad (4.3.10)
\end{aligned}$$

where $\Delta \log p_{\mathbf{x}_s \to (\mathbf{y}_s, \mathbf{x}_{s+1})}$ is given by Equation 4.3.9, while $\log p(\mathbf{y}_s \mid \mathbf{x}_{s+1})$ and $\log p(\mathbf{x}_S)$ (at the coarsest resolution S) are given by Equation 4.2.5:

$$\mathbf{z}_s = g_s(\mathbf{y}_s \mid \mathbf{x}_{s+1}); \quad \log p(\mathbf{y}_s \mid \mathbf{x}_{s+1}) = \Delta \log p_{(\mathbf{y}_s \to \mathbf{z}_s) \mid \mathbf{x}_{s+1}} + \log p(\mathbf{z}_s), \qquad (4.3.11)$$

$$\mathbf{z}_S = g_S(\mathbf{x}_S); \qquad \log p(\mathbf{x}_S) = \Delta \log p_{\mathbf{x}_S \to \mathbf{z}_S} + \log p(\mathbf{z}_S). \tag{4.3.12}$$

The coarsest resolution S can be chosen such that the last CNF operates on the image distribution at a small enough resolution that is easy to model unconditionally. All other CNFs are conditioned on the immediate coarser image. The conditioning itself is achieved by concatenating the input image of the CNF with the coarser image. This model could be seen as a stack of CNFs connected in an autoregressive fashion.

Typically, likelihood-based generative models are compared using the metric of bits-per-dimension (BPD), i.e. the negative log likelihood per pixel in the image. Hence, we train our MRCNF to minimize the average BPD of the images in the training dataset, computed using Equation 4.3.13:

$$\text{BPD}(\mathbf{x}) = -\log p(\mathbf{x}) / \text{dims}(\mathbf{x}). \tag{4.3.13}$$

We use FFJORD [Grathwohl et al., 2019] as the baseline model for our CNFs. In addition, we use to two regularization terms introduced by RNODE [Finlay et al., 2020] to speed up the training of FFJORD models by stabilizing the learnt dynamics: the kinetic energy $\mathcal{K}(\theta)$ and the Jacobian norm $\mathcal{B}(\theta)$ of the flow $f(\mathbf{v}(t), t, \theta)$ described in subsection 4.2.2:

$$\mathcal{K}(\theta) = \int_{t_0}^{t_1} \| f(\mathbf{v}(t), t, \theta) \|_2^2 \, \mathrm{d}t \quad ; \tag{4.3.14}$$

$$\mathcal{B}(\theta) = \int_{t_0}^{t_1} \| \epsilon^\top \nabla_z f(\mathbf{v}(t), t, \theta) \|_2^2 \, \mathrm{d}t, \quad \epsilon \sim \mathcal{N}(0, I). \tag{4.3.15}$$

Parallel training: Note that although the final log likelihood $\log p(\mathbf{x})$ involves sequentially summing over values returned by all S CNFs, the log likelihood term of each CNF is independent of the others. Conditioning is done using ground truth images. Hence, each CNF can be trained independently, in parallel.

Generation: Given an S-resolution model, we first sample $\mathbf{z}_s, s = 1, \ldots, S$ from the latent noise distributions. The CNF g_s at resolution s transforms the noise sample $\mathbf{z}_s$ to high-level information $\mathbf{y}_s$ conditioned on the immediate coarse image $\mathbf{x}_{s+1}$ (except g_S which is unconditioned). $\mathbf{y}_s$ and $\mathbf{x}_{s+1}$ are then combined to form $\mathbf{x}_s$ using M from Equation 4.3.4. This process is repeated progressively from coarser to finer resolutions, until the finest resolution image $\mathbf{x}_1$ is computed (see Figure 4.1). It is to be noted that the generated image at one resolution is used to condition the CNF at the finer resolution.

$$\begin{cases} \mathbf{x}_S = g_S^{-1}(\mathbf{z}_S) & s = S, \\ \mathbf{y}_s = g_s^{-1}(\mathbf{z}_s \mid \mathbf{x}_{s+1}); \quad \mathbf{x}_s = M(\mathbf{y}_s, \mathbf{x}_{s+1}) & s = S\text{-}1 \to 1. \end{cases} \tag{4.3.16}$$

4.3.4 Multi-resolution noise

We further decompose the noise image as well into its respective coarser components. This means ultimately we use only one noise image at the finest level, and it is decomposed into multiple resolutions using Equation 4.3.4. $\mathbf{x}_{s+1}$ is mapped to noise of 1/4 variance, $\mathbf{y}_s$ is mapped to noise of c-factored variance (see Figure 4.1). Although this is optional, it preserves interpretation between the single- and multi-resolution models.

4.3.5 8-bit to uniform

The change-of-variables formula gives the change in log probability from the transformation of $\mathbf{u}$ to $\mathbf{v}$:

$$\log p(\mathbf{u}) = \log p(\mathbf{v}) + \log \left| \det \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}\mathbf{u}} \right| .$$

Specifically, the change from an 8-bit image to one with pixel values in range [0, 1] is:

$$\begin{aligned}
\mathbf{b}_S^{(p)} &= \frac{\mathbf{a}_S^{(p)}}{256}, \\
&\implies \log p(\mathbf{a}_S) = \log p(\mathbf{b}_S) + \log \left| \det \frac{\mathrm{d}\mathbf{b}}{\mathrm{d}\mathbf{a}} \right| , \\
&\implies \log p(\mathbf{a}_S) = \log p(\mathbf{b}_S) + \log \left(\frac{1}{256} \right)^{D_S} , \\
&\implies \log p(\mathbf{a}_S) = \log p(\mathbf{b}_S) - D_S \log 256. \\
\implies \mathrm{bpd}(\mathbf{a}_S) &= \frac{-\log p(\mathbf{a}_S)}{D_S \log 2}, \\
&= \frac{-(\log p(\mathbf{b}_S) - D_S \log 256)}{D_S \log 2}, \\
&= \frac{-\log p(\mathbf{b}_S)}{D_S \log 2} + \frac{\log 256}{\log 2}, \\
&= \mathrm{bpd}(\mathbf{x}) + 8,
\end{aligned}$$

where bpd($\mathbf{x}$) is given from Equation 4.3.13.

4.3.6 Simple example of density estimation in MRCNF

If Euler method is used as the ODE solver, density estimation Equation 4.2.2 reduces to:

$$\mathbf{v}(t_1) = \mathbf{v}(t_0) + (t_1 - t_0) f_s(\mathbf{v}(t_0), t_0 \mid \mathbf{c}), \tag{4.3.17}$$

where f_s is a neural network, t_0 is the "time" at which the state is image $\mathbf{x}$, and t_1 when it is noise $\mathbf{z}$. We start at scale S with an image sample $\mathbf{x}_S$, and assume $t_0 = 0$ and $t_1 = 1$:

$$\begin{cases} \mathbf{z}_S = \mathbf{x}_S + f_S(\mathbf{x}_S,\, t_0 \mid \mathbf{x}_{S-1}), \\ \mathbf{z}_{S-1} = \mathbf{x}_{S-1} + f_{S-1}(\mathbf{x}_{S-1},\, t_0 \mid \mathbf{x}_{S-2}), \\ \vdots \\ \mathbf{z}_1 = \mathbf{x}_1 + f_1(\mathbf{x}_1,\, t_0 \mid \mathbf{x}_0), \\ \mathbf{z}_0 = \mathbf{x}_0 + f_0(\mathbf{x}_0,\, t_0). \end{cases} \tag{4.3.18}$$

Then, density is estimated using Equation (4.3.10).

4.3.7 Simple example of generation using MRCNF

If Euler method is used as the ODE solver, generation Equation 4.2.2 reduces to:

$$\mathbf{v}(t_0) = \mathbf{v}(t_1) + (t_0 - t_1) f_s(\mathbf{v}(t_1), t_1 \mid \mathbf{c}), \tag{4.3.19}$$

i.e. the state is integrated backwards from t_1 (i.e. $\mathbf{z}_s$) to t_0 (i.e. $\mathbf{x}_s$). We start at scale 0 with a noise sample $\mathbf{z}_0$, and assume t_0 and t_1 are 0 and 1 respectively:

$$\begin{cases} \mathbf{x}_0 = \mathbf{z}_0 - f_0(\mathbf{z}_0,\, t_1), \\ \mathbf{x}_1 = \mathbf{z}_1 - f_1(\mathbf{z}_1,\, t_1 \mid \mathbf{x}_0), \\ \vdots \\ \mathbf{x}_{S-1} = \mathbf{z}_{S-1} - f_{S-1}(\mathbf{z}_{S-1},\, t_1 \mid \mathbf{x}_{S-2}), \\ \mathbf{x}_S = \mathbf{z}_S - f_S(\mathbf{z}_S,\, t_1 \mid \mathbf{x}_{S-1}). \end{cases} \tag{4.3.20}$$

4.4 Related work

Multi-resolution approaches already serve as a key component of state-of-the-art GAN [Denton et al., 2015, Karras et al., 2018, Karnewar and Wang, 2020] and VAE [Razavi et al., 2019, Vahdat and Kautz, 2020] based deep generative models. The idea is to take advantage of the fact that much of the information in an image is contained in a coarsened version, which allows us to deal with simpler problems (coarser images) in a progressive fashion. This helps make models more efficient and effective. Deconvolutional CNNs [Long et al., 2015, Radford et al., 2015] use upsampling layers to generate images more effectively. Modern state-of-the-art generative models have also injected noise at different levels to improve sample quality [Brock et al., 2019, Karras et al., 2020, Vahdat and Kautz, 2020]. Several works [Oord et al., 2016, Reed et al., 2017, Menick and Kalchbrenner, 2019, Razavi et al., 2019] have also shown how the inductive bias of the multi-resolution structure helps alleviate some of the problems of image quality in likelihood-based models.

Several prior works on normalizing flows [Kingma and Dhariwal, 2018, Hoogeboom et al., 2019a,b, Song et al., 2019, Ma et al., 2019, Durkan et al., 2019, Chen et al., 2020, Ho et al., 2019a, Lee et al., 2020, Yu et al., 2020] build on RealNVP [Dinh et al., 2017]. Although they achieve great results in terms of BPD and image quality, they nonetheless report results from significantly higher number of parameters (some with 100x!), and several times GPU hours of training.

STEER [Ghosh et al., 2020] introduced temporal regularization to CNFs by making the final time of integration stochastic. However, we found that this increased training time without significant BPD improvement.

$$\text{STEER [Ghosh et al., 2020]:} \begin{cases} \mathbf{v}(t_1) = \mathbf{v}(t_0) + \int_{t_0}^{T} f(\mathbf{v}(t), t)\ \mathrm{d}t; \\ T \sim \text{Uniform}(t_1 - b, t_1 + b);\ \ b < t_1 - t_0. \end{cases} \tag{4.4.1}$$

"Multiple scales" in prior normalizing flows: Normalizing flows [Dinh et al., 2017, Kingma and Dhariwal, 2018, Grathwohl et al., 2019] try to be "multi-scale" by transforming the input in a smart way (squeezing operation) such that the width of the features progressively reduces in the direction of image to noise, while maintaining the total dimensions. This happens while operating at a *single resolution*. In contrast, our model stacks normalizing flows at *multiple resolutions* in an autoregressive fashion by conditioning on the images at coarser resolutions.

Other classes of generative models that map from a complex distribution to a known noise distribution are Denoising diffusion probabilistic models (DDPM) [Sohl-Dickstein et al., 2015, Ho et al., 2020, Song et al., 2021b] which use a predefined noising process, and score-based generative models [Song and Ermon, 2019, 2020, Jolicoeur-Martineau et al., 2021b, Song et al., 2021b] which estimate the gradient of the log density with respect to the input (i.e. the *score*) of corrupted data with progressively lesser intensities of noise. In contrast, CNFs learn a reversible noising/denoising process using a Neural ODE.

4.4.1 WaveletFlow

WaveletFlow [Yu et al., 2020] is a recent innovation on the normalizing flow, wherein the image is decomposed into a lower-resolution average image, and 3 other informative components using the Discrete Wavelet Transformation. The 3 components at each resolution are mapped to noise using a normalizing flow conditioned on the average image at that resolution. WaveletFlow builds on the Glow [Kingma and Dhariwal, 2018] architecture. It uses an orthogonal transformation, which does not preserve range, and adds a constant term to the log likelihood at each resolution. Best results are obtained when WaveletFlow models with a high parameter count are trained for a long period of time. We fix these issues using our MRCNF.

Comparison to WaveletFlow: We emphasize that there are important and crucial differences between our MRCNF and WaveletFlow. We generalize the notion of a multi-resolution image representation (subsection 4.3.2), and show that Wavelets are one case of this general formulation. WaveletFlow builds on the Glow [Kingma and Dhariwal, 2018] architecture, while ours builds on CNFs [Grathwohl et al., 2019, Finlay et al., 2020]. We also make use of the notion of multi-resolution decomposition of the noise, which is optional, but is not taken into account by WaveletFlow. WaveletFlow uses an orthogonal transformation which does not preserve range ; our MRCNF uses Equation 4.3.4 which is volume-preserving and range-preserving. Finally, WaveletFlow applies special sampling techniques to obtain better samples from its model. We have so far not used such techniques for generation, but we believe they can potentially help our models as well. By making these important changes, we fix many of the previously discussed issues with WaveletFlow. For a more detailed ablation study, please check subsection 4.6.3.

4.5 Experimental details

4.5.1 Models

We used the same neural network architecture as in RNODE [Finlay et al., 2020]. The CNF at each resolution consists of a stack of *bl* blocks of a 4-layer deep convolutional network comprised of 3x3 kernels and softplus activation functions, with 64 hidden dimensions, and time t concatenated to the spatial input. In addition, except at the coarsest resolution, the immediate coarser image is also concatenated with the state. The integration time of each piece is [0, 1]. The number of blocks *bl* and the corresponding total number of parameters are given in Table 4.1.

Table 4.1. Number of parameters for different models with different total number of resolutions (res), and the number of channels (ch) and number of blocks (bl) per resolution.

resolutions	ch	bl	Param
1	64	2	0.16M
	64	4	0.32M
	64	14	1.10M
2	64	8	1.33M
	64	20	3.34M
	64	40	6.68M
3	64	6	1.53M
	64	8	2.04M
	64	20	5.10M

4.5.2 Gradient norm

In order to avoid exploding gradients, We clipped the norm of the gradients [Pascanu et al., 2013] by a maximum value of 100.0. In case of using adversarial loss, we first clip the gradients provided by the adversarial loss by 50.0, sum up the gradients provided by the log-likelihood loss, and then clip the summed gradients by 100.0.

4.6 Experimental results

We train MRCNF models on the CIFAR10 [Krizhevsky et al., 2009a] dataset at finest resolution of 32x32, and the ImageNet [Deng et al., 2009] dataset at 32x32, 64x64, 128x128. We build on top of the code provided in [Finlay et al., 2020][1]. In all cases, we train using *only one* NVIDIA RTX 20280 Ti GPU with 11GB.

In Table 4.2, we compare our results with prior work in terms of (lower is better in all cases) the BPD of the images of the test datasets under the trained models, the number of parameters used by the model, and the number of GPU hours taken to train. The most relevant models for comparison are the 1-resolution FFJORD [Grathwohl et al., 2019] models, and their regularized version RNODE [Finlay et al., 2020], since our model directly converts their architecture into multi-resolution. Other relevant comparisons are previous flow-based methods [Dinh et al., 2017, Kingma and Dhariwal, 2018, Song et al., 2019, Ho et al., 2019a, Yu et al., 2020], however their core architecture (RealNVP [Dinh et al., 2017]) is quite different from FFJORD.

BPD: At lower resolution spaces, we achieve comparable BPDs in less time with far fewer parameters than previous normalizing flows (and non flow-based approaches). However, the power of the multi-resolution formulation is more evident at higher resolutions: we achieve better BPD for ImageNet64 with significantly fewer parameters and less time using only one GPU. A more complete table can be found in the appendix.

Train time: All our experiments used only one GPU, and took significantly less time to train than 1-resolution CNFs, and all prior works including flow-based and non-flow-based models. For example on CIFAR-10, Glow [Kingma and Dhariwal, 2018] used 8 GPUs for 7 days, MintNet [Song et al., 2019] used 2 GPUs for $\approx$ 5 days, 1-resolution FFJORD [Grathwohl et al., 2019] used 6 GPUs for $\approx$ 5 days. All our models used 1 GPU for $\leq$ 1 day.

To make a fair comparison with previous methods, we report the total time taken to train the CNFs of all resolutions one after another on a single GPU. We also maintained the batch size of the finest resolution the same as that in the previous CNF works, but used bigger batch sizes to train coarser resolutions. However, since all the CNFs can be trained in parallel, the actual training time in practice could be much lower.

[1] https://github.com/cfinlay/ffjord-rnode

Table 4.2. Unconditional image generation metrics (lower is better): parameters (P), bits-per-dimension, time (in hours). All our models were trained on only *one* NVIDIA V100 GPU.

	CIFAR10			ImageNet32			ImageNet64		
	BPD	P	Time	BPD	P	Time	BPD	P	Time
Non Flow-based Prior Work									
PixelRNN [Oord et al., 2016]	3.00			3.86			3.63		
Gated PixelCNN [Van den Oord et al., 2016]	3.03			3.83		60	3.57		60
Parallel Multiscale [Reed et al., 2017]				3.95			3.70		
Image Transformer [Parmar et al., 2018]	2.90			3.77					
PixelSNAIL [Chen et al., 2018b]	2.85			3.80					
SPN [Menick and Kalchbrenner, 2019]				3.85	150.0M		3.53	150.0M	
Sparse Transformer [Child et al., 2019]	2.80	59.0M					3.44	152.0M	7days
Axial Transformer [Ho et al., 2019b]				3.76			3.44		
PixelFlow++ [Nielsen and Winther, 2020]	2.92								
NVAE [Vahdat and Kautz, 2020]	2.91		55	3.92		70			
Dist-Aug Sparse Tx [Jun et al., 2020]	2.56	152.0M					3.42	152.0M	
Flow-based Prior Work									
IAF [Kingma et al., 2016]				3.11					
RealNVP [Dinh et al., 2017]	3.49			4.28	46.0M		3.98	96.0M	
Glow [Kingma and Dhariwal, 2018]	3.35	44.0M		4.09	66.1M		3.81	111.1M	
i-ResNets [Behrmann et al., 2019]									
Emerging [Hoogeboom et al., 2019a]	3.34	44.7M		4.09	67.1M		3.81	67.1M	
IDF [Hoogeboom et al., 2019b]	3.34			4.18			3.90		
S-CONF [Karami et al., 2019]	3.34								
MintNet [Song et al., 2019]	3.32	17.9M	≥5days	4.06	17.4M				
Residual Flow [Chen et al., 2019]	3.28			4.01			3.76		
MaCow [Ma et al., 2019]	3.16	43.5M					3.69	122.5M	
Neural Spline Flows [Durkan et al., 2019]	3.38	11.8M					3.82	15.6M	
Flow++ [Ho et al., 2019a]	3.08	31.4M		3.86	169.0M		3.69	73.5M	
ANF [Huang et al., 2020]	3.05			3.92			3.66		
MEF [Xiao and Liu, 2020]	3.32	37.7M		4.05	37.7M		3.73	46.6M	
VFlow [Chen et al., 2020]	2.98			3.83					
Woodbury NF [Lu and Huang, 2020]	3.47			4.20			3.87		
NanoFlow [Lee et al., 2020]	3.25								
ConvExp [Hoogeboom et al., 2020]	3.218								
Wavelet Flow [Yu et al., 2020]				4.08	64.0M		3.78	96.0M	822
TayNODE [Kelly et al., 2020]	1.039								
1-resolution Continuous Normalizing Flow									
FFJORD [Grathwohl et al., 2019]	3.40	0.9M	≥5days	‡3.96	‡2.0M	‡>5days	x		x
RNODE [Finlay et al., 2020]	3.38	1.4M	31.84	‡2.36	2.0M	‡30.1	*3.83	2.0M	*256.4
				§3.49	§1.6M	§40.39			
FFJORD + STEER [Ghosh et al., 2020]	3.40	1.4M	86.34	3.84	2.0M	>5days			
RNODE + STEER [Ghosh et al., 2020]	3.397	1.4M	22.24	2.35	2.0M	24.90			
				§3.49	§1.6M	§30.07			
(Ours) Multi-Resolution Continuous Normalizing Flow (MRCNF)									
2-resolution MRCNF	3.65	1.3M	19.79	3.77	1.3M	18.18	3.44	2.0M	42.30
2-resolution MRCNF	3.54	3.3M	36.47	3.78	6.7M	17.98	x	6.7M	x
3-resolution MRCNF	3.79	1.5M	17.44	3.97	1.5M	13.78	3.55	2.0M	35.39
3-resolution MRCNF	3.60	5.1M	38.27	3.93	10.2M	41.20	x	7.6M	x

- Unreported values. †As reported in Chen et al. [2019]. ‡As reported in Ghosh et al. [2020]. §Re-implemented by us. ‘x’: Fails to train. *RNODE [Finlay et al., 2020] used 4 GPUs to train.

4.6.1 Super-resolution

Our formulation also allows for super-resolution of images (Figure 4.3) free of cost since our framework is autoregressive in resolution. At any stage, one can condition on a ground truth low-resolution image and generate the corresponding high-resolution image.

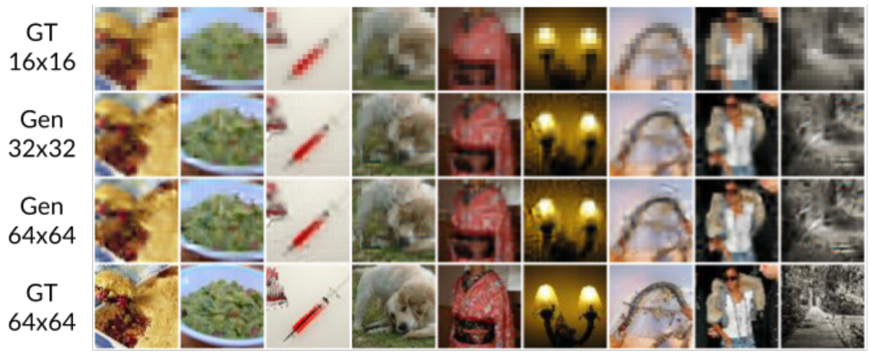


Fig. 4.3. ImageNet: Example of super-resolving to 64x64 from ground truth 16x16. Row 1: ground truth 16x16, Row 2: generated 32x32, Row 3: generated 64x64 Row 4: ground truth 64x64.

4.6.2 Progressive training

We trained an MRCNF model on ImageNet128 by training only the finest resolution (128x128) conditioned on the immediate coarser (64x64) images, and attached it to a 3-resolution model trained on ImageNet64. The resultant 4-resolution ImageNet128 model gives a BPD of 3.31 (Table 4.3) with just 2.74M parameters in ≈60 GPU hours.

Table 4.3. Metrics for unconditional ImageNet128 generation. Param is number of parameters, Time is in hours. '-' indicates unreported values.

ImageNet128 ($\downarrow$)	BPD	Param	Time
Parallel Multiscale [Reed et al., 2017]	3.55	-	-
SPN [Menick and Kalchbrenner, 2019]	3.08	250.00M	-
(Ours) 4-resolution MRCNF	3.31	2.74M	58.59

4.6.3 Ablation study

Our MRCNF method differs from WaveletFlow in three respects:

(1) we use CNFs, while WaveletFlow uses the discrete vairant of normalizing flows,

(2) we use Equation 4.3.4 instead of Equation 4.3.2 as used by WaveletFlow,

(3) we use multi-resolution noise.

We check the individual effects of these changes in an ablation study in Table 4.4, and conclude that:

(1) Simply replacing the normalizing flows in WaveletFlow with CNFs does not produce the best results. It does improve the BPD and training time compared to WaveletFlow.
(2) Using our unimodular transformation in Equation 4.3.4 instead of the original Wavelet Transformation of Equation 4.3.2 not only improves the BPD, it also consistently decreases training time.
(3) As expected, the use of multi-resolution noise does not have a critical impact on either BPD or training time. We use it anyway so as to retain interpretation with 1-resolution models.

Table 4.4. Ablation study across using Wavelet in Equation 4.3.2, and multi-resolution noise formulation in subsection 4.3.4. P is number of parameters, Time is in hours. Lower is better in all cases. '-' indicates unreported values. 'x' : Fails to train3.

	CIFAR10			**ImageNet64**		
(↓)	BPD	P	Time	BPD	P	Time
WaveletFlow [Yu et al., 2020]	-	-	-	3.78	98.0M	822.00
1-resolution CNF (RNODE) [Finlay et al., 2020]	3.38	1.4M	31.84	3.83	2.0M	256.40
2-resolution						
eq. (4.3.2) WaveletFlow with CNF w/o multi-res noise	3.68	1.3M	27.25	x	2.0M	x
eq. (4.3.2) WaveletFlow with CNF w/ multi-res noise	3.69	1.3M	25.88	x	2.0M	x
eq. (4.3.4) MRCNF w/o multi-res noise	3.66	1.3M	19.79	3.48	2.0M	42.33
eq. (4.3.4) MRCNF w/ multi-res noise **(Ours)**	3.65	1.3M	19.69	3.44	2.0M	42.30
3-resolution						
eq. (4.3.2) WaveletFlow with CNF w/o multi-res noise	3.82	1.5M	22.99	3.62	2.0M	43.37
eq. (4.3.2) WaveletFlow with CNF w/ multi-res noise	3.82	1.5M	25.28	3.62	2.0M	44.21
eq. (4.3.4) MRCNF w/o multi-res noise	3.79	1.5M	17.25	3.57	2.0M	35.42
eq. (4.3.4) MRCNF w/ multi-res noise **(Ours)**	3.79	1.5M	17.44	3.55	2.0M	35.39

Thus, our MRCNF model is not a trivial replacement of normalizing flows with CNFs in WaveletFlow. We generalize the notion of multi-resolution image representation, in which the Discrete Wavelet Transform is one of many possibilities. We then derived a unimodular transformation that adds no change in likelihood.

4.6.4 Adversarial loss

Several works [Makhzani et al., 2015, Grover et al., 2018, Lee et al., 2018a, Beckham et al., 2019] have found it useful to add an adversarial loss to pre-existing losses to generate images that better resemble the true data distribution. Similar to [Grover et al., 2018], we conducted experiments with an additional adversarial loss at each resolution. However in our experiments so far, we could achieve neither better BPDs nor better Fréchet Inception Distance (FID)s [Heusel et al., 2017]. As noted in [Theis et al., 2016], since likelihood-based models tend to cover all the modes by minimizing KL-divergence while GAN-based methods tend to mode collapse by minimizing JS-divergence, it is possible that the two approaches are incompatible, and so combining them is not trivial.

4.6.5 FID v/s temperature

Table 4.5 lists the FID values of generated images from MRCNF models trained on CIFAR10, with different temperature settings on the Gaussian.

	Temperature					
	1.0	**0.9**	**0.8**	**0.7**	**0.6**	**0.5**
1-resolution CNF	138.82	147.62	175.93	284.75	405.34	466.16
2-resolution MRCNF	89.55	106.21	171.53	261.64	370.38	435.17
3-resolution MRCNF	88.51	104.39	152.82	232.53	301.89	329.12
4-resolution MRCNF	92.19	104.35	135.58	186.71	250.39	313.39

Table 4.5. FID v/s temperature for MRCNF models trained on CIFAR10.

4.7 Examining Out-of-Distribution (OoD) behaviour

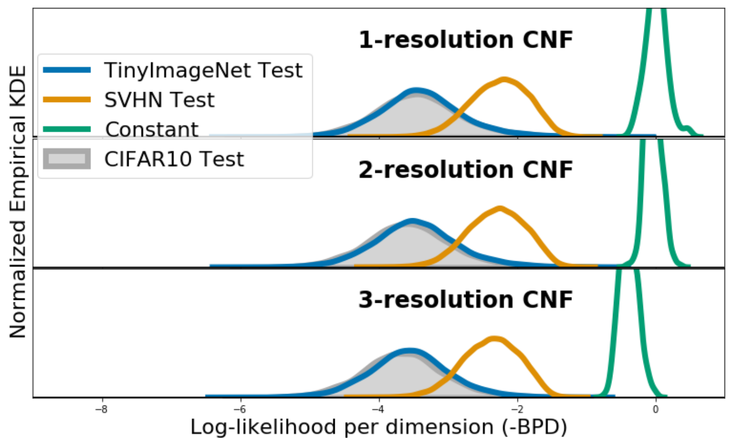


Fig. 4.4. Histogram of log likelihood per dimension i.e. −BPD (estimated using normalized empirical Kernel Density Estimation) of OoD datasets (TinyImageNet, SVHN, Constant) under (MR)CNF models trained on CIFAR10. As with other likelihood-based generative models such as Glow & PixelCNN, OoD datasets have higher likelihood under (MR)CNFs.

The derivation of likelihood-based models suggests that the density of an image under the model is an effective measure of its likelihood of being in-distribution. However, recent works [Theis et al., 2016, Nalisnick et al., 2019a, Serrà et al., 2020, Nalisnick et al., 2019b] have pointed out that it is possible that images drawn from other distributions have higher model likelihood. Examples have been shown where normalizing flow models (such as Glow) trained on CIFAR10 images assign higher likelihood to SVHN [Netzer et al., 2011] images.

This could have serious implications on their practical applicability. Some also note that likelihood-based models do not generate images with good sample quality as they avoid assigning small probability to OoD data points, hence model likelihood (-BPD) is not effective for detecting OoD data in such models.

We conduct the same experiments with (MR)CNFs, and find that similar conclusions can be drawn. Figure 4.4 plots the histogram of log likelihood per dimension (-BPD) of OoD images (SVHN, TinyImageNet) under MRCNF models trained on CIFAR10. It can be observed that the likelihood of the OoD SVHN is higher than CIFAR10 for MRCNF, similar to the findings for Glow, PixelCNN, VAE in earlier works [Nalisnick et al., 2019a, Choi et al., 2018, Serrà et al., 2020, Nalisnick et al., 2019b, Kirichenko et al., 2020].

One possible explanation put forward by Nalisnick et al. [2019b] is that "typical" images are less "likely" than constant images, which is a consequence of the distribution of a Gaussian in high dimensions. Indeed, as our Figure 4.4 shows, constant images have the highest likelihood under MRCNFs, while randomly generated (uniformly distributed) pixels have the least likelihood (not shown in figure due to space constraints).

[Choi et al., 2018, Nalisnick et al., 2019b] suggest using "typicality" as a better measure of OoD. However, [Serrà et al., 2020] observe that the complexity of an image plays a significant role in the training of likelihood-based generative models. They propose a new metric S as an out-of-distribution detector:

$$S(\mathbf{x}) = \mathrm{BPD}(\mathbf{x}) - L(\mathbf{x}). \tag{4.7.1}$$

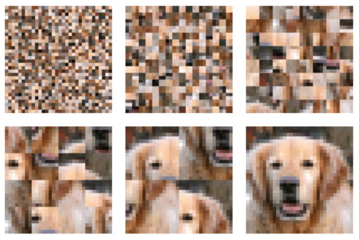
where $L(\mathbf{x})$ is the complexity of an image $\mathbf{x}$ measured as the length of the best compressed version of $\mathbf{x}$ (we use FLIF [Sneyers and Wuille, 2016] following Serrà et al. [2020]) normalized by the number of dimensions.

We perform a similar analysis as Serrà et al. [2020] to test how S compares with -bpd for OoD detection. For different MRCNF models trained on CIFAR10, we compute the area under the receiver operating characteristic curve (auROC) using -bpd and S as standard evaluation for the OoD detection task [Hendrycks et al., 2019, Serrà et al., 2020]. Table 4.6 shows that S does perform better than -bpd in the case of (MR)CNFs, similar to the findings in Serrà et al. [2020] for Glow and PixelCNN++. SVHN seems easier to detect as OoD for Glow than MRCNFs. However, OoD detection performance is about the same for TinyImageNet. We also observe that MRCNFs are better at OoD than CNFs.

Other OoD methods [Hendrycks and Gimpel, 2017, Liang et al., 2018, Lee et al., 2018b, Sabeti and Høst-Madsen, 2019, Høst-Madsen et al., 2019, Hendrycks et al., 2019] are not suitable, as identified in Serrà et al. [2020].

Table 4.6. auROC for OoD detection using -bpd and S[Serrà et al., 2020], for models trained on CIFAR10.

CIFAR10	SVHN		TIN	
(trained)	-BPD	S	-BPD	S
Glow	0.08	0.95	0.66	0.72
1-res CNF	0.07	0.16	0.48	0.60
2-res MRCNF	0.06	0.25	0.46	0.66
3-res MRCNF	0.05	0.25	0.46	0.66



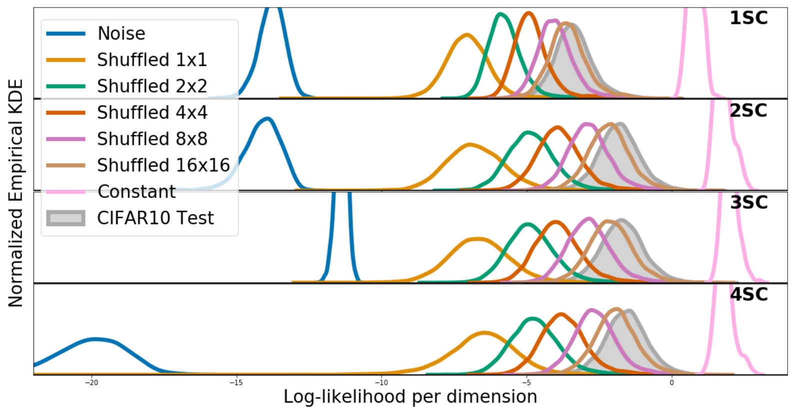


Fig. 4.5. (a) Example of shuffling different-sized patches of a 32x32 image: (left to right, top to bottom) 1x1, 2x2, 4x4, 8x8, 16x16, 32x32 (unshuffled) (b) Histogram of log likelihood per dimension (normalized empirical Kernel Density Estimate) for MRCNF models at different resolutions, trained on CIFAR10.

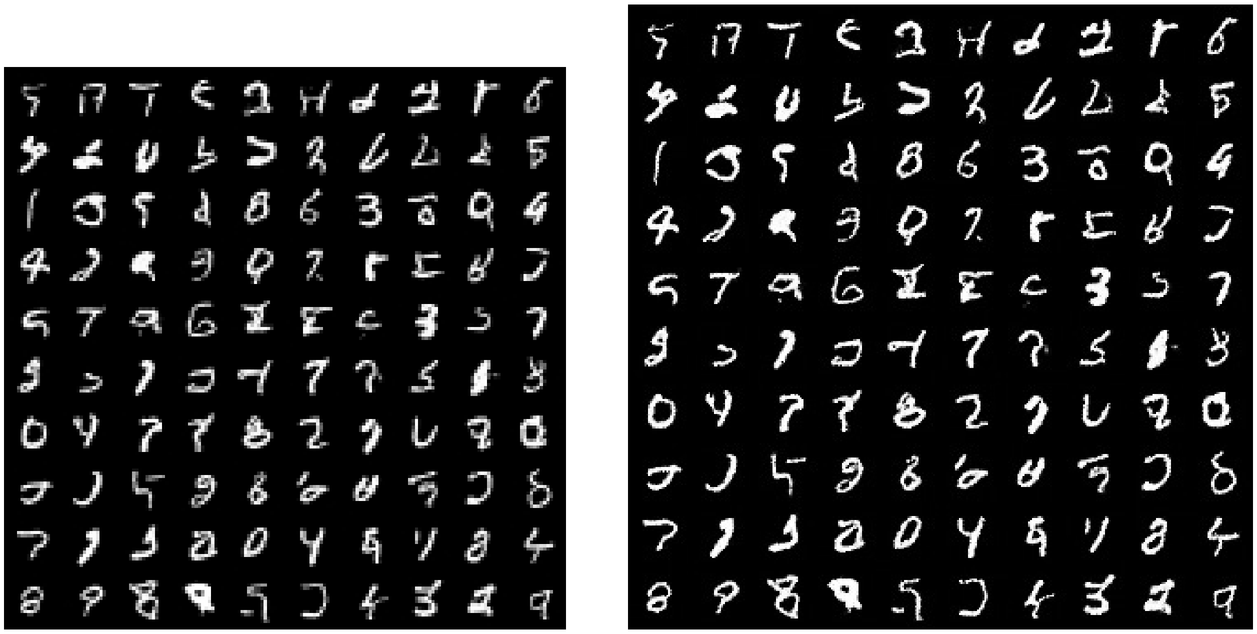
4.7.1 Shuffled in-distribution images

Prior work [Kirichenko et al., 2020] concludes that normalizing flows do not represent images based on their semantic contents, but rather directly encode their visual appearance. We verify this for continuous normalizing flows by estimating the density of in-distribution test images, but with patches of pixels randomly shuffled. Figure 4.5 (a) shows an example of images of shuffled patches of varying size, Figure 4.5 (b) shows the graph of the their log-likelihoods.

That shuffling pixel patches would render the image semantically meaningless is reflected in the FID between CIFAR10-Train and these sets of shuffled images — 1x1: 340.42, 2x2: 299.99, 4x4: 235.22, 8x8: 101.36, 16x16: 33.06, 32x32 (i.e. CIFAR10-Test): 3.15. However, we see that images with large pixel patches shuffled are quite close in likelihood to the unshuffled images (Figure 4.5 (b)), suggesting that since their visual content has not changed much they are almost as likely as unshuffled images under MRCNFs.

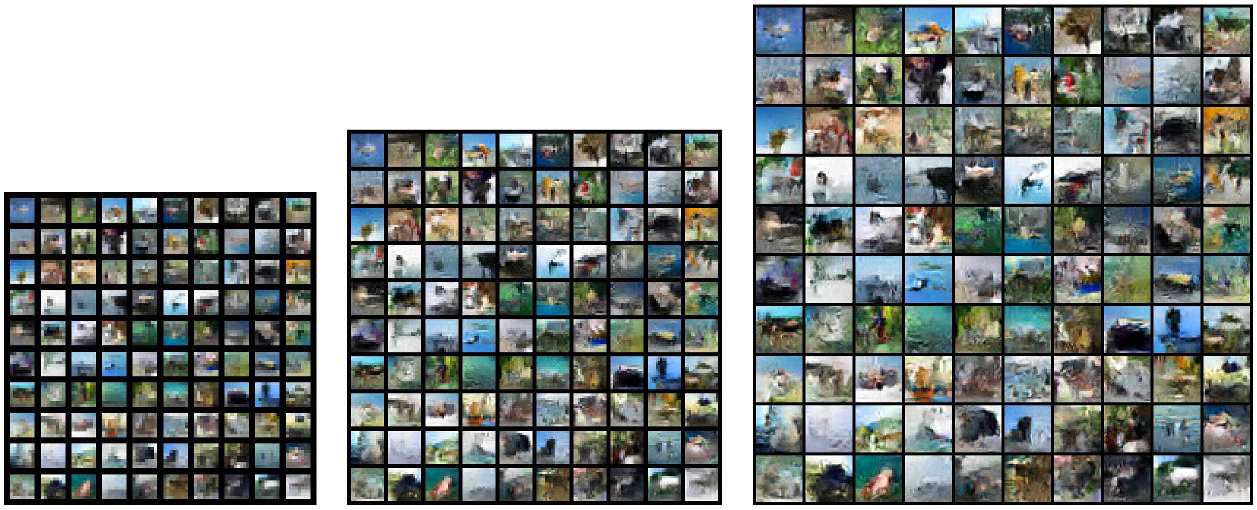
4.8 Qualitative samples

We show qualitative examples from the MNIST [Deng, 2012] and CIFAR10 [Krizhevsky et al., 2009b] in figs. 4.6 and 4.7.



(a) Generated samples at 16x16 **(b)** Corresponding generated samples at 32x32

Fig. 4.6. Generated samples from MNIST at different resolutions.



(a) 8x8 **(b)** 16x16 **(c)** 32x32

Fig. 4.7. Generated samples from CIFAR10 at different resolutions.

4.9 Conclusion

We have presented a Multi-Resolution approach to Continuous Normalizing Flows (MRCNF). MRCNF models achieve comparable or better performance in significantly less training time, training on a single GPU, with a fraction of the number of parameters of other competitive models. Although the likelihood values for 32x32 resolution datasets such as CIFAR10 and ImageNet32 do not improve over the baseline, ImageNet64 and above see a marked improvement. The performance is better for higher resolutions, as seen in the case of ImageNet128. We also conducted an ablation study to note the effects of each change we introduced in the formulation.

In addition, we show that (Multi-Resolution) Continuous Normalizing Flows have similar out-of-distribution properties as other Normalizing Flows.

Chapter 5

Neural Inverse Kinematics with 3D Human Pose Prior [Voleti et al., 2022b]

5.0 Prologue to article

5.0.1 Article details

SMPL-IK: Learned Morphology-Aware Inverse Kinematics for AI Driven Artistic Workflows. Vikram Voleti, Boris Oreshkin, Florent Bocquelét, Felix Harvey, Louis-Simon Ménard, Christopher Pal. *SIGGRAPH Asia 2022 (Technical Communications)*

Personal contribution: The project began with discussions between the authors during Vikram's internship at Unity Technologies partly funded by MITACS, Canada. This project was in collaboration with Boris Oreshkin, and the DeepPose team at Unity Labs, Montreal. Christopher Pal provided advice and guidance throughout the project. Vikram performed a literature survey on the state-of-the-art pose estimation models from image/video, identifying and comparing 20+ datasets of 2D and 3D human pose in various formats, and 20+ methods of 3D human pose estimation. Vikram also learned the technical details of 3D character editing in Unity, 3D graphics pipelines, etc. Boris Oreshkin proposed the main idea of leveraging a 3D human pose model as a prior to improve existing pose estimation methods. Vikram analyzed the literature, identified a strong 3D human pose prior (SMPL), and convinced the team to use SMPL prior for 3D animation tasks moving forward. Boris Oreshkin, Florent Bocquelét, and Vikram collectively worked on integrating SMPL into the team's state-of-the-art human pose estimation model ProtoRes. Vikram identified a huge human pose dataset in the SMPL format called AMASS. Boris and Vikram worked on training SMPL-integrated ProtoRes on AMASS, and testing on other standard datasets (Human3.6M, etc.). Vikram and Boris also worked on stacking an image-to-pose model (ROMP) before this to extract 3D pose from an image, and calculated metrics on standard datasets. Vikram worked with Louis-Simon Ménard on integrating this into the Unity Labs software. Boris and Florent proposed shape inversion to expand the pipeline to work on non-human characters. Project progress was managed and tracked by the full team using JIRA, as well as through weekly team meetings and regular chats on Slack. Vikram made a demo of the full pipeline from image to 3D pose editing of human and non-human 3D characters. The team compiled the work into a research paper published in the Technical Communications track at SIGGRAPH Asia 2022. Vikram made a video for the conference (SIGGRAPH Asia) explaining the paper, methodology, and results.

5.0.2 Context

Inverse Kinematics (IK) systems are often rigid with respect to their input character, thus requiring user intervention to be adapted to new skeletons. Many computer vision pose estimation algorithms naturally operate in the Skinned Multi-Person Linear (SMPL) space, and this extension would open new content authoring opportunities. However, to date SMPL models have not been integrated with advanced machine learning IK tools, and this represents a clear research gap.

5.0.3 Contributions

In this paper we aim at creating a flexible, learned IK solver applicable to a wide variety of human morphologies. We extend a state-of-the-art machine learning IK solver to operate on the well known Skinned Multi-Person Linear model (SMPL). We call our model SMPL-IK, and show that when integrated into real-time 3D software, this extended system opens up opportunities for defining novel AI-assisted animation workflows. For example, when chained with existing pose estimation algorithms, SMPL-IK accelerates posing by allowing users to bootstrap 3D scenes from 2D images while allowing for further editing. Additionally, we propose a novel SMPL Shape Inversion mechanism (SMPL-SI) to map arbitrary humanoid characters to the SMPL space, allowing artists to leverage SMPL-IK on custom characters. In addition to qualitative demos showing proposed tools, we present quantitative SMPL-IK baselines on the H36M and AMASS datasets. Our code is publicly available `https://github.com/boreshkinai/smpl-ik`, and a video explaining the paper at SIGGRAPH Asia 2022 is available at `https://www.youtube.com/watch?v=FixF406owB4`.

5.0.4 Research impact

This work integrated the use of SMPL in the context of pose estimation using ProtoRes. Although there are follow-up works that also show the validity of our approach [Ma et al., 2022], the adoption of our work hasn't been as widespread yet as we had hoped in the research community. Our work at Unity continues to be used in Unity's products, specifically in the Unity Labs software, allowing animators to edit 3D characters with flexibility.

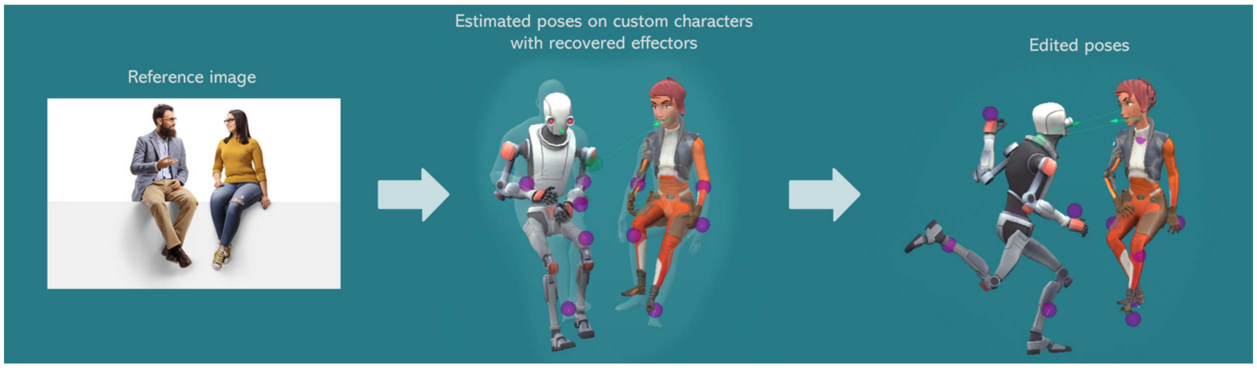

Reference image
Estimated poses on custom characters with recovered effectors
Edited poses

Fig. 5.1. Our approach unlocks novel artistic workflows such as the one depicted above. An animator uses an image to initialize an editable 3D scene. Thus a multi-person 3D scene acquired from an RGB picture is populated with custom user-defined characters whose 3D poses are further edited with the state-of-the-art machine learning inverse kinematics tool integrated in a real-time 3D development software.

5.1 Introduction

Inverse Kinematics (IK) is the problem of estimating 3D positions and rotations of body joints given some end-effector locations [Kawato et al., 1993, Aristidou et al., 2018]. IK is an ill-posed nonlinear problem with multiple solutions. For example, given the 3D location of the right hand, what is a realistic human pose? It has been shown recently that machine learning IK model can be integrated with 3D content authoring user interface to produce a very effective pose authoring tool [Oreshkin et al., 2021, Bocquelet et al., 2022]. Using this tool, an animator provides a terse pose definition via a limited set of positional and angular constraints. The computer tool fills in the rest of the pose, minimizing pose authoring overhead.

The Skinned Multi-Person Linear (SMPL) model is a principled and popular way of jointly modelling human mesh, skeleton and pose [Loper et al., 2015]. It would seem natural to extend this model with inverse kinematics capabilities: making both human shape/mesh and pose editable using independent parameters. Additionally, many computer vision pose estimation algorithms naturally operate in the SMPL space making them natively compatible with a hypothetical SMPL IK model. This extension would open new content authoring opportunities. However, to date SMPL models have not been integrated with advanced machine learning IK tools, and this represents a clear research gap.

In our work we close this gap, exploring and solving two inverse problems in the context of the SMPL human mesh representation: SMPL-IK, an Inverse Kinematics model, and SMPL-SI, a Shape Inversion model. We show how these new components can be used to create new artistic workflows driven by AI algorithms. For example, we demonstrate the tool integrating SMPL-IK and SMPL-SI with an off-the-shelf image-to-pose model, initializing a multi-person 3D scene editable via flexible and easy-to-use user controls.

We demonstrate the tool integrating SMPL-IK with an off-the-shelf image-to-pose model, and use that as a starting point for an editable SMPL character whose gender, shape and pose are editable via flexible and easy-to-use user controls. Furthermore, we show that by including the proposed SMPL-SI model in the workflow we add the additional flexibility of handling custom user supplied characters in the same universal SMPL space by finding an SMPL approximation of the user supplied character via SMPL-SI.

In summary, SMPL-IK accelerates posing by allowing users to bootstrap 3D scenes from 2D images, while allowing for further realistic editing. Additionally, we propose a novel SMPL shape inversion mechanism to map arbitrary humanoid characters to the SMPL space, allowing artists to leverage SMPL-IK on custom 3D characters.

Our main contributions are as follows:

- made SMPL-IK by integrating SMPL into a state-of-the-art 3D human pose estimation model called ProtoRes [Oreshkin et al., 2021].
- achieved 3D human pose estimation in SMPL format with only partial input pose required, by training SMPL-IK on AMASS dataset [Mahmood et al., 2019].
- calculated 3D human pose metrics on standard 3D human pose datasets (AMASS and Human3.6M) in Table 5.2.
- expanded this model's capability to non-human bodies by proposing Shape Inversion.
- stacked an image-to-pose model (ROMP [Sun et al., 2021b]) before this, thus making an 3D human pose estimator from images.
- unified everything into a pipeline that makes a realistic 3D pose editor requiring only partial pose input, initialized by a 3D human pose from an image.

5.2 Background

5.2.1 Skinned Multi-Person Linear model (SMPL)

Skinned Multi-Person Linear model (SMPL) is a realistic 3D human body model based on skinning and blend shapes [Loper et al., 2015]. It is parameterized by two kinds of parameters: shape/beta parameters that control the body shape, and pose parameters that control pose-dependent deformations. SMPL realistically represents a wide range of human body morphologies, and pose-dependent deformations of the body. There have been some extensions to the SMPL model such as SMPL+H [Romero et al., 2017], SMPL-X [Pavlakos et al., 2019], STAR [Osman et al., 2020], etc. SMPL remains a widely accepted model to represent realistic human body pose and is prevalently used for 3D pose estimation of humans in images and video [Bogo et al., 2016, Luo et al., 2020, Li et al., 2021, Rajasegaran et al., 2021, Sun et al., 2021a].

5.2.2 3D human pose datasets

In Table 5.1, we identify various datasets that contain ground truth information on human motion. We also add comments on the type of data contained in the datasets i.e. whether they are images or videos or only 3D content such as Motion Capture (MoCap), whether the dataset contains indoor or outdoor scenes, whether they are of a single person or multiple people. In addition, we specify some details each dataset such as the type of people, how the data was processed, number of images, number of subjects, number of hours of recording, etc.

5.2.3 Inverse Kinematics (IK)

Inverse Kinematics (IK) is the estimation of 3D positions and rotations of body joints given some end-effector locations. It is a prominent problem in robotics and animation, and is traditionally solved by analytical or iterative optimization methods comprehensively reviewed by Aristidou et al. [2018]. Solving IK using machine learning techniques has consistently attracted attention [Bócsi et al., 2011, D'Souza et al., 2001, De Angulo and Torras, 2008], with more work focusing on neural networks based methods [El-Sherbiny et al., 2018, Bensadoun et al., 2022, Mourot et al., 2022].

In the animation space, the current neural IK state-of-the-art is ProtoRes [Oreshkin et al., 2021]. It takes a variable set of effector positions, rotations or look-at targets as inputs, and performs IK to reconstruct all joint locations and rotations. Its effectiveness in editing complex 3D character poses has been demonstrated in a real-time live demo [Bocquelet et al., 2022]. One limitation of ProtoRes is that, being trained on a fixed skeleton, it does not explicitly include any learnt body shape prior.

In this work, we relax this limitation by integrating SMPL into ProtoRes, and training on a large dataset of SMPL human pose data called AMASS [Mahmood et al., 2019].

5.2.4 Retargeting

Retargeting is the task of transferring the pose of a source character to a target character with a different morphology (bone lengths) and possibly a different topology (number of joints, connectivity, etc.) [Gleicher, 1998]. Retargeting is a ubiquitous task in animation, and procedural tools exist for retargeting between skeletons of different morphologies and topologies [3D, 2022].

5.2.5 SMPL and IK

There is very little prior work that attempts to use an IK-enabled SMPL model for 3D character animation. Bebko et al. [2021] pose SMPL characters in the Unity platform, but do not perform any IK. Zhou [2020] performs IK on SMPL parameters using standard optimization, but only in the full pose context, which has very limited applicability for artistic

Table 5.1. Table of 3D human pose datasets

Dataset	**Type of data**	**Comments**
	images + 3D pose	
CMU MoCap data	images + 3D pose (NOT synchronized)	- >11k motions, 40hrs of motion data, 300+ subjects - MoCap NOT synced with images! - GT using cameras, etc.
	video + 3D pose	
HumanEva [Sigal et al., 2010]	video + 3D pose - Single person, indoor	- 40k frames+3D pose @60Hz with 7 Vicon cameras - 30k frames of pure MoCap - GT with MoCap software
H3D (UCB) [Bourdev and Malik, 2009]	video + 3D pose - Single person, outdoor (small dataset)	- 2k frames (1500 train, 500 test) - 19 keypoints (joints, eyes, nose, etc.) - GT from manual annotation
Human 3.6M [Ionescu et al., 2014]	video + 3D pose - Single person, indoor	- 3.6M frames+poses indoor with 4 Vicon cameras - 11 professional actors (5F, 6M) imitate real poses - Hybrid dataset: virtual character in real video - GT using 4 cameras, software, etc.
MPI-INF-3DHP [Mehta et al., 2017]	video + 3D pose - Single person, indoor	- >1.3M frames, 14 cameras (500k frames, 5 cameras at chest height) - 8 actors (4M, 4F), 8 activity sets each of ≈1min - walking, sitting, exercise poses, dynamic actions - more pose classes than Human3.6m - 2 sets of clothing: casual everyday, plain-colored - GT from multi-view marker-less MoCap
TotalCapture [Trumble et al., 2017]	video + 3D pose - Single person, indoor	- 1.9M frames using 8 Vicon cameras - 5 people (4M, 1F) perform 5 actions repeated 3 times - GT 3D MoCap w/ 8 Vicon cameras + IMUs
MuCo-3DHP [Mehta et al., 2018]	video + 3D pose - Multi-person, indoor	- Composited from MPI-INF-3DHP - GT 3D pose using multi-view marker-less MoCap
	video + 3D pose in SMPL format	
UP-3D [Lassner et al., 2017]	video + 3D pose (SMPL) - Single person, outdoor	- 7k frames - 5.5k from LSP, LSP-extended, MPII-HumanPose - 1.5k from FashionPose - GT 3D pose by fitting SMPL on 2D pose
MuPoTS-3D [Mehta et al., 2018] (eval only)	video + 3D pose (SMPL) - Multi-person, indoor & outdoor	- >8k frames, 20 sequences, 8 subjects - GT 3D pose using multi-view marker-less MoCap

Mannequin [Mehta et al., 2018]	video + 3D pose (SMPL) - Multi-person, outdoor - Videos: people as mannequins	- 24k frames, 567 scenes, 742 subjects ($\leq$5 per frame) - Videos from mannequin challenge - GT using optimization of SfM and tracking to SMPL
3DPW [von Marcard et al., 2018]	video + 3D pose (SMPL) - Multi-person, outdoor	- >51k frames, 60 sequences, 1700secs @ 30Hz - 7 actors in 18 clothing styles - GT "MoCap" using IMUs $\rightarrow$ 3D pose by fitting SMPL
SMART [Chen et al., 2021b]	Sport video + 3D pose (SMPL) - Multi-person, indoor	- 45k frames of 30 athletes, 9 activities - GT marker-based MoCap using 12 Vicon cameras $\rightarrow$ SMPL
MoVi [Ghorbani et al., 2021]	video + 3D pose (SMPL) - Multi-person, indoor	- 700k frames with 90 subjects (60 female, 30 male, 5 left-handed) - 20 pre-defined actions and 1 self-chosen movement - 5 data capture rounds, only 'S1' and 'S2' for video+3D - GT using 4 cameras + IMU $\rightarrow$ 3D pose using MoSh++
Only 3D pose		
AMASS [Mahmood et al., 2019]	Only 3D pose (SMPL)	- 42 hours of MoCap, 346 subjects, 11451 motion - Combines CMU, MPI-HDM05, MPIPose Limits, KIT, BioMotion Lab, TCD, ACCAD - SMPL 3D shape (16), DMPL soft tissue coeffs (8), and full SMPL pose (90)
Synchronized Scans and Markers (SSM)	Only 3D pose (MoCap+Body)	Part of AMASS training: dense 3D meshes in motion, with marker-based mocap

pose editing. VPoser [Pavlakos et al., 2019] trains a Variational Auto-Encoder (VAE) to work as a prior on 3D human pose obtained from SMPL. This VAE is used as an iterative IK solver for a pose defined via keypoints. However, the VPoser architecture only works with relatively dense *positional* inputs (no ability to handle sparse heterogeneous effector scenarios has been demonstrated). It also requires on-line L-BFGS optimization, making it too rigid and computationally expensive for pose authoring. There is a clear gap between SMPL and IK: existing IK models suitable for artistic pose editing do not support SMPL, and existing SMPL-based models have insufficient IK cababilities.

5.3 SMPL Inverse Kinematics (SMPL-IK)

We propose SMPL-IK, a learned morphology-aware inverse kinematics module that accounts for SMPL shape and gender information to compute the full pose including the root joint position and 3D rotations of all SMPL joints based on a partially-defined pose specified by SMPL β-parameters (body shape), gender flag, and a few input effectors (positions, rotations, or look-at targets). SMPL-IK supports effector combinations of arbitrary number and type. SMPL-IK extends the learned inverse kinematics model ProtoRes [Oreshkin et al., 2021]. ProtoRes only deals with a fixed morphology scenario in which an ML-based IK model is trained on a fixed skeleton. We remove this limitation by conditioning the ProtoRes computation on the SMPL β-parameters and gender (see Section 5.6.1 for technical details). This results in an IK model that can operate on the wide range of morphologies incorporated in the expansive dataset used to create the SMPL model itself.

There are multiple advantages of this extension, including the following. First, rich public datasets can be used to train a learned IK model, in our case we train on the large AMASS dataset [Mahmood et al., 2019]. Second, an animator can now edit both the pose and the body shape of a flexible SMPL-based puppet using a state-of-the-art learned IK tool, which we demonstrate in Section 5.9.1. Third, training IK in SMPL space unlocks a seamless interface with off-the-shelf AI algorithms operating in a standardized SMPL space, such as computer vision pose estimation backbones.

5.4 SMPL Shape Inversion (SMPL-SI)

SMPL-SI maps arbitrary humanoid skeletons onto their SMPL approximations by learning a mapping from skeleton features to the corresponding SMPL β-parameters (solving the inverse shape problem). Therefore, it can be used to map arbitrary user-supplied skeletons in the SMPL representation, and hence integrate SMPL-IK with custom user skeletons. Recall that the SMPL model implements the following forward equation:

$$\mathbf{p} = \mathrm{SMPL}(\beta, \theta), \tag{5.4.1}$$

mapping shape parameters $\beta \in \mathbb{R}^{10}$ and pose angles $\theta \in \mathbb{R}^{22\times3}$ into SMPL joint positions $\mathbf{p} \in \mathbb{R}^{24\times3}$. Datasets such as H36M contain multiple tuples $(\mathbf{p}_i, \beta_i, \theta_i)$. In principle, the pairs of H36M's skeleton features $\mathbf{f}_i$ extracted from $(\mathbf{p}_i, \theta_i)$ and corresponding labels β_i, could be used for training a shape inversion model:

$$\hat{\beta} = \mathrm{SMPL\text{-}SI}(\mathbf{f}). \tag{5.4.2}$$

However, the H36M training set contains only 6 subjects, meaning the entire dataset has only 6 distinct β_i vectors, thus insufficient for learning any meaningful SMPL-SI.

Accordingly, we propose to train the SMPL-SI model as follows. We randomly sample 20k tuples $(\mathbf{p}_i, \beta_i)$ with $\widetilde{\beta}_i = [\epsilon_i, s_i]$, where $\epsilon_i \in \mathbb{R}^{10}$ is a sample from uniform distribution $\mathcal{U}(-5, 5)$ and $s_i \in \mathbb{R}$ is the scale sampled from $\mathcal{U}(0.2, 2)$. Scale s_i is used to account for the variation in the overall size of the user-supplied characters relative to the standard SMPL model. We then use the SMPL forward equation equation 5.4.1 to compute joint positions $\widetilde{\mathbf{p}}_i$ corresponding to $\widetilde{\beta}_i$ and θ_i set to the T-pose. We then compute skeleton features $\widetilde{\mathbf{f}}_i$ for each $\widetilde{\mathbf{p}}_i$ as distances between the following pairs of joints: (right hip, right knee), (right knee, right ankle), (head, right ankle), (head, right wrist), (right shoulder, right elbow), (right elbow, right wrist). Finally, we implement the kernel density estimator using the 20k samples and the user skeleton features $\mathbf{f}$ to estimate the shape parameters $\widetilde{\beta}$ of the SMPL model:

$$\hat{\beta} = \sum_i \frac{\widetilde{\beta}_i w_i}{\sum_j w_j}, \quad w_i = \kappa((\mathbf{f} - \widetilde{\mathbf{f}}_i)/h). \tag{5.4.3}$$

The implementation uses a Gaussian kernel κ with width $h = 0.02$. The reason for this is that since there can be multiple equally plausible β's for each skeleton (making SMPL-SI an ill-defined problem, like many other inverse problems), a point solution of the inverse problem may be degenerate. To address this, the general solution is formulated in probabilistic Bayesian terms based on the joint generative distribution of skeleton shape and features $p(\widetilde{\beta}, \mathbf{f})$. The Bayesian β-estimator is then derived from the corresponding posterior distribution of β parameters given features:

$$\hat{\beta} = \int \widetilde{\beta} p(\widetilde{\beta}|\mathbf{f}) d\widetilde{\beta}, \tag{5.4.4}$$

Note that $\hat{\beta}$ mixes a few likely values of $\widetilde{\beta}$ corresponding to posterior distribution modes. Decomposing $p(\widetilde{\beta}, \mathbf{f}) = p(\mathbf{f}|\widetilde{\beta})p(\widetilde{\beta})$ we get:

$$\hat{\beta} = \int \frac{\widetilde{\beta} p(\mathbf{f}|\widetilde{\beta}) p(\widetilde{\beta}) d\widetilde{\beta}}{\int p(\mathbf{f}|\widetilde{\beta}) p(\widetilde{\beta}) d\widetilde{\beta}}. \tag{5.4.5}$$

Since the joint distribution $p(\widetilde{\beta}, \mathbf{f})$ is unknown, we approximate it using a combination of kernel density estimation and Monte-Carlo sampling. Assuming conservative uniform prior for $p(\widetilde{\beta})$, we sample β as described above and we use a kernel density estimator $p(\mathbf{f}|\widetilde{\beta}) \approx \frac{1}{hN} \sum_i \kappa(\frac{\mathbf{f} - \widetilde{\mathbf{f}}_i}{h})$. Using this in equation 5.4.5 together with Monte-Carlo sampling from $p(\widetilde{\beta})$, results in equation 5.4.3.

5.5 Proposed workflow

Figure 5.1 presents a high-level summary of the proposed artistic workflow for 3D scene authoring from an image, while Figure 5.2 provides the detailed overview of how it is implemented for a user-defined humanoid character. Section 5.9.1, Section 5.9.3 and Section 5.9.2 depict simpler workflows for authoring SMPL poses, image labeling in the SMPL space and

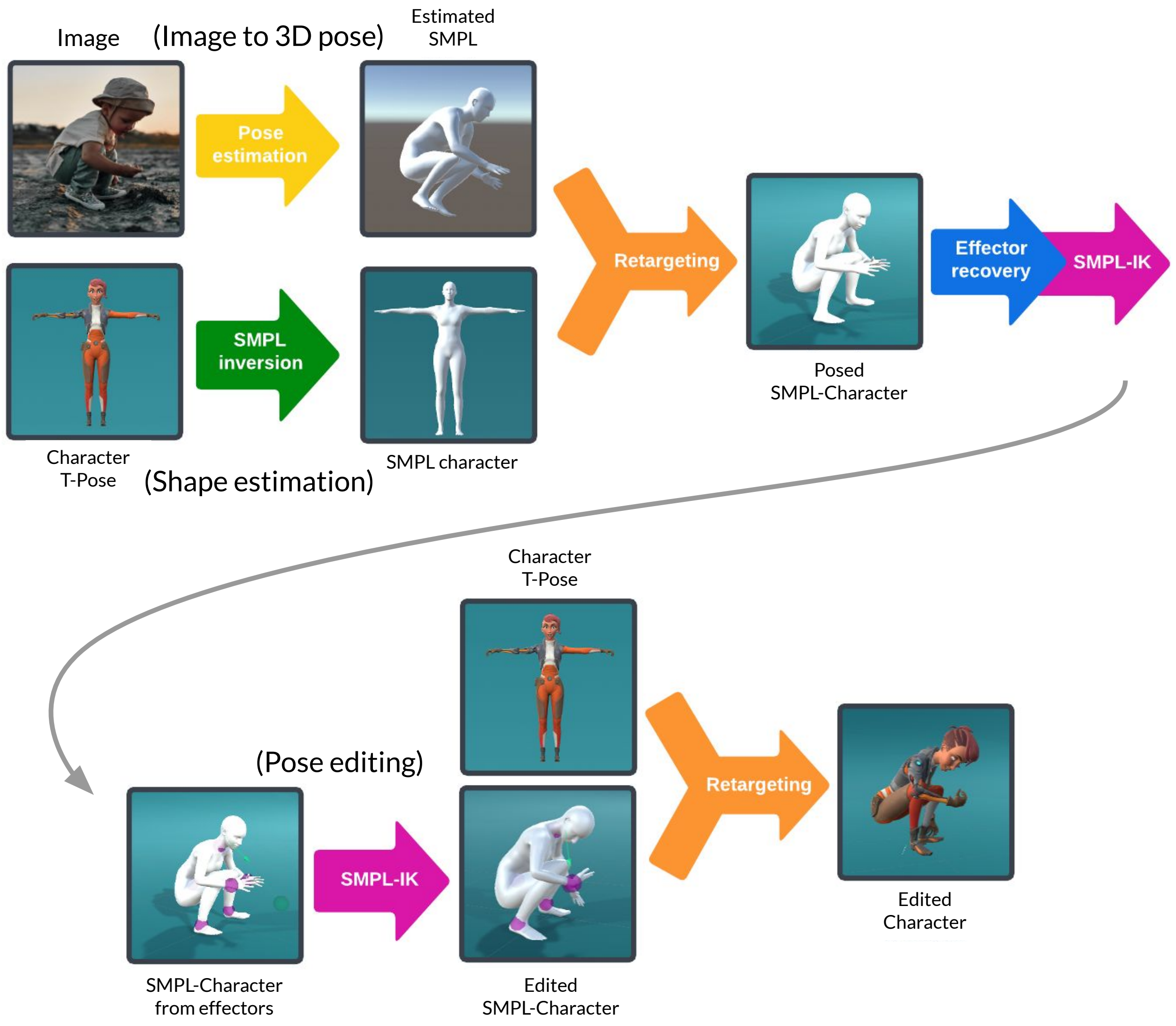


Fig. 5.2. Pipeline for pose estimation and editing from a 2D image to a custom humanoid 3D character in the same pose as the human in the 2D image. Using any advanced animation tool, our approach enables an animator to apply poses extracted from an image to a custom user-defined 3D character and edit them further using SMPL-IK, our state-of-the-art machine learning inverse kinematics tool, integrated in real-time 3D development software (such as Unity as shown here). Given an RGB image of a human in a certain pose, a pose estimation algorithm is used to estimate the human's pose in terms of SMPL parameters. Our novel SMPL-SI then maps the skeleton of the user-defined 3D character onto its SMPL approximation, and the estimated human pose is retargeted onto the SMPL shape approximation obtained from SMPL-SI. This gives an SMPL mesh in the human pose with the morphology of the custom 3D character. From here, the animator can perform pose editing in the SMPL space of pose parameters, using our proposed SMPL-IK, while keeping the shape parameters fixed. To provide the user with effectors that control the pose of the 3D character (shown in purple in the demo videos), our proposed Effector Recovery method extracts only a few effectors to create an editable initial pose, while best preserving the estimated pose.

authoring poses on custom characters from scratch. These were implemented in the 3D real-time Unity engine for validation. These workflows leverage SMPL-IK and SMPL-SI building blocks as well as some others described in the rest of this section.

5.5.1 Overall pipeline

Our overall pipeline from an image with a human to a custom humanoid 3D character in the same pose as the human is as is shown in Figure 5.2. The individual components are expanded upon in the below subsections.

Image to 3D pose: First, given an RGB image of a human in a certain pose, a state-of-the-art pose estimation algorithm is used to extract an estimate of the human's pose in terms of simple parameters. In practice, we use the ROMP model [Sun et al., 2021b] for pose estimation, but any other pose estimation model that outputs SMPL parameters could be used. As mentioned before, SMPL parameters consist of shape/beta parameters that control the body shape, and pose parameters that control pose-dependent deformations.

Custom character shape estimation using SMPL-SI: Then, we use our novel framework of SMPL Shape Inversion (SMPL-SI) to map the skeleton of a user-defined humanoid 3D character onto its SMPL approximation. SMPL-SI learns a mapping from skeleton features to the corresponding simple shape/beta parameters, thus solving the inverse shape problem. These features are the distance between key joints in the canonical pose, like distance between the elbow and the wrist of the character. We achieve this by implementing the kernel density estimator for the shape parameters of the SMPL model, approximating the user-supplied skeleton.

Retarget pose to custom character: We then retarget the initial pose estimation result onto the SMPL shape approximation of the custom 3D character obtained via SMPL-SI. This gives us an SMPL mesh in the pose provided in the image, but with the morphology of the custom 3D character.

Pose editing: From here, we perform pose editing using our proposed SMPL-IK in the SMPL space of pose parameters, keeping the shape parameters fixed. The user is provided with effectors that control the pose of the 3D character (shown in purple in the demo videos). When a user edits the pose of one or more effectors, a new full pose is estimated by SMPL-IK.

SMPL-IK computes the full pose, including the root joint position and 3D rotations of all SMPL joints, based on a partially defined pose specified by SMPL beta parameters, that is the body shape, gender flag, and a few input effectors, such as the positions, rotations, or look-at targets. SMPL-IK supports effector combinations of arbitrary number and type.

Effector recovery: We also propose effector recovery to extract only a few effectors to create an editable initial pose, while best preserving the estimated pose. More effectors means better reconstruction at the cost of less freedom to the posing model.

Hence, given the edited SMPL character with the new full pose, this pose is then retargeted back to the custom 3D character. Because the morphology of the SMPL character is similar to that of the custom character, this retargeting best preserves the edited pose.

Each of these stages is expanded upon below.

5.5.2 Image to 3D pose

We propose to process a monocular RGB image to initialize an editable 3D scene as shown in Figure 5.1. Several methods exist for pose estimation from RGB inputs, most recent of which include ROMP [Sun et al., 2021a] and HybrIK [Li et al., 2021]. In our approach, we use a pre-trained ROMP model that predicts shape, 3D joint rotations and 3D root joint location for each human instance in the image. The outputs of pose estimation can be directly used to edit the estimated 3D SMPL mesh using SMPL-IK, leading to advanced 3D labelling tools that can be used to refine pose estimation and augmented reality datasets, as described in Section 5.9.2. Alternatively, pose estimation results can be retargeted to user-supplied 3D characters, in which case the 3D scene with retargeted characters is further edited through the combination of SMPL-IK and SMPL-SI as explained below.

5.5.3 Custom character shape estimation using SMPL-SI

Pose estimation algorithms output pose in the standardized SMPL space, whereas users may wish to repurpose the pose towards their own custom character. We use SMPL-SI to find the best approximation of the custom character by estimating its corresponding SMPL β parameters from the custom skeleton features (e.g. certain bone lengths). The SMPL character created using SMPL-SI provides a good approximation of the user character hence providing for its smooth integration with SMPL-IK, operating in the standard SMPL space.

5.5.4 Retarget pose to custom character

In Figure 5.2, procedural retargeting first retargets the initial pose estimation result onto the SMPL approximation of the user-defined character obtained via SMPL-SI. Second, it retargets the pose edited by the animator with SMPL-IK back on the user character. On both occasions, SMPL-SI makes the job of procedural retargeting easier. First, it aligns the topology of user character with the SMPL space. Second, the SMPL character derived via SMPL-SI is a close approximation of the user character, simplifying the transfer of the pose edited with SMPL-IK back onto the user character.

5.5.5 Pose editing

Pose editing relies on the Unity UX integration of SMPL-IK similar to one of ProtoRes and augmented with the SMPL shape editing controls as well as pose estimation, SMPL-SI, retargeting and effector recovery integrations. Editing happens directly in the user character space following the WYSIWYG paradigm. A full pipeline demo is presented in Section 5.9.5.

5.5.6 Effector recovery

Pose estimation outputs a full pose (24 3D joint angles and 3D root joint location) of each human in the scene. The pose editing process constrained by this information would be very tedious. SMPL-IK makes pose authoring efficient using very sparse constraints (e.g. using 5-6 effectors). Therefore, we propose to extract only a few effectors to create an editable initial pose. We call this *Effector Recovery*, which proceeds starting from an empty set of effectors, given the full pose provided by the computer vision backbone, in an iterative greedy fashion. Out of the remaining effectors, we add one at a time, run a new candidate effector configuration through SMPL-IK, and obtain the pose reconstructed from this configuration. We then choose a new effector configuration by retaining the candidate effector set that minimizes the L2 joint reconstruction error in the character space. We repeat this process until either the maximum number of allowed effectors is reached, or the reconstruction error falls below a fixed threshold. We find this greedy algorithm very effective in producing a minimalistic set of effectors most useful in retaining the initial pose, which is shown in supplementary video discussed in Section 5.9.4.

5.6 SMPL-IK details

5.6.1 SMPL-IK neural network diagram

Figure 5.3 shows the overall architecture of the model. The inputs to the model are a variable number of 3D positions, rotations, look-at (direction of the head), the tolerance in the estimation, the ID of the joints being given as input, their type (position or rotation or look-at), and the SMPL parameters of body shape and gender. These inputs are fed to a variant of the ProtoRes model [Oreshkin et al., 2021]. The power of the ProtoRes model is that it is capable of handling a variable number of inputs using a specific architecture in the Pose Encoder module. Then, a Pose Decoder module transforms the features encoded by the Pose Encoder into meaningful outputs for the full pose : all joint positions and rotations. The Pose Decoder consists of a global position decoder, an inverse kinematics decoder that outputs the joint rotations, and a forward kinematics decoder that outputs the joint positions.

It is in the Pose Decoder that we incorporate SMPL. The inverse and forward kinematics are handled by the relevant equations in SMPL, conditioned on the body shape and gender provided by the user.

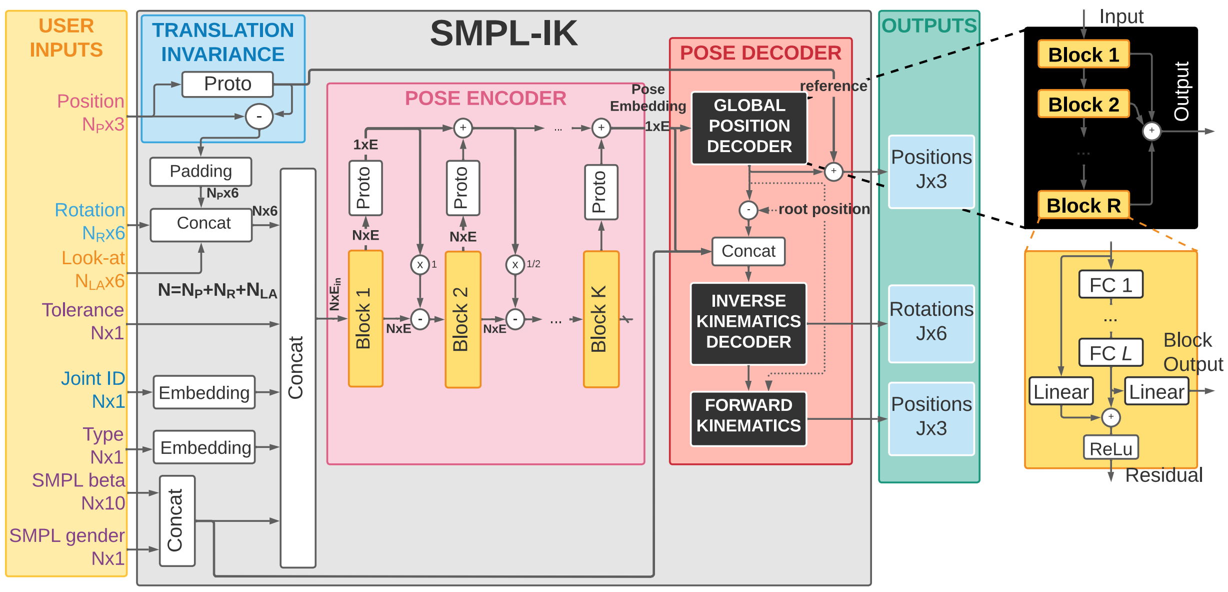


Fig. 5.3. SMPL-IK neural network diagram. Note input conditioning on β and gender inputs.

5.6.2 SMPL-IK training details

The training process overall follows that of ProtoRes [Oreshkin et al., 2021]. We found that naively training on AMASS works well, while for H36M, simply training on its 6 subjects results in the lack of generalization in the β subspace of inputs. To overcome this, we used the following β augmentation strategy. For each sample drawn from the H36M dataset, we added white Gaussian noise with unit variance, and recalculated joint positions based on the augmented β value and the pose θ from the dataset. The model was trained on the augmented H36M dataset. We found that overall, the model quality was better when it was trained on the AMASS dataset, although the quality of the H36M model was also acceptable.

5.6.3 SMPL-IK evaluation details

The datasets used to measure quantitative generalization results are H36M and AMASS. We used H36M train and test splits derived in ROMP [Sun et al., 2021a], which in turn follow H36M Protocol 2 (subjects S1, S5, S6, S7, S8 for training and S9, S11 for test, plus 1:10 subsampling of the training set). For AMASS, we take the train/validation/test splits from Mahmood et al. [2019] (valdation datasets: HumanEva, MPI_HDM05, SFU, MPI_mosh; test datasets: Transitions_mocap, SSM_synced; training datasets: everything else).

The evaluation metrics we chose to quantify SMPL-IK are commonly used in the context of H36M and AMASS datasets: MPJPE, PA-MPJPE, and the geodesic rotation error, which was shown to be important in quantifying the quality of realistic poses in Oreshkin et al. [2021]. The metrics are defined as follows.

Mean Per Joint Position Error (MPJPE) is computed by flattening all poses and joints into the leading dimension, resulting in the ground truth $\mathbf{p} \in \mathbb{R}^{N\times 3}$ and its prediction $\widehat{\mathbf{p}}$:

$$\mathrm{MPJPE}(\mathbf{p}, \widehat{\mathbf{p}}) = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{p}_i - \widehat{\mathbf{p}}_i\|_2. \tag{5.6.1}$$

PA-MPJPE, Procrustes aligned MPJPE, is MPJPE calculated after each estimated 3D pose in the batch is aligned to its respective ground truth by the Procrustes method, which is simply a similarity transformation.

GE, geodesic error, between a rotation matrix $\mathbf{R}$ and its prediction $\widehat{\mathbf{R}}$, Salehi et al. [2018]:

$$\mathrm{GE}(\mathbf{R}, \widehat{\mathbf{R}}) = \arccos\left[(\mathrm{tr}(\widehat{\mathbf{R}}^T\mathbf{R}) - 1)/2\right]. \tag{5.6.2}$$

All metrics in Table 5.2 are computed on test sets of AMASS and H36M using models trained on respective training sets using the randomized effector benchmark framework described in detail in Oreshkin et al. [2021]. Notably, we evaluate the model's performance by assessing pose reconstruction quality from sparse variable inputs. We randomly sample 2 to 64 effectors to be used as inputs and average reconstruction errors across multiple iterations.

5.7 Empirical results

Table 5.2. SMPL-IK benchmark following the randomized effector scheme [Oreshkin et al., 2021] on AMASS and H36M datasets, based on MPJPE (Mean Per Joint Position Error), PA-MPJPE (Procrustes-Aligned MPJPE), and GE (Geodesic Error) metrics.

AMASS			H36M		
MPJPE	PA-MPJPE	GE	MPJPE	PA-MPJPE	GE
59.3	52.5	0.1602	65.8	57.9	0.224

In Table 5.2, we report pose reconstruction errors of our SMPL-IK approach for two datasets: AMASS [Mahmood et al., 2019] and Human3.6M [Ionescu et al., 2014] (see Section 5.6.3 for more details). Since the evaluation metrics are reconstruction errors from sparse inputs, it is not possible to compare with prior methods that don't use this ramndomized effector scheme [Oreshkin et al., 2021].

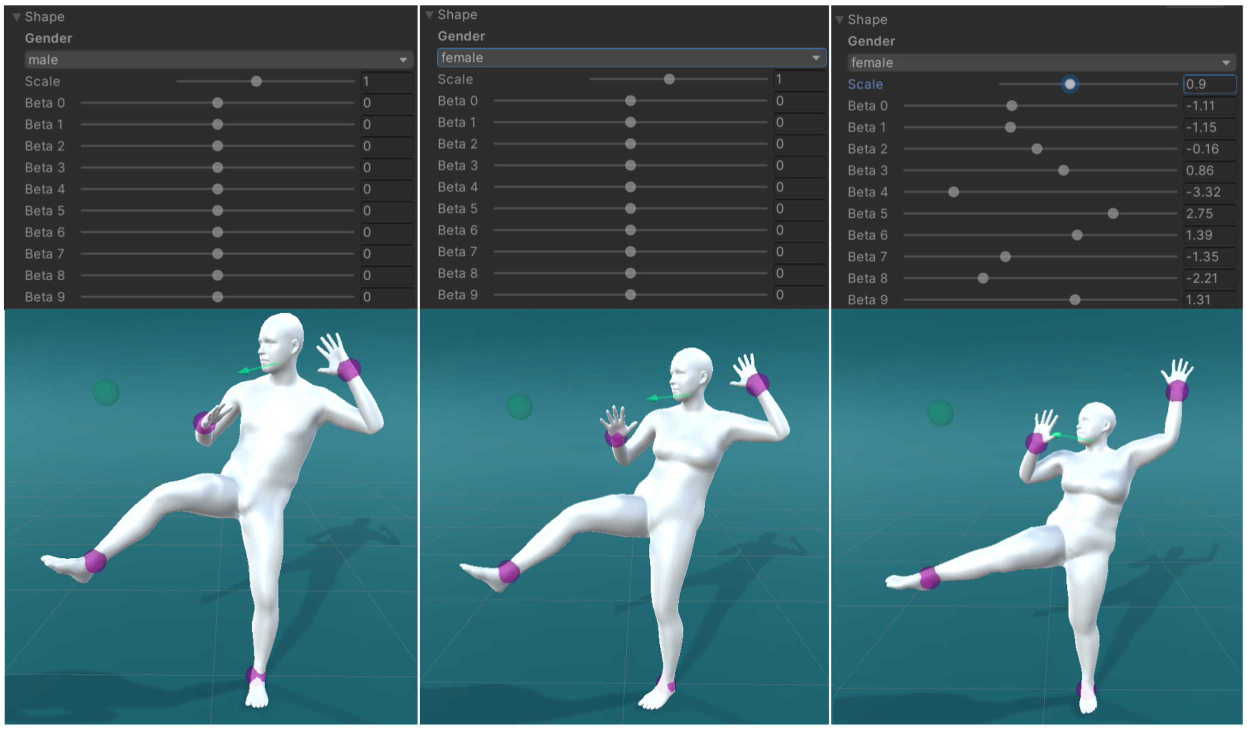
5.8 Limitations

SMPL-IK and SMPL-SI are most effective when dealing with realistic human shapes and poses, because they are trained on the SMPL model and realistic 3D pose data from the AMASS dataset. Obviously, they perform worse when dealing with unrealistic and disproportionate human body types, such as those of certain cartoon characters. SMPL-SI relies on a set of joints to compute user character features. These joints are present in most characters, but without them its operation is not viable.

5.9 Demos

All demo videos are available here: `https://drive.google.com/drive/u/1/folders/1bHwoZjAX9njFCGszzLpUtOFGXxsOsWKW`.

5.9.1 Editing both shape and pose demo



Shape
Gender
male
Scale 1
Beta 0 0
Beta 1 0
Beta 2 0
Beta 3 0
Beta 4 0
Beta 5 0
Beta 6 0
Beta 7 0
Beta 8 0
Beta 9 0

Shape
Gender
female
Scale 1
Beta 0 0
Beta 1 0
Beta 2 0
Beta 3 0
Beta 4 0
Beta 5 0
Beta 6 0
Beta 7 0
Beta 8 0
Beta 9 0

Shape
Gender
female
Scale 0.9
Beta 0 -1.11
Beta 1 -1.15
Beta 2 -0.16
Beta 3 0.86
Beta 4 -3.32
Beta 5 2.75
Beta 6 1.39
Beta 7 -1.35
Beta 8 -2.21
Beta 9 1.31

Fig. 5.4. Morphology-aware learned IK. Left: posing the average male SMPL character. Center: result of modifying only the SMPL gender parameter. Right: result of additionally modifying the SMPL β shape parameters.

This is shown in the video `Demo_Pose_and_Shape_Editing.mp4`. Compared to ProtoRes, SMPL-IK adds the additional flexibility of editing body shape together with pose. Figure 5.4 demonstrates the user interface of shape editing, including the gender setting and the controls for the 10 SMPL β parameters. In addition, the demo video shows how pose and shape of the SMPL character can be edited simultaneously. In this video, we demonstrate the benefit of SMPL-IK in pose authoring. First, we show how different effectors can be successfully manipulated using SMPL-IK leading to different realistic poses of the same body. Then, we show how changing body type, described by gender and scale, leads to different realistic versions of the same pose for different bodies. Finally, we show fine-grained modification of the body type by manipulating the SMPL shape parameters of the body. At every step, the corresponding pose is estimated using our SMPL-IK approach.

5.9.2 Labeling tool demo

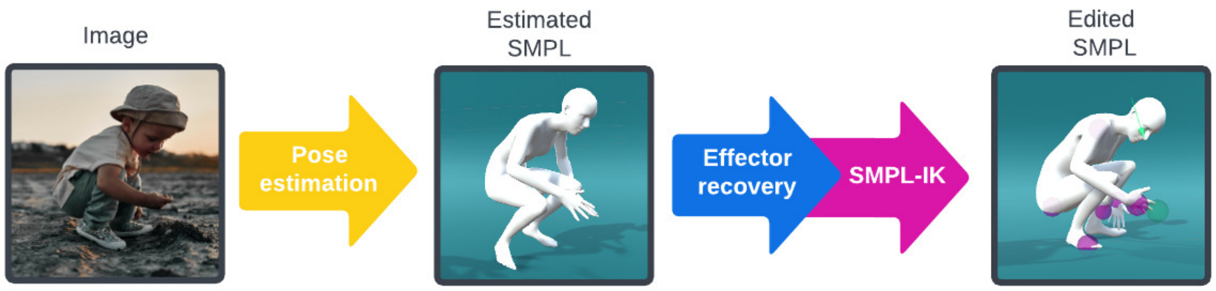


Fig. 5.5. Pipeline for 2D image labeling with accurate 3D pose.

This is shown in the video `Demo_Labeling_Tool.mp4`. One of the drawbacks of the current pose estimation datasets based on real data is that only 3D or 2D positions of joints are actually labeled. However, it was shown that rotations are very important for representing a believable naturally looking pose [Oreshkin et al., 2021]. SMPL-IK can be used as a labeling tool to add the missing 3D-rotation information to existing datasets, elevating them to the next level with minimal human effort. Given an image of a human, our SMPL-IK approach (combined with an off-the-shelf image-to-pose estimator) provides an editable 3D SMPL model in a pose close to the one in the image (see Figure 5.5). The labeling tool based on SMPL-IK and its integration with Unity can be used to correct the joint rotations and specify the correct lookat (head/eyes direction) that is most often estimated incorrectly by the current state-of-the-art pose estimation algorithms due to the absence of this information in the current pose estimation datasets.

5.9.3 Pose authoring on a custom character

Figure 5.6 depicts the simplified workflow that is used for authoring a pose for the custom user defined character using a combination of SMPL-IK and SMPL-SI. Supplementary videos `Demo_Authoring_Pose_Child.mp4`, `Demo_Authoring_Pose_Child.mp4`, `Demo_Authoring_Pose_Female.mp4`, `Demo_Authoring_Pose_Male.mp4`, `Demo_Authoring_Pose_Strong.mp4` show how SMPL-SI can be used to manipulate 4 custom characters (child, female, male and strong male) with different proportions and morphologies.

5.9.4 Effector recovery

The video `Demo_Effector_Recovery.mp4` demonstrates the effector recovery mechanism in action. It shows the effect of changing the maximum number of effectors hyperparameter as well as the effect of changing number of recovered effectors on the initial pose extracted from image. It is clear that a relatively small number of effectors are sufficient to recover a good initialization for the editable pose.

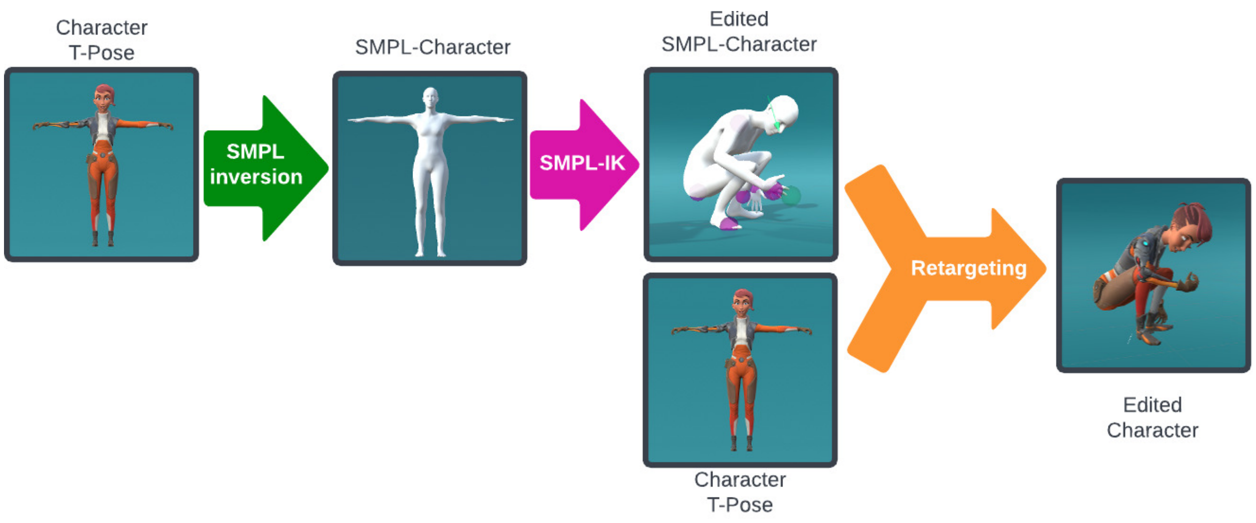


Fig. 5.6. Pose authoring with a custom humanoid character via SMPL-IK and SMPL-SI.

5.9.5 Full pipeline demos

5.9.5.1 `Demo_Crouch_FineTuning.mp4` :

In this video, we show how to edit a pose in Unity using our approach. First, given a user-provided 3D character, SMPL-SI is used to estimate the SMPL body shape parameters that best fit the character. This SMPL-Character is shown in the video transparently along with the 3D character, and is also shown in the second image in Section 5.9.3. Then, given an image of a human in a pose, such as the crouched baby in Section 5.9.2, an off-the-shelf image-to-pose estimator is used to obtain its SMPL pose parameters. Then, the SMPL-Character is retargeted onto the estimated pose. Next, for further editing of the character from the new pose, Effector Recovery is performed to recover the best effectors that describe that pose for that character. The effectors are shown in purple. These effectors can now be used to edit the pose as the user wishes. Optionally, more effectors could be activated for further fine-tuned editing, including both positional and rotational effectors.

5.9.5.2 `Demo_Sitting_Editing.mp4` :

In this video, we demonstrate the case shown in Figure 5.1, with two humanoid 3D characters. As image of two people sitting is loaded, the poses of the two people are estimated using an off-the-shelf image-to-pose model, and the two 3D characters are retargeted to these estimated poses. Further, the pose of the 3D characters are then edited by manipulating the effectors. The video shows the various effectors and the effect of manipulating them. Every manipulation uses our SMPL-IK approach to estimate the realistic pose of that character.

5.10 Conclusion

In this work, we introduce SMPL-IK, an integration of SMPL into ProtoRes, a state-of-the-art inverse kinematics-based 3D human pose estimation model. Using this integrated model, we achieved full 3D human pose estimation from only partial input pose by training SMPL-IK on AMASS dataset. This included a thorough investigation into publicly available 3D human pose datasets. We then expanded this model's capability to non-human bodies by proposing Shape Inversion, a retargeting technique. We also extended its capability to 3D human pose estimation from images, by stacking an image-to-pose model (ROMP) before SMPL-IK. Our model also allows pose editing by running SMPL-IK on newer poses formed by change in few pose effectors. We also proposed Effector Recovery, an iterative method to extract the minimal set of pose effectors that critically define a full pose. Finally, we unified everything into a pipeline that makes a realistic 3D pose editor requiring only partial pose input, initialized by a 3D human pose from an image.

Chapter 6

Non-Isotropic Denoising Diffusion Models [Voleti et al., 2022c]

6.0 Prologue to article

6.0.1 Article details

Score-based Denoising Diffusion with Non-Isotropic Gaussian Noise Models. Vikram Voleti, Christopher Pal, Adam Oberman. *Advances in Neural Information Processing Systems (NeurIPS) 2022 Workshop.*

Personal contribution: The project began with discussions between the authors at Mila. Vikram worked on deriving a non-isotropic formulation of the noise process in denoising diffusion models. Adam Oberman and Christopher Pal provided advice and guidance throughout the project, including explaining the mathematical details of Stochastic Differential Equations, providing text books and source material to understand Gaussian processes better. Vikram derived the relevant equations for the isotropic Gaussian formulations (DDPM and SMLD covered in Chapter 2), then derived the equations for the non-isotropic Gaussian variant. Adam Oberman provided the literature on Gaussian Free Fields (GFFs). Vikram Voleti wrote the code for the non-isotropic variant of DDPM, GFF, and conducted experiments on image generation.

6.0.2 Context

Generative models based on denoising diffusion techniques have led to an unprecedented increase in the quality and diversity of imagery that is now possible to create with neural generative models. However, most contemporary state-of-the-art methods are derived from a standard isotropic Gaussian formulation. A non-isotropic Gaussian variant had not been explored yet.

6.0.3 Contributions

In this work, we examine the situation where non-isotropic Gaussian distributions are used. We present the key mathematical derivations for creating denoising diffusion models using an underlying non-isotropic Gaussian noise model. We also provide initial experiments with the CIFAR10 dataset to help verify empirically that this more general modelling approach can also yield high-quality samples.

6.0.4 Research impact

This work derived the use of non-isotropic Gaussian process in the context of denoising diffusion models. Other works then derived the use of non-Gaussian noise processes such as Gamma noise [Nachmani et al., 2021], Poisson noise [Xu et al., 2022], Heat dissipation process [Rissanen et al., 2022]. The results on image generation using this non-isotropic formulation are by themselves not much better than those using the isotropic formulation. However, other work has utilized the real power of this formulation, and expanded the modality to a continuous domain i.e. a function space. Thus, as the forward process perturbs input functions gradually using a Gaussian process, instead of the data itself, it is now possible to model infinite-dimensional data using denoising diffusion models. This is shown by Hagemann et al. [2023], Bond-Taylor and Willcocks [2023], and concurrently by Lim et al. [2023] — a project Vikram contributed to in collaboration with NVIDIA.

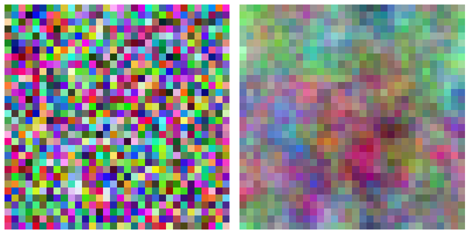

(a) Isotropic (b) Non-isotropic

Fig. 6.1. Gaussian noise samples.

6.1 Introduction

Score-based denoising diffusion models Song and Ermon [2019], Ho et al. [2020], Song et al. [2021b] have seen great success as generative models for images Dhariwal and Nichol [2021], Song and Ermon [2020], as well as other modes such as video Ho et al. [2022b], Yang et al. [2022], Voleti et al. [2022a], audio Kong et al. [2021], Chen et al. [2021a], etc. The underlying framework relies on a noising "forward" process that adds noise to real images (or other data), and a denoising "reverse" process that iteratively removes noise. In most cases, the noise distribution used is the isotropic Gaussian i.e. noise samples are independently and identically distributed (IID) as the standard normal at each pixel.

In this work, we lay the theoretical foundations and derive the key mathematics for a non-isotropic Gaussian formulation for denoising diffusion models. It is our hope that these insights may open the door to new classes of models. One type of non-isotropic Gaussian noise arises in a family of models known as Gaussian Free Fields (GFFs) Sheffield [2007], Berestycki [2015], Bramson et al. [2016], Werner and Powell [2020] (a.k.a. Gaussian Random Fields). GFF noise can be obtained by either convolving isotropic Gaussian noise with a filter, or applying frequency masking of noise. In either case this procedure allows one to model or generate smoother and correlated types of Gaussian noise. In Figures 6.1 and 6.3, we compare examples of isotropic Gaussian noise with GFF noise obtained using a frequency space window function consisting of $w(f) = \frac{1}{f}$.

Our contributions here consist of the following: (1) deriving the key mathematics for score-based denoising diffusion models using non-isotropic multivariate Gaussian distributions, (2) examining the special case of a GFF and the corresponding non-Isotropic Gaussian noise model, and (3) showing that diffusion models trained (eg. on the CIFAR-10 dataset Krizhevsky et al. [2009a]) using a GFF noise process are also capable of yielding high-quality samples comparable to models based on isotropic Gaussian noise.

Section 6.2 and Section 6.3 contain detailed derivations of our Non-Isotropic DDPM (NI-DDPM) and Non-Isotropic SMLD (NI-SMLD) denoising diffusion models. Section 6.4 provides a direct comparison between DDPM [Ho et al., 2020] and our NI-DDPM. Section 6.5 derives Gaussian Free Fields (GFFs) in connection with the previous sections. Section 6.6 provides results of image generation experiments using DDPM and our NI-DDPM.

6.2 Non-isotropic DDPM (NI-DDPM)

6.2.1 Forward (data to noise) for NI-DDPM

For a fixed sequence of positive scales $0 < \beta_1 < \cdots < \beta_L < 1$, $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$, the **transition "forward"** process is:

$$p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \boldsymbol{\Sigma}) \implies \quad \mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\sqrt{\boldsymbol{\Sigma}}\mathbf{z}_{t-1}, \tag{6.2.1}$$

where $\sqrt{\boldsymbol{\Sigma}}$ is the matrix square root of $\boldsymbol{\Sigma}$ (e.g. as given by Cholesky decomposition).

Note : $\sqrt{\boldsymbol{\Sigma}}$ is *not* the element-wise square root of $\boldsymbol{\Sigma}$.

The **cumulative "forward"** process can be derived as:

$$p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\boldsymbol{\Sigma}). \tag{6.2.2}$$

$$\implies \mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon} \implies \boldsymbol{\epsilon} = \sqrt{\boldsymbol{\Sigma}^{-1}}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}. \tag{6.2.3}$$

6.2.2 Score for NI-DDPM

$$\nabla_{\mathbf{x}_t} \log p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = -\boldsymbol{\Sigma}^{-1}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{1-\bar{\alpha}_t} = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon}. \tag{6.2.4}$$

Derivation of the score value:

$$\begin{aligned}
&p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\boldsymbol{\Sigma}),\\
&= \frac{1}{(2\pi)^{D/2}((1-\bar{\alpha}_t)|\boldsymbol{\Sigma}|)^{1/2}}\exp\left(-\frac{1}{2(1-\bar{\alpha}_t)}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)\right).\\
\implies &\log p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = -\log((2\pi)^{D/2}((1-\bar{\alpha}_t)|\boldsymbol{\Sigma}|)^{1/2})\\
&\qquad\qquad - \frac{1}{2(1-\bar{\alpha}_t)}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0),\\
\implies &\nabla_{\mathbf{x}_t}\log p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = -\frac{1}{2(1-\bar{\alpha}_t)}2\boldsymbol{\Sigma}^{-1}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon}.
\end{aligned}$$

6.2.3 Score-matching objective for NI-DDPM

The objective for score estimation in NI-DDPM at noise level $\bar{\alpha}_t$ is:

$$\begin{aligned}
\ell^{\text{NI-DDPM}}(\boldsymbol{\theta}; \bar{\alpha}_t) &\triangleq \frac{1}{2}\mathbb{E}_{p_t^{\text{NI-DDPM}}(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t) + \boldsymbol{\Sigma}^{-1}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{1-\bar{\alpha}_t}\right\|_2^2\right], \qquad (6.2.5)\\
&\triangleq \frac{1}{2}\mathbb{E}_{p_t^{\text{NI-DDPM}}(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t) + \frac{1}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon}\right\|_2^2\right].
\end{aligned}$$

6.2.4 Variance of score for NI-DDPM

$$\begin{aligned}\mathbb{E}\left[\left\|\nabla_{\mathbf{x}_t}\log q^{\text{NI-DDPM}}_{\bar{\alpha}_t}(\mathbf{x}_t \mid \mathbf{x}_0)\right\|_2^2\right], &= \mathbb{E}\left[\left\|-\boldsymbol{\Sigma}^{-1}\frac{\mathbf{x}_t-\sqrt{\bar{\alpha}_t}\mathbf{x}_0}{1-\bar{\alpha}_t}\right\|_2^2\right],\\ &= \mathbb{E}\left[\left\|\boldsymbol{\Sigma}^{-1}\frac{\sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}}{1-\bar{\alpha}_t}\right\|_2^2\right] = \frac{1}{1-\bar{\alpha}_t}\boldsymbol{\Sigma}^{-1}\mathbb{E}\left[\|\boldsymbol{\epsilon}\|_2^2\right] = \frac{1}{1-\bar{\alpha}_t}\boldsymbol{\Sigma}^{-1}.\end{aligned} \tag{6.2.6}$$

6.2.5 Overall objective function for NI-DDPM

The overall objective function weights the score-matching objective by the inverse of the variance of the score at each time step:

$$\mathcal{L}^{\text{NI-DDPM}}(\boldsymbol{\theta};\{\bar{\alpha}_t\}_{t=1}^L) \triangleq \mathbb{E}_t\ \lambda(\bar{\alpha}_t)\ \ell^{\text{NI-DDPM}}(\boldsymbol{\theta};\bar{\alpha}_t). \tag{6.2.7}$$

We consider three possibilities for the loss weight $\lambda(\bar{\alpha}_t)$:

(a) $\boldsymbol{\lambda_a(\bar{\alpha}_t) = (1-\bar{\alpha}_t)\Sigma}$.

$$\begin{aligned}\mathcal{L}_a^{\text{NI-DDPM}}(\boldsymbol{\theta};\{\bar{\alpha}_t\}_{t=1}^L) &\triangleq \mathbb{E}_{t,p_t(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\|\sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t)+\sqrt{\boldsymbol{\Sigma}^{-1}}\frac{(\mathbf{x}_t-\sqrt{\bar{\alpha}_t}\mathbf{x}_0)}{\sqrt{1-\bar{\alpha}_t}}\right\|_2^2\right],\\ &= \mathbb{E}_{t,\boldsymbol{\epsilon},\mathbf{x}_0}\left[\left\|\sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t)+\boldsymbol{\epsilon}\right\|_2^2\right].\end{aligned} \tag{6.2.8}$$

(b) $\boldsymbol{\lambda_b(\bar{\alpha}_t) = (1-\bar{\alpha}_t)}$.

$$\begin{aligned}\mathcal{L}_b^{\text{NI-DDPM}}(\boldsymbol{\theta};\{\bar{\alpha}_t\}_{t=1}^L) &\triangleq \mathbb{E}_{t,p_{\bar{\alpha}_t}(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\|\sqrt{1-\bar{\alpha}_t}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t)+\boldsymbol{\Sigma}^{-1}\frac{(\mathbf{x}_t-\sqrt{\bar{\alpha}_t}\mathbf{x}_0)}{\sqrt{1-\bar{\alpha}_t}}\right\|_2^2\right],\\ &= \mathbb{E}_{t,\boldsymbol{\epsilon},\mathbf{x}_0}\left[\left\|\sqrt{1-\bar{\alpha}_t}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t)+\sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon}\right\|_2^2\right].\end{aligned} \tag{6.2.9}$$

(c) $\boldsymbol{\lambda_c(\bar{\alpha}_t) = (1-\bar{\alpha}_t)\Sigma^2}$.

$$\begin{aligned}\mathcal{L}_c^{\text{NI-DDPM}}(\boldsymbol{\theta};\{\bar{\alpha}_t\}_{t=1}^L) &\triangleq \mathbb{E}_{t,p_t(\mathbf{x}_t|\mathbf{x}_0)p(\mathbf{x}_0)}\left[\left\|\sqrt{1-\bar{\alpha}_t}\boldsymbol{\Sigma}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t)+\frac{(\mathbf{x}_t-\sqrt{\bar{\alpha}_t}\mathbf{x}_0)}{\sqrt{1-\bar{\alpha}_t}}\right\|_2^2\right],\\ &= \mathbb{E}_{t,\boldsymbol{\epsilon},\mathbf{x}_0}\left[\left\|\sqrt{1-\bar{\alpha}_t}\boldsymbol{\Sigma}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t)+\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}\right\|_2^2\right].\end{aligned} \tag{6.2.10}$$

6.2.6 Noise-matching objective for NI-DDPM

A score model that matches the actual score-noise relationship in eq. (6.2.4) is:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t). \tag{6.2.11}$$

In this case, the overall objective function changes to the noise-matching objective:

$$\mathcal{L}_a^{\text{NI-DDPM}}(\boldsymbol{\theta};\{\bar{\alpha}_t\}_{t=1}^L) \triangleq \mathbb{E}_{t,\boldsymbol{\epsilon},\mathbf{x}_0}\left[\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t,\bar{\alpha}_t)\|_2^2\right]. \tag{6.2.12}$$

$$\mathcal{L}_b^{\text{NI-DDPM}}(\boldsymbol{\theta}; \{\bar{\alpha}_t\}_{t=1}^L) \triangleq \mathbb{E}_{t,\boldsymbol{\epsilon},\mathbf{x}_0}\left[\left\|\sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon} - \sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t)\right\|_2^2\right]. \tag{6.2.13}$$

$$\mathcal{L}_c^{\text{NI-DDPM}}(\boldsymbol{\theta}; \{\bar{\alpha}_t\}_{t=1}^L) \triangleq \mathbb{E}_{t,\boldsymbol{\epsilon},\mathbf{x}_0}\left[\left\|\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon} - \sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, \bar{\alpha}_t)\right\|_2^2\right]. \tag{6.2.14}$$

6.2.7 Reverse (noise to data) for NI-DDPM

The goal is to estimate the **reverse** transition probability $q_t(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$. This is intractable, but it is possible to estimate it conditioned on $\mathbf{x}_0$ i.e. $q_t(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$.

We know from the forward process that:

$$p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\boldsymbol{\Sigma}).$$

$$\implies \hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t)). \tag{6.2.15}$$

From Bayes' theorem, we compute the parameters of $q_t(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ i.e. the reverse process additionally conditioning on $\mathbf{x}_0$, with the help of Bishop and Nasrabadi [2006] 2.116. For a variable $\mathbf{u}$ distributed as a normal with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Lambda}^{-1}$, and a dependent variable $\mathbf{v}$ conditionally distributed as a normal with mean $\mathbf{Au}+\mathbf{b}$ and covariance matrix $\mathbf{L}^{-1}$, the marginal distribution of $\mathbf{v}$, and the other conditional distribution $p(\mathbf{u} \mid \mathbf{v})$ are given as:

$$\boxed{\begin{aligned} p(\mathbf{u}) &= \mathcal{N}(\mathbf{u} \mid \boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}),\\ p(\mathbf{v} \mid \mathbf{u}) &= \mathcal{N}(\mathbf{v} \mid \mathbf{Au}+\mathbf{b}, \mathbf{L}^{-1})\\ \implies p(\mathbf{v}) &= \mathcal{N}(\mathbf{v} \mid \mathbf{A}\boldsymbol{\mu}+\mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^T),\\ \implies p(\mathbf{u} \mid \mathbf{v}) &= \mathcal{N}(\mathbf{u} \mid \mathbf{C}(\mathbf{A}^T\mathbf{L}(\mathbf{v}-\mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}), \mathbf{C}), \mathbf{C} = (\boldsymbol{\Lambda} + \mathbf{A}^T\mathbf{L}\mathbf{A})^{-1}. \end{aligned}}$$

For NI-DDPM, $p(\mathbf{u}) = p_{t-1}^{\text{NI-DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_0), p(\mathbf{v}) = p_t^{\text{NI-DDPM}}(\mathbf{x}_t)$:

$$\begin{aligned} p_{t-1}^{\text{NI-DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_{t-1} \mid \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\boldsymbol{\Sigma}).\\ p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t \mid \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\boldsymbol{\Sigma}).\\ \implies \text{Given } p(\mathbf{u}) &= p_{t-1}^{\text{NI-DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_0), p(\mathbf{v} \mid \mathbf{u}) = p_t^{\text{NI-DDPM}}(\mathbf{x}_t \mid \mathbf{x}_0),\\ &\text{we need } p(\mathbf{u} \mid \mathbf{v}) = q_{t-1}^{\text{NI-DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0).\\ \implies \boldsymbol{\mu} &= \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, \boldsymbol{\Lambda}^{-1} = (1-\bar{\alpha}_{t-1})\boldsymbol{\Sigma}, \mathbf{A} = \sqrt{1-\beta_t}, \mathbf{b} = \mathbf{0}, \mathbf{L}^{-1} = \beta_t\boldsymbol{\Sigma}.\\ \implies \mathbf{C} &= \left(\frac{1}{1-\bar{\alpha}_{t-1}}\boldsymbol{\Sigma}^{-1} + (1-\beta_t)\frac{1}{\beta_t}\boldsymbol{\Sigma}^{-1}\right)^{-1} = \left(\frac{\cancel{\beta_t} + 1 - \cancel{\beta_t} - \alpha_t}{(1-\bar{\alpha}_{t-1})\beta_t}\boldsymbol{\Sigma}^{-1}\right)^{-1},\\ &= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\boldsymbol{\Sigma} = \tilde{\beta}_t\boldsymbol{\Sigma}.\\ \implies \mathbf{C}(\mathbf{A}^T\mathbf{L}(\mathbf{y}-\mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}) &= \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t\boldsymbol{\Sigma}\left(\sqrt{1-\beta_t}\frac{1}{\beta_t}\boldsymbol{\Sigma}^{-1}\mathbf{x}_t + \frac{1}{1-\bar{\alpha}_{t-1}}\boldsymbol{\Sigma}^{-1}\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\right), \end{aligned}$$

$$= \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t.$$

Thus, the parameters of the distribution of the reverse process are:

$$\therefore q_t^{\text{NI-DDPM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1} \mid \tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_{t-1}\boldsymbol{\Sigma}) \text{ , where}$$

$$\tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t; \quad \tilde{\beta}_{t-1} = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \tag{6.2.17}$$

$$\Longrightarrow \mathbf{x}_{t-1} = \tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \hat{\mathbf{x}}_0) + \sqrt{\tilde{\beta}_{t-1}}\sqrt{\boldsymbol{\Sigma}}\mathbf{z}, \tag{6.2.18}$$

where

$$\tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \hat{\mathbf{x}}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\left(\frac{1}{\sqrt{\bar{\alpha}_t}}\Big(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}^*\Big)\right) + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t,$$

$$= \frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}}\frac{\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_t - \frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}}\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}^* + \frac{\sqrt{1-\beta_t}}{1-\bar{\alpha}_t}\Big(1-\bar{\alpha}_{t-1}\Big)\mathbf{x}_t,$$

$$= \frac{1}{\sqrt{1-\beta_t}}\frac{\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{1-\beta_t}}{1-\bar{\alpha}_t}\left(1-\frac{\bar{\alpha}_t}{1-\beta_t}\right)\mathbf{x}_t - \frac{1}{\sqrt{1-\beta_t}}\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}^*,$$

$$= \frac{1}{\sqrt{1-\beta_t}}\left(\frac{\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{1-\beta_t}{1-\bar{\alpha}_t}\left(1-\frac{\bar{\alpha}_t}{1-\beta_t}\right)\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}^*\right),$$

$$= \frac{1}{\sqrt{1-\beta_t}}\left(\cancel{\frac{\cancel{\beta_t} + 1 - \cancel{\beta_t} - \bar{\alpha}_t}{1-\bar{\alpha}_t}}\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}^*\right),$$

$$\Longrightarrow \tilde{\boldsymbol{\mu}}_{t-1}(\mathbf{x}_t, \hat{\mathbf{x}}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)\right). \tag{6.2.19}$$

(6.2.20)

$$\Longrightarrow \mathbf{x}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}}\ \boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)\right) + \sqrt{\tilde{\beta}_{t-1}}\sqrt{\boldsymbol{\Sigma}}\ \mathbf{z}, \tag{6.2.21}$$

$$= \frac{1}{\sqrt{1-\beta_t}}\Big(\mathbf{x}_t + \beta_t\boldsymbol{\Sigma}\ \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t)\Big) + \sqrt{\tilde{\beta}_{t-1}}\sqrt{\boldsymbol{\Sigma}}\ \mathbf{z}. \tag{6.2.22}$$

6.2.8 Sampling in NI-DDPM

We perform sampling at each time step in 2 parts:

Step 1 (from eq. (6.2.15)*):* $\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t)).$ (6.2.23)

Step 2 (from eq. (6.2.18)*):* $\mathbf{x}_{t-1} = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\hat{\mathbf{x}}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \sqrt{\tilde{\beta}_{t-1}}\sqrt{\boldsymbol{\Sigma}}\mathbf{z}.$ (6.2.24)

6.2.9 Sampling in NI-DDPM using DDIM

DDIM replaces *Step 2* with the NI-DDPM forward process eq. (6.2.3):

$$\textit{Step 1: } \hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t)). \tag{6.2.25}$$

$$\textit{Step 2: } \mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t). \tag{6.2.26}$$

This is derived from the following distributions from Song et al. [2021a]:

$$p_L^{\text{NI-DDPM}}(\mathbf{x}_L \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_L \mid \sqrt{\bar{\alpha}_L}\mathbf{x}_0, (1-\bar{\alpha}_L)\boldsymbol{\Sigma}). \tag{6.2.27}$$

$$q_t^{\text{NI-DDIM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1} \mid \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{\sqrt{1-\bar{\alpha}_t}}, \mathbf{0}\right). \tag{6.2.28}$$

$$\Longrightarrow p_t^{\text{NI-DDIM}}(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t \mid \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\boldsymbol{\Sigma}\right). \tag{6.2.29}$$

6.2.10 Expected Denoised Sample (EDS) for NI-DDPM

From Saremi and Hyvarinen [2019], we know that the expected denoised sample $\mathbf{x}_0^*(\mathbf{x}_t, \bar{\alpha}_t) \triangleq \mathbb{E}_{\mathbf{x}_0 \sim q_t(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}_0]$ and the optimal score $\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t)$ are related as (as mentioned earlier in eq. (2.4.41)):

$$\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t) = \mathbb{E}\left[\|\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t \mid \mathbf{x}_0)\|_2^2\right](\mathbf{x}_0^*(\mathbf{x}_t, \bar{\alpha}_t) - \mathbf{x}_t). \tag{6.2.30}$$

For NI-DDPM with non-isotropic Gaussian noise of covariance $(1-\bar{\alpha}_t)\boldsymbol{\Sigma}$,

$$\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t) = \frac{1}{1-\bar{\alpha}_t}\boldsymbol{\Sigma}^{-1}(\mathbf{x}_0^*(\mathbf{x}_t, \bar{\alpha}_t) - \mathbf{x}_t). \tag{6.2.31}$$

$$\Longrightarrow \mathbf{x}_0^*(\mathbf{x}_t, \bar{\alpha}_t) = \mathbf{x}_t + (1-\bar{\alpha}_t)\boldsymbol{\Sigma}\ \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, \bar{\alpha}_t) = \mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\ \boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t). \tag{6.2.32}$$

6.2.11 SDE formulation : Non-Isotropic Variance Preserving (NIVP) SDE

For NI-DDPM i.e. Non-Isotropic Variance Preserving (NIVP) SDE, the forward equation and transition probability are derived (below) as:

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\ \mathrm{d}t + \sqrt{\beta(t)}\sqrt{\boldsymbol{\Sigma}}\ \mathrm{d}\mathbf{w}. \tag{6.2.33}$$

$$p_{0t}^{\text{NIVP}}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}\left(\mathbf{x}(t) \mid \mathbf{x}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, \boldsymbol{\Sigma}(\mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s})\right). \tag{6.2.34}$$

6.2.11.1 Derivations :

Forward process: We know from eq. (6.2.1) that:

$$\mathbf{x}_t = \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{t-1}.$$
$$\Longrightarrow\ \mathbf{x}(t+\Delta t) = \sqrt{1-\beta(t+\Delta t)\Delta t}\ \mathbf{x}(t) + \sqrt{\beta(t+\Delta t)\Delta t}\ \sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}(t),$$

$$\approx \left(1 - \frac{1}{2}\beta(t+\Delta t)\Delta t\right)\mathbf{x}(t) + \sqrt{\beta(t+\Delta t)\Delta t}\ \sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}(t),$$
$$\approx \mathbf{x}(t) - \frac{1}{2}\beta(t)\Delta t\ \mathbf{x}(t) + \sqrt{\beta(t)\Delta t}\ \sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}(t).$$
$$\implies \mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\ \mathrm{d}t + \sqrt{\beta(t)}\sqrt{\boldsymbol{\Sigma}}\ \mathrm{d}\mathbf{w}. \tag{6.2.35}$$

Mean (from eq. 5.50 in Sarkka & Solin (2019)):

$$\mathrm{d}\mathbf{x} = \mathbf{f}\ \mathrm{d}t + \mathbf{G}\ \mathrm{d}\mathbf{w} \implies \frac{\mathrm{d}\boldsymbol{\mu}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}].$$
$$\therefore \frac{\mathrm{d}\boldsymbol{\mu}_{\text{NI-DDPM}}(t)}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[-\frac{1}{2}\beta(t)\mathbf{x}] = -\frac{1}{2}\beta(t)\mathbb{E}_{\mathbf{x}}(\mathbf{x}) = -\frac{1}{2}\beta(t)\boldsymbol{\mu}_{\text{NI-DDPM}}(t),$$
$$\implies \frac{\mathrm{d}\boldsymbol{\mu}_{\text{NI-DDPM}}(t)}{\boldsymbol{\mu}_{\text{NI-DDPM}}(t)} = -\frac{1}{2}\beta(t)\mathrm{d}t \implies \log \boldsymbol{\mu}_{\text{NI-DDPM}}(t)|_0^t = -\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s,$$
$$\implies \log \boldsymbol{\mu}_{\text{NI-DDPM}}(t) - \log \boldsymbol{\mu}(0) = -\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s \implies \log \frac{\boldsymbol{\mu}_{\text{NI-DDPM}}(t)}{\boldsymbol{\mu}(0)} = -\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s,$$
$$\implies \boldsymbol{\mu}_{\text{NI-DDPM}}(t) = \boldsymbol{\mu}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}.$$

Covariance (from eq. 5.51 in Sarkka & Solin (2019)):

$$\mathrm{d}\mathbf{x} = \mathbf{f}\ \mathrm{d}t + \mathbf{G}\ \mathrm{d}\mathbf{w} \implies \frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{cov}}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}(\mathbf{x}-\boldsymbol{\mu})^T] + \mathbb{E}_{\mathbf{x}}[(\mathbf{x}-\boldsymbol{\mu})\mathbf{f}^T] + \mathbb{E}_{\mathbf{x}}[\mathbf{G}\mathbf{G}^T].$$
$$\therefore \frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{NI-DDPM}}(t)}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[-\frac{1}{2}\beta(t)\mathbf{x}\mathbf{x}^T] + \mathbb{E}_{\mathbf{x}}[\mathbf{x}(-\frac{1}{2}\beta(t)\mathbf{x})^T] + \mathbb{E}_{\mathbf{x}}[\sqrt{\beta(t)}\sqrt{\boldsymbol{\Sigma}}\sqrt{\beta(t)}\sqrt{\boldsymbol{\Sigma}}],$$
$$= -\beta(t)\boldsymbol{\Sigma}_{\text{NI-DDPM}}(t) + \beta(t)\boldsymbol{\Sigma} = \beta(t)(\boldsymbol{\Sigma} - \boldsymbol{\Sigma}_{\text{NI-DDPM}}(t)),$$
$$\implies \frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{NI-DDPM}}(t)}{\boldsymbol{\Sigma} - \boldsymbol{\Sigma}_{\text{NI-DDPM}}(t)} = \beta(t)\mathrm{d}t \implies -\log(\boldsymbol{\Sigma} - \boldsymbol{\Sigma}_{\text{NI-DDPM}}(t))|_0^t = \int_0^t \beta(s)\mathrm{d}s,$$
$$\implies -\log(\boldsymbol{\Sigma} - \boldsymbol{\Sigma}_{\text{NI-DDPM}}(t)) + \log(\boldsymbol{\Sigma} - \boldsymbol{\Sigma}_{\mathbf{x}}(0)) = \int_0^t \beta(s)\mathrm{d}s,$$
$$\implies \frac{\boldsymbol{\Sigma} - \boldsymbol{\Sigma}_{\text{NI-DDPM}}(t)}{\boldsymbol{\Sigma} - \boldsymbol{\Sigma}_{\mathbf{x}}(0)} = e^{-\int_0^t \beta(s)\mathrm{d}s} \implies \boldsymbol{\Sigma}_{\text{NI-DDPM}}(t) = \boldsymbol{\Sigma} - e^{-\int_0^t \beta(s)\mathrm{d}s}(\boldsymbol{\Sigma} - \boldsymbol{\Sigma}_{\mathbf{x}}(0)),$$
$$\implies \boldsymbol{\Sigma}_{\text{NI-DDPM}}(t) = \boldsymbol{\Sigma} + e^{-\int_0^t \beta(s)\mathrm{d}s}(\boldsymbol{\Sigma}_{\mathbf{x}}(0) - \boldsymbol{\Sigma}).$$

For each data point $\mathbf{x}(0)$, $\boldsymbol{\mu}(0) = \mathbf{x}(0)$, $\boldsymbol{\Sigma}_{\mathbf{x}}(0) = \mathbf{0}$:

$$\implies \boldsymbol{\mu}_{\text{NI-DDPM}}(t) = \mathbf{x}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s},$$
$$\boldsymbol{\Sigma}_{\text{NI-DDPM}}(t) = \boldsymbol{\Sigma} + e^{-\int_0^t \beta(s)\mathrm{d}s}(\mathbf{0} - \boldsymbol{\Sigma}) = \boldsymbol{\Sigma}(\mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s}).$$
$$\therefore \text{NI-DDPM i.e. } p_{0t}^{\text{NIVP}}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}\left(\mathbf{x}(t) \mid \mathbf{x}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, \boldsymbol{\Sigma}(\mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s})\right).$$

6.3 Non-isotropic SMLD (NI-SMLD)

6.3.1 Forward (data to noise) for NI-SMLD

In NI-SMLD, for a fixed sequence of positive scales $0 < \sigma_1 < \cdots \sigma_L < 1$, and a noise sample $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ from a standard normal distribution, and a clean data point $\mathbf{x}_0$, the **cumulative** "**forward**" process is:

$$q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_i \mid \mathbf{x}_0, \sigma_i^2 \boldsymbol{\Sigma}). \tag{6.3.1}$$

$$\implies \mathbf{x}_i = \mathbf{x}_0 + \sigma_i \sqrt{\boldsymbol{\Sigma}} \boldsymbol{\epsilon} \implies \boldsymbol{\epsilon} = \sqrt{\boldsymbol{\Sigma}^{-1}} \frac{\mathbf{x}_i - \mathbf{x}_0}{\sigma_i}. \tag{6.3.2}$$

The **transition** "**forward**" process can be derived as:

$$q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_{i+1} \mid \mathbf{x}_i) = \mathcal{N}(\mathbf{x}_{i+1} \mid \mathbf{x}_i, (\sigma_{i+1}^2 - \sigma_i^2) \boldsymbol{\Sigma}). \tag{6.3.3}$$

$$\implies \mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2} \sqrt{\boldsymbol{\Sigma}} \boldsymbol{\epsilon}_{i-1}. \tag{6.3.4}$$

6.3.2 Score for NI-SMLD

$$q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_i \mid \mathbf{x}_0, \sigma_i^2 \boldsymbol{\Sigma}). \tag{6.3.5}$$

$$\implies \nabla_{\mathbf{x}_i} \log q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i \mid \mathbf{x}_0) = -\boldsymbol{\Sigma}^{-1} \frac{\mathbf{x}_i - \mathbf{x}_0}{\sigma_i^2} = -\sqrt{\boldsymbol{\Sigma}^{-1}} \frac{\boldsymbol{\epsilon}}{\sigma_i}. \tag{6.3.6}$$

6.3.3 Score-matching objective function for NI-SMLD

The objective function for SMLD at noise level σ is:

$$\ell^{\text{NI-SMLD}}(\boldsymbol{\theta}; \sigma_i) \triangleq \frac{1}{2} \mathbb{E}_{q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i|\mathbf{x}_0) p(\mathbf{x}_0)} \left[\left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) + \boldsymbol{\Sigma}^{-1} \frac{\mathbf{x}_i - \mathbf{x}_0}{\sigma_i^2} \right\|_2^2 \right], \tag{6.3.7}$$

$$= \frac{1}{2} \mathbb{E}_{q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i|\mathbf{x}_0) p(\mathbf{x}_0)} \left[\left\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) + \frac{1}{\sigma_i} \sqrt{\boldsymbol{\Sigma}^{-1}} \boldsymbol{\epsilon} \right\|_2^2 \right]. \tag{6.3.8}$$

6.3.4 Variance of score for NI-SMLD

$$\mathbb{E}\left[\left\| \nabla_{\mathbf{x}_i} \log q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i \mid \mathbf{x}_0) \right\|_2^2 \right] = \mathbb{E}\left[\left\| -\boldsymbol{\Sigma}^{-1} \frac{\mathbf{x}_i - \mathbf{x}_0}{\sigma_i^2} \right\|_2^2 \right],$$

$$= \mathbb{E}\left[\left\| \boldsymbol{\Sigma}^{-1} \frac{\sigma_i \sqrt{\boldsymbol{\Sigma}} \boldsymbol{\epsilon}}{\sigma_i^2} \right\|_2^2 \right] = \frac{1}{\sigma_i^2} \boldsymbol{\Sigma}^{-1} \mathbb{E}\left[\|\boldsymbol{\epsilon}\|_2^2 \right] = \frac{1}{\sigma_i^2} \boldsymbol{\Sigma}^{-1}. \tag{6.3.9}$$

6.3.5 Overall objective function for NI-SMLD

The overall objective function weights the score-matching objective $\ell^{\text{NI-SMLD}}(\boldsymbol{\theta}; \sigma_i)$ by the inverse of the variance of the score at each time step $\implies \lambda(\sigma_i) = \sigma_i^2 \boldsymbol{\Sigma}$:

$$\mathcal{L}^{\text{NI-SMLD}}(\boldsymbol{\theta}; \{\bar{\alpha}_t\}_{t=1}^L) \triangleq \mathbb{E}_t \ \lambda(\bar{\alpha}_t) \ \ell^{\text{NI-SMLD}}(\boldsymbol{\theta}; \bar{\alpha}_t),$$

$$= \frac{1}{2L} \sum_{i=1}^{L} \mathbb{E}_{q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i|\mathbf{x}_0)p(\mathbf{x}_0)} \left[\left\| \sigma_i \sqrt{\mathbf{\Sigma}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) + \sqrt{\mathbf{\Sigma}^{-1}} \frac{(\mathbf{x}_i - \mathbf{x}_0)}{\sigma_i} \right\|_2^2 \right],$$

$$= \frac{1}{2L} \sum_{i=1}^{L} \mathbb{E}_{q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i|\mathbf{x}_0)p(\mathbf{x}_0)} \left[\left\| \sigma_i \sqrt{\mathbf{\Sigma}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) + \boldsymbol{\epsilon} \right\|_2^2 \right]. \tag{6.3.10}$$

6.3.6 Unconditional NI-SMLD score estimation

An unconditional score model is:

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) = -\sqrt{\mathbf{\Sigma}^{-1}} \frac{1}{\sigma_i} \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_i). \tag{6.3.11}$$

In this case, the overall objective function changes to:

$$\mathcal{L}^{\text{NI-SMLD}}(\boldsymbol{\theta}; \{\sigma_i\}_{i=1}^{L}) \triangleq \frac{1}{2L} \sum_{i=1}^{L} \mathbb{E}_{q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i|\mathbf{x}_0)p(\mathbf{x}_0)} \left[\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_i) \|_2^2 \right],$$

$$= \frac{1}{2L} \sum_{i=1}^{L} \mathbb{E}_{q_{\sigma_i}^{\text{NI-SMLD}}(\mathbf{x}_i|\mathbf{x}_0)p(\mathbf{x}_0)} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_0 + \sigma_i \sqrt{\mathbf{\Sigma}} \boldsymbol{\epsilon}) \right\|_2^2 \right]. \tag{6.3.12}$$

6.3.7 Sampling i.e. Reverse (noise to data) for NI-SMLD

Similar to SMLD, NI-SMLD does not explicitly define a reverse process. Instead, similar to Song and Ermon [2019, 2020], Jolicoeur-Martineau et al. [2021a], we derive Annealed Langevin Sampling below to transform from noise to data.

Forward : $\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2} \sqrt{\mathbf{\Sigma}} \boldsymbol{\epsilon}_{i-1}$.

Reverse: Annealed Langevin Sampling for NI-SMLD:

$$\mathbf{x}_L^0 \sim \mathcal{N}(\mathbf{0}, \sigma_{\max} \sqrt{\mathbf{\Sigma}}).$$

$$\left.\begin{aligned} &\mathbf{x}_i^0 = \mathbf{x}_{i+1}^M, \\ &\mathbf{x}_i^{m+1} \leftarrow \mathbf{x}_i^m + \alpha_i \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i^m, \sigma_i) + \sqrt{2\alpha_i} \sqrt{\mathbf{\Sigma}} \boldsymbol{\epsilon}_i^{m+1}, m = 1, \cdots, M. \end{aligned}\right\} i = L, \cdots, 1 \tag{6.3.13}$$

$$\alpha_i = \epsilon \sigma_i^2 / \sigma_L^2.$$

Consistent Annealed Sampling [Jolicoeur-Martineau et al., 2021b] for NI-SMLD:

$$\mathbf{x}_{i-1} \leftarrow \mathbf{x}_i + \alpha_i \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma_i) + \beta \sigma_{i-1} \sqrt{\mathbf{\Sigma}} \boldsymbol{\epsilon}_{i-1}, i = L, \cdots, 1. \tag{6.3.14}$$

$$\alpha_i = \epsilon \sigma_t^2 / \sigma_{\min}^2;\ \beta = \sqrt{1 - \gamma^2 (1 - \epsilon / \sigma_{\min}^2)^2};\ \gamma = \sigma_t / \sigma_{t-1}; \sigma_t > \sigma_{t-1}.$$

6.3.8 Expected Denoised Sample (EDS) for NI-SMLD

From Saremi and Hyvarinen [2019], assuming non-isotropic Gaussian noise of covariance $\sigma^2 \mathbf{\Sigma}$, we know that the EDS $\mathbf{x}_0^*(\mathbf{x}_t, \sigma) \triangleq \mathbb{E}_{\mathbf{x}_0 \sim q_\sigma(\mathbf{x}_0|\mathbf{x}_t)}[\mathbf{x}]$ and optimal score $\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_I, \sigma)$ are related as:

$$\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma) = \frac{1}{\sigma^2} \mathbf{\Sigma}^{-1} (\mathbf{x}_0^*(\mathbf{x}_i, \sigma) - \mathbf{x}_t).$$

$$\implies \mathbf{x}_0^*(\mathbf{x}_i, \sigma) = \mathbf{x}_i + \sigma^2 \mathbf{\Sigma}\ \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_i, \sigma) = \mathbf{x}_i - \sigma \sqrt{\mathbf{\Sigma}}\ \boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_i). \tag{6.3.15}$$

6.3.9 SDE formulation : Non-Isotropic Variance Exploding (NIVE) SDE

For NI-SMLD i.e. Non-Isotropic Variance Exploding (NIVE) SDE, the forward equation and transition probability are derived (below) as:

$$\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\,\sqrt{\boldsymbol{\Sigma}}\,\mathrm{d}\mathbf{w}. \tag{6.3.16}$$

$$p_{0t}^{\text{NIVE}}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}\left(\mathbf{x}(t) \mid \mathbf{x}(0), \sigma^2(t)\boldsymbol{\Sigma}\right). \tag{6.3.17}$$

6.3.9.1 Derivations :

Forward process: We know from eq. (6.3.4) that:

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{i-1}.$$

$$\implies \mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \sqrt{(\sigma^2(t+\Delta t) - \sigma^2(t))\Delta t}\,\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}(t),$$

$$\approx \mathbf{x}(t) + \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}\Delta t}\sqrt{\boldsymbol{\Sigma}}\mathbf{w}(t).$$

$$\implies \mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\,\sqrt{\boldsymbol{\Sigma}}\,\mathrm{d}\mathbf{w}. \tag{6.3.18}$$

Thus, eq. (6.3.16) is derived.

Mean and **Covariance** (from eq. 5.50 and eq.5.51 in Särkkä and Solin [2019]) for a random variable $\mathbf{x}$ that changes according to a stochastic process with drift and diffusion coefficients $\mathbf{f}$ and $\mathbf{G}$, change as:

$$\mathrm{d}\mathbf{x} = \mathbf{f}\,\mathrm{d}t + \mathbf{G}\,\mathrm{d}\mathbf{w} \implies \frac{\mathrm{d}\boldsymbol{\mu}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}], \tag{6.3.19}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{cov}}}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{f}(\mathbf{x}-\boldsymbol{\mu})^T] + \mathbb{E}_{\mathbf{x}}[(\mathbf{x}-\boldsymbol{\mu})\mathbf{f}^T] + \mathbb{E}_{\mathbf{x}}[\mathbf{G}\mathbf{G}^T]. \tag{6.3.20}$$

For NI-SMLD, $\mathbf{f} = \mathbf{0}, \boldsymbol{\mu} = \mathbf{0}, \mathbf{G} = \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\sqrt{\boldsymbol{\Sigma}}$.

$$\frac{\mathrm{d}\boldsymbol{\mu}_{\text{NI-SMLD}}(t)}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}[\mathbf{0}] = \mathbf{0}, \tag{6.3.21}$$

$$\implies \boldsymbol{\mu}_{\text{NI-SMLD}}(t) = \boldsymbol{\mu}(0) = \mathbf{x}(0). \tag{6.3.22}$$

$$\frac{\mathrm{d}\boldsymbol{\Sigma}_{\text{NI-SMLD}}(t)}{\mathrm{d}t} = \mathbb{E}_{\mathbf{x}}\left[\mathbf{0} + \mathbf{0} + \sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\sqrt{\boldsymbol{\Sigma}}\sqrt{\frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}}\sqrt{\boldsymbol{\Sigma}}\right] = \frac{\mathrm{d}[\sigma^2(t)]}{\mathrm{d}t}\boldsymbol{\Sigma}, \tag{6.3.23}$$

$$\implies \boldsymbol{\Sigma}_{\text{NI-SMLD}}(t) = \sigma^2(t)\boldsymbol{\Sigma}. \tag{6.3.24}$$

$\therefore$ NI-SMLD i.e. $p_{0t}^{\text{NIVE}}(\mathbf{x}(t) \mid \mathbf{x}(0)) = \mathcal{N}(\mathbf{x}(t) \mid \mathbf{x}(0), \sigma^2(t)\boldsymbol{\Sigma})$. Thus, eq. (6.3.17) is derived.

6.3.10 Initial noise scale for NI-SMLD

Building on Proposition 1 in Song and Ermon [2020], let $\hat{p}_{\sigma_1}(\mathbf{x}) \triangleq \frac{1}{N}\sum_{i=1}^{N} p^{(i)}(\mathbf{x})$, where $p^{(i)}(\mathbf{x}) \triangleq \mathcal{N}(\mathbf{x} \mid \mathbf{x}^{(i)}, \sigma_1^2\mathbf{\Sigma})$. With $r^{(i)}(\mathbf{x}) \triangleq \frac{p^{(i)}(\mathbf{x})}{\sum_{k=1}^{N} p^{(k)}(\mathbf{x})}$, the score function is $\nabla_{\mathbf{x}} \log \hat{p}_{\sigma_1}(\mathbf{x}) = \sum_{i=1}^{N} r^{(i)}(\mathbf{x}) \nabla_{\mathbf{x}} \log p^{(i)}(\mathbf{x})$.

We know that:

$$\mathcal{N}(\mathbf{x} \mid \mathbf{x}^{(i)}, \sigma_1^2\mathbf{\Sigma}) = \frac{1}{(2\pi)^{D/2}\sigma_1^D|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2\sigma_1^2}(\mathbf{x}-\mathbf{x}^{(i)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(i)})\right).$$

$$\begin{aligned}
\mathbb{E}_{p^{(i)}(\mathbf{x})}[r^{(j)}(\mathbf{x})] &= \int \frac{p^{(i)}(\mathbf{x})p^{(j)}(\mathbf{x})}{\sum_{k=1}^{N} p^{(k)}(\mathbf{x})}\mathrm{d}\mathbf{x} \leq \int \frac{p^{(i)}(\mathbf{x})p^{(j)}(\mathbf{x})}{p^{(i)}(\mathbf{x}) + p^{(j)}(\mathbf{x})}\mathrm{d}\mathbf{x}, \\
&= \frac{1}{2}\int \frac{2}{\frac{1}{p^{(i)}(\mathbf{x})} + \frac{1}{p^{(j)}(\mathbf{x})}}\mathrm{d}\mathbf{x} \leq \frac{1}{2}\int \sqrt{p^{(i)}(\mathbf{x})p^{(j)}(\mathbf{x})}\mathrm{d}\mathbf{x}, \\
&= \frac{1}{2}\frac{1}{(2\pi)^{D/2}\sigma_1^D|\mathbf{\Sigma}|^{1/2}} \int \exp\Bigg(-\frac{1}{4\sigma_1^2}\Big((\mathbf{x}-\mathbf{x}^{(i)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(i)}) + (\mathbf{x}-\mathbf{x}^{(j)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(j)})\Big)\mathrm{d}\mathbf{x}, \\
&= \frac{1}{2}\frac{1}{(2\pi)^{D/2}\sigma_1^D|\mathbf{\Sigma}|^{1/2}} \int \exp\Bigg(-\frac{1}{4\sigma_1^2}\Big((\mathbf{x}-\mathbf{x}^{(i)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(i)}) + (\mathbf{x}-\mathbf{x}^{(j)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(i)}), \\
&\qquad + (\mathbf{x}-\mathbf{x}^{(j)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})\Big)\mathrm{d}\mathbf{x}, \\
&= \frac{1}{2}\frac{1}{(2\pi)^{D/2}\sigma_1^D|\mathbf{\Sigma}|^{1/2}} \int \exp\Bigg(-\frac{1}{4\sigma_1^2}\Big((\mathbf{x}-\mathbf{x}^{(i)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(i)}) + (\mathbf{x}-\mathbf{x}^{(i)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(i)}), \\
&\qquad + (\mathbf{x}^{(i)}-\mathbf{x}^{(j)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(i)}) + (\mathbf{x}-\mathbf{x}^{(i)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)}), \\
&\qquad + (\mathbf{x}^{(i)}-\mathbf{x}^{(j)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})\Big)\mathrm{d}\mathbf{x}, \\
&= \frac{1}{2}\frac{1}{(2\pi)^{D/2}\sigma_1^D|\mathbf{\Sigma}|^{1/2}} \int \exp\Bigg(-\frac{1}{2\sigma_1^2}\Big((\mathbf{x}-\mathbf{x}^{(i)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}^{(i)}) + 2(\mathbf{x}-\mathbf{x}^{(i)})^T\mathbf{\Sigma}^{-1}\frac{(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})}{2}, \\
&\qquad + \frac{(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})}{2}^T\mathbf{\Sigma}^{-1}\frac{(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})}{2} - \frac{(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})}{2}^T\mathbf{\Sigma}^{-1}\frac{(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})}{2}, \\
&\qquad + \frac{1}{2}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})\Big)\mathrm{d}\mathbf{x}, \\
&= \frac{1}{2}\exp\Bigg(-\frac{1}{8\sigma_1^2}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})\Bigg).
\end{aligned}$$

It is desirable to choose σ_1 that is proportional to the numerator term so that $\mathbb{E}_{p^{(i)}(\mathbf{x})}[r^{(j)}(\mathbf{x})]$ has a reasonably large value. To make this happen, from Song and Ermon [2020]:

$$\begin{aligned}
&\implies \frac{1}{\sigma_1^2}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})^T\mathbf{\Sigma}^{-1}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)}) \approx 1, \\
&\implies (\sqrt{\mathbf{\Sigma}^{-1}}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)}))^T(\sqrt{\mathbf{\Sigma}^{-1}}(\mathbf{x}^{(i)}-\mathbf{x}^{(j)})) \approx \sigma_1^2,
\end{aligned}$$

$$
\begin{aligned}
&\implies \left\|\sqrt{\mathbf{\Sigma}^{-1}}(\mathbf{x}^{(i)} - \mathbf{x}^{(j)})\right\|_2 \approx \sigma_1, \\
&\implies \left\|\sigma_N \text{ Real}\Big(\mathbf{W}_N^{-1}\mathbf{K}\mathbf{W}_N(\mathbf{x}^{(i)} - \mathbf{x}^{(j)})\Big)\right\|_2 \approx \sigma_1, \\
&\implies \left\|\sigma_N \mathbf{W}_N^{-1}\mathbf{K}\mathbf{W}_N\mathbf{x}^{(i)} - \sigma_N \mathbf{W}_N^{-1}\mathbf{K}\mathbf{W}_N\mathbf{x}^{(j)}\right\|_2 \approx \sigma_1.
\end{aligned}
$$

For CIFAR10, this $\sigma_1 \approx 20$ for NI-SMLD (whereas for SMLD $\sigma_1 \approx 50$).

6.3.11 Other noise scales

Building on Proposition 2 in Song and Ermon [2020], let image $\mathbf{x} \in \mathbb{R}^D \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, and $r = ||\mathbf{x}||_2$. For simplification of analysis, because image dimensions D are typically quite large, we can assume:

$$p_{\sigma_i}(r) = \mathcal{N}(r \mid m_i, s_i^2), \text{where } m_i \triangleq \sqrt{D}\sigma_i; s_i^2 \triangleq \sigma_i^2/2.$$

Using the three-sigma rule of thumb [Grafarend, 2006], $p_{\sigma_i}(r)$ has high density in:

$$\mathcal{I}_i \triangleq (m_i - 3s_i, m_i + 3s_i).$$

Given the discrete nature of σ_is, we need the radial components of $p_{\sigma_i}(\mathbf{x})$ and $p_{\sigma_{i-1}}(\mathbf{x})$ to have large overlap. This naturally leads us to fix $p_{\sigma_i}(r \in \mathcal{I}_{i-1})$ to be a moderately high constant, Song and Ermon [2020] chose 0.5. Given $\Phi(.)$ is the Cumulative Density Function (CDF) of standard Gaussian, and $\gamma \triangleq \sigma_{i-1}/\sigma_i$ considering a geometric progression of σ_is:

$$
\begin{aligned}
p_{\sigma_i}(r \in \mathcal{I}_{i-1}) &= \Phi\left(\frac{(m_{i-1} + 3s_{i-1}) - m_i}{s_i}\right) - \Phi\left(\frac{(m_{i-1} - 3s_{i-1}) - m_i}{s_i}\right), \\
&= \Phi\left(\frac{\sqrt{2}}{\sigma_i}(\sqrt{D}\sigma_{i-1} + \frac{3\sigma_{i-1}}{\sqrt{2}} - \sqrt{D}\sigma_i)\right) - \Phi\left(\frac{\sqrt{2}}{\sigma_i}(\sqrt{D}\sigma_{i-1} - \frac{3\sigma_{i-1}}{\sqrt{2}} - \sqrt{D}\sigma_i)\right), \\
&= \Phi\left(\frac{1}{\sigma_i}(\sqrt{2D}(\sigma_{i-1} - \sigma_i) + 3\sigma_{i-1})\right) - \Phi\left(\frac{1}{\sigma_i}(\sqrt{2D}(\sigma_{i-1} - \sigma_i) - 3\sigma_{i-1})\right), \\
&= \Phi\left(\sqrt{2D}(\gamma - 1) + 3\gamma\right) - \Phi\left(\sqrt{2D}(\gamma - 1) - 3\gamma\right) \approx 0.5.
\end{aligned}
$$

From the previous discussion, given $\sigma_1 = 20$, and setting $\sigma_L = 0.01$, $L = 207$, $\gamma = 1.0376$ (whereas for SMLD $\sigma_1 = 50, L = 232$).

6.3.12 Configuring annealed Langevin dynamics

Building on Proposition 3 in Song and Ermon [2020], $\gamma = \frac{\sigma_{i-1}}{\sigma_i}$. For $\alpha = \epsilon \cdot \frac{\sigma_i^2}{\sigma_L^2}$, we have $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \text{Var}[\mathbf{x}_T])$, where

$$\frac{\text{Var}[\mathbf{x}_T]}{\sigma_i^2} = \gamma^2\mathbf{P}^T\mathbf{\Sigma}\mathbf{P}^T + \frac{2\epsilon}{\sigma_L^2}\sum_{t=0}^{T-1}(\mathbf{P}^t\mathbf{\Sigma}\mathbf{P}^t), \tag{6.3.25}$$

where $\mathbf{P} = \mathbf{I} - \frac{\alpha}{\sigma_i^2}\mathbf{\Sigma}^{-1} = \mathbf{I} - \frac{\epsilon}{\sigma_L^2}\mathbf{\Sigma}^{-1}$ (proof below).

Hence, we choose ϵ s.t. $\frac{Var[\mathbf{x}_T]}{\sigma_i^2} \approx 1, \implies \epsilon = 3.1\text{e}{-7}$ for $T = 5$, $\epsilon = 2.0\text{e}{-6}$ for $T = 1$ (whereas for SMLD $\epsilon = 6.2\text{e}{-6}$ for $T = 5$)

Proof:

First, the conditions we know are:

$$\mathbf{x}_0 \sim p_{\sigma_{i-1}}(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \mathbf{0}, \sigma_{i-1}^2 \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}\sigma_{i-1}^D |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2\sigma_{i-1}^2}\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x}\right),$$

$$\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x}_t) = \nabla_{\mathbf{x}}\left(-\log(\text{const.}) - \frac{1}{2\sigma_i^2}\mathbf{x}_t^T\boldsymbol{\Sigma}^{-1}\mathbf{x}_t\right) = -\frac{1}{\sigma_i^2}\boldsymbol{\Sigma}^{-1}\mathbf{x}_t,$$

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \alpha\nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x}_t) + \sqrt{2\alpha}\mathbf{g}_t = \mathbf{x}_t - \alpha\frac{1}{\sigma_i^2}\boldsymbol{\Sigma}^{-1}\mathbf{x}_t + \sqrt{2\alpha}\mathbf{g}_t,$$

where $\mathbf{g}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, $\alpha = \epsilon\frac{\sigma_i^2}{\sigma_L^2}$. Therefore, the variance of $\mathbf{x}_t$ satisfies

$$\text{Var}[\mathbf{x}_t] = \begin{cases} \sigma_{i-1}^2\boldsymbol{\Sigma} & \text{if } t = 0 \\ \text{Var}[\big(\mathbf{I} - \frac{\alpha}{\sigma_i^2}\boldsymbol{\Sigma}^{-1}\big)\mathbf{x}_{t-1}] + 2\alpha\boldsymbol{\Sigma} & \text{otherwise.} \end{cases}$$

$$\text{Var}[\mathbf{A}\mathbf{x}] = \mathbf{A}\text{Var}[\mathbf{x}]\mathbf{A}^T \implies \text{Var}[\mathbf{x}_t] = \Big(\mathbf{I} - \frac{\alpha}{\sigma_i^2}\boldsymbol{\Sigma}^{-1}\Big)\text{Var}[\mathbf{x}_{t-1}]\Big(\mathbf{I} - \frac{\alpha}{\sigma_i^2}\boldsymbol{\Sigma}^{-1}\Big) + 2\alpha\boldsymbol{\Sigma}.$$

Let $\mathbf{P} = \mathbf{I} - \frac{\alpha}{\sigma_i^2}\boldsymbol{\Sigma}^{-1} = \mathbf{I} - \frac{\epsilon}{\sigma_L^2}\boldsymbol{\Sigma}^{-1}$.

$$\begin{aligned} \implies \text{Var}[\mathbf{x}_t] &= \mathbf{P}\text{Var}[\mathbf{x}_{t-1}]\mathbf{P} + 2\alpha\boldsymbol{\Sigma} = \mathbf{P}(\mathbf{P}\text{Var}[\mathbf{x}_{t-2}]\mathbf{P} + 2\alpha\boldsymbol{\Sigma})\mathbf{P} + 2\alpha\boldsymbol{\Sigma} \\ &= \mathbf{P}\mathbf{P}\text{Var}[\mathbf{x}_{t-2}]\mathbf{P}\mathbf{P} + 2\alpha(\mathbf{P}\boldsymbol{\Sigma}\mathbf{P} + \boldsymbol{\Sigma}) = \mathbf{P}^{(2)}\text{Var}[\mathbf{x}_{t-2}]\mathbf{P}^{(2)} + 2\alpha(\mathbf{P}\boldsymbol{\Sigma}\mathbf{P} + \boldsymbol{\Sigma}), \end{aligned}$$

$$\implies \text{Var}[\mathbf{x}_T] = \mathbf{P}^{(T)}\text{Var}[\mathbf{x}_0]\mathbf{P}^{(T)} + 2\alpha\sum_{t=0}^{T-1}(\mathbf{P}^{(t)}\boldsymbol{\Sigma}\mathbf{P}^{(t)}) = \sigma_{i-1}^2\mathbf{P}^{(T)}\boldsymbol{\Sigma}\mathbf{P}^{(T)} + 2\epsilon\frac{\sigma_i^2}{\sigma_L^2}\sum_{t=0}^{T-1}(\mathbf{P}^{(t)}\boldsymbol{\Sigma}\mathbf{P}^{(t)}),$$

$$\implies \frac{\text{Var}[\mathbf{x}_T]}{\sigma_i^2} = \gamma^2\mathbf{P}^{(T)}\boldsymbol{\Sigma}\mathbf{P}^{(T)} + \frac{2\epsilon}{\sigma_L^2}\sum_{t=0}^{T-1}(\mathbf{P}^{(t)}\boldsymbol{\Sigma}\mathbf{P}^{(t)}).$$

6.3.13 Optimal conditional score function

Jolicoeur-Martineau et al. [2021b] discovered in Appendix E that for SMLD, the conditional score estimate in expectation does not match the theoretic value when derived from the unconditional score model in the case of a single data point $\mathbf{x}_0$:

$$\text{unconditional } \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i) = \frac{1}{L}\sum_{i=1}^{L}\left(\frac{\mathbb{E}_{\mathbf{x}_0\sim q_{\sigma_i}(\mathbf{x}_0|\mathbf{x}_i)}[\mathbf{x}_0] - \mathbf{x}_i}{\sigma_i}\right) = \frac{\mathbf{x}_0 - \mathbf{x}_i}{\sigma_H}, \tag{6.3.26}$$

$$\implies \text{conditional } \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) = \frac{1}{\sigma_i}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i) = \frac{\mathbf{x}_0 - \mathbf{x}_i}{\sigma_i\sigma_H} \neq \frac{\mathbf{x}_0 - \mathbf{x}_i}{\sigma_i^2}, \tag{6.3.27}$$

where $\frac{1}{\sigma_H} = \frac{1}{L}\sum_{i=1}^{L}\frac{1}{\sigma_i}$, i.e. σ_H is the harmonic mean of the σ_is used to train.

In the case of NI-SMLD, using calculus of variations,

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{s}} &= \int\!\!\int q_\sigma(\mathbf{x}_i, \mathbf{x}, \sigma)\left(\mathbf{s}(\mathbf{x}_i) + \sqrt{\mathbf{\Sigma}^{-1}}\frac{\mathbf{x}_i - \mathbf{x}}{\sigma}\right)\mathrm{d}\mathbf{x}\mathrm{d}\sigma = 0,\\
\iff \mathbf{s}(\mathbf{x}_i)q(\mathbf{x}_i) &= \sqrt{\mathbf{\Sigma}^{-1}}\int\!\!\int q_\sigma(\mathbf{x}_i, \mathbf{x})p(\sigma)\left(\frac{\mathbf{x}_i - \mathbf{x}}{\sigma}\right)\mathrm{d}\mathbf{x}\mathrm{d}\sigma,\\
\iff \mathbf{s}(\mathbf{x}_i)q(\mathbf{x}_i) &= \sqrt{\mathbf{\Sigma}^{-1}}\mathbb{E}_{\sigma\sim p(\sigma)}\left[\int q_\sigma(\mathbf{x}_i \mid \mathbf{x})q(\mathbf{x}_i)\left(\frac{\mathbf{x}_i - \mathbf{x}}{\sigma}\right)\mathrm{d}\mathbf{x}\right],\\
\iff \mathbf{s}(\mathbf{x}_i) &= \sqrt{\mathbf{\Sigma}^{-1}}\mathbb{E}_{\sigma\sim p(\sigma)}\left[\int q_\sigma(\mathbf{x}_i \mid \mathbf{x})\left(\frac{\mathbf{x}_i - \mathbf{x}}{\sigma}\right)\mathrm{d}\mathbf{x}\right],\\
\iff \text{unconditional } \mathbf{s}(\mathbf{x}_i) &= \sqrt{\mathbf{\Sigma}^{-1}}\mathbb{E}_{\sigma\sim p(\sigma)}\left[\frac{\mathbb{E}_{\mathbf{x}\sim q_\sigma(\mathbf{x}|\mathbf{x}_i)}[\mathbf{x}] - \mathbf{x}}{\sigma}\right].
\end{aligned}$$

For SMLD, in the case of a single data point $\mathbf{x}_0$:

$$\begin{aligned}
\text{unconditional } \mathbf{s}(\mathbf{x}_i) &= \sqrt{\mathbf{\Sigma}^{-1}}\frac{\mathbf{x}_0 - \mathbf{x}_i}{\sigma_H},\\
\implies \text{conditional } \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) &= \frac{1}{\sigma_i}\sqrt{\mathbf{\Sigma}^{-1}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i) = \mathbf{\Sigma}^{-1}\frac{\mathbf{x}_0 - \mathbf{x}_i}{\sigma_i\sigma_H} \neq \mathbf{\Sigma}^{-1}\frac{\mathbf{x}_0 - \mathbf{x}_i}{\sigma_i^2}. \qquad (6.3.28)
\end{aligned}$$

Hence, even for NI-SMLD, the discrepancy between the theoretical value of the score, and the estimated conditional score derived from the unconditional score persists. We correct for this discrepancy while sampling:

$$\text{conditional } \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i, \sigma_i) = \frac{\sigma_H}{\sigma_i}\left[\frac{1}{\sigma_i}\sqrt{\mathbf{\Sigma}^{-1}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i)\right] = \frac{\sigma_H}{\sigma_i^2}\sqrt{\mathbf{\Sigma}^{-1}}\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_i). \qquad (6.3.29)$$

6.4 Isotropic DDPM v/s Non-isotropic DDPM

Below is an essential summary of isotropic and non-isotropic Gaussian denoising diffusion models for a more direct comparison.

6.4.1 Isotropic Gaussian denoising diffusion models (DDPM)

We perform our analysis below within the Denoising Diffusion Probabilistic Models (DDPM) [Ho et al., 2020] framework, but our analysis is valid for all other types of score-based denoising diffusion models.

In DDPM, for a fixed sequence of positive scales $0 < \beta_1 < \cdots < \beta_L < 1$, $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$, and a noise sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the cumulative "**forward**" noising process is:

$$q_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \implies \mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}. \qquad (6.4.1)$$

The "**reverse**" process involves iteratively **sampling** $\mathbf{x}_{t-1}$ from $\mathbf{x}_t$ conditioned on $\mathbf{x}_0$ i.e. $p_{t-1}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$, obtained from $q_t(\mathbf{x}_t \mid \mathbf{x}_0)$ using Bayes' rule. For this, first $\boldsymbol{\epsilon}$ is estimated using a neural network $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$. Then, using $\hat{\mathbf{x}}_0 = \left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right)/\sqrt{\bar{\alpha}_t}$ from

	Isotropic	Non-Isotropic
Forward process	$q_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar\alpha_t}\mathbf{x}_0, (1-\bar\alpha_t)\mathbf{I})$ $\Rightarrow \mathbf{x}_t = \sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\boldsymbol{\epsilon}$	$q_t(\enclose{circle}{\mathbf{x}_t} \mid \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar\alpha_t}\mathbf{x}_0, (1-\bar\alpha_t)\boxed{\boldsymbol{\Sigma}})$ $\Rightarrow \enclose{circle}{\mathbf{x}_t} = \sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\boxed{\sqrt{\boldsymbol{\Sigma}}}\boldsymbol{\epsilon}$
Score $\mathbf{s} = \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t \mid \mathbf{x}_0)$	$\mathbf{s} = -\frac{1}{\sqrt{1-\bar\alpha_t}}\boldsymbol{\epsilon}$	$\mathbf{s} = -\frac{1}{\sqrt{1-\bar\alpha_t}}\boxed{\sqrt{\boldsymbol{\Sigma}^{-1}}}\boldsymbol{\epsilon}$
Score-matching loss $\ell(\boldsymbol{\theta}; t) = \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}} \Vert \mathbf{s} - \underline{\mathbf{s}_{\boldsymbol{\theta}}(\enclose{circle}{\mathbf{x}_t}, t)} \Vert_2^2$	$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_t\ (1-\bar\alpha_t)\,\ell(\boldsymbol{\theta}; t)$	$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_t\ (1-\bar\alpha_t)\boxed{\boldsymbol{\Sigma}}\,\ell(\boldsymbol{\theta}; t)$
Noise-matching loss	$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \Vert \boldsymbol{\epsilon} - \underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)} \Vert_2^2$	$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \Vert \boldsymbol{\epsilon} - \underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\enclose{circle}{\mathbf{x}_t}, t)} \Vert_2^2$
Sampling w/ DDPM	1) $\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar\alpha_t}}(\mathbf{x}_t - \sqrt{1-\bar\alpha_t}\,\underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)})$ 2) $\mathbf{x}_{t-1} = \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0) + \sqrt{\tilde\beta_t}\mathbf{z}_t$	1) $\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar\alpha_t}}(\enclose{circle}{\mathbf{x}_t} - \sqrt{1-\bar\alpha_t}\boxed{\sqrt{\boldsymbol{\Sigma}}}\underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\enclose{circle}{\mathbf{x}_t}, t)})$ 2) $\mathbf{x}_{t-1} = \tilde{\boldsymbol{\mu}}_t(\enclose{circle}{\mathbf{x}_t}, \hat{\mathbf{x}}_0) + \sqrt{\tilde\beta_t}\boxed{\sqrt{\boldsymbol{\Sigma}}}\mathbf{z}_t$
Sampling w/ DDIM	1) $\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar\alpha_t}}(\mathbf{x}_t - \sqrt{1-\bar\alpha_t}\,\underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)})$ 2) $\mathbf{x}_{t-1} = \sqrt{\bar\alpha_{t-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar\alpha_{t-1}}\,\underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}$	1) $\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar\alpha_t}}(\enclose{circle}{\mathbf{x}_t} - \sqrt{1-\bar\alpha_t}\boxed{\sqrt{\boldsymbol{\Sigma}}}\underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\enclose{circle}{\mathbf{x}_t}, t)})$ 2) $\mathbf{x}_{t-1} = \sqrt{\bar\alpha_{t-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar\alpha_{t-1}}\boxed{\sqrt{\boldsymbol{\Sigma}}}\underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\enclose{circle}{\mathbf{x}_t}, t)}$
Expected Denoised Sample $\mathbf{x}_0^*(\mathbf{x}_t, t) = \mathbb{E}_{\mathbf{x}_0 \sim p_t(\mathbf{x}_0 \mid \mathbf{x}_t)}[\mathbf{x}_0]$	$\mathbf{x}_0^*(\mathbf{x}_t, t) = \mathbf{x}_t - \sqrt{1-\bar\alpha_t}\,\underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, t)}$	$\mathbf{x}_0^*(\mathbf{x}_t, t) = \enclose{circle}{\mathbf{x}_t} - \sqrt{1-\bar\alpha_t}\boxed{\sqrt{\boldsymbol{\Sigma}}}\underline{\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\enclose{circle}{\mathbf{x}_t}, t)}$
SDE formulation	$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\mathbf{w}$ $p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0)) =$ $\mathcal{N}\left(\mathbf{x}(0)\, e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, (\mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s})\right)$	$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\mathrm{d}t + \sqrt{\beta(t)}\boxed{\sqrt{\boldsymbol{\Sigma}}}\mathrm{d}\mathbf{w}$ $p_{0t}(\enclose{circle}{\mathbf{x}(t)} \mid \mathbf{x}(0)) =$ $\mathcal{N}\left(\mathbf{x}(0)\, e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s}, \boxed{\boldsymbol{\Sigma}}(\mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s})\right)$

Fig. 6.2. Isotropic v/s Non-Isotropic Denoising Diffusion Probabilistic Models

eq. (6.4.1), $\mathbf{x}_{t-1}$ is sampled:

$$p_{t-1}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \hat{\mathbf{x}}_0) = \mathcal{N}(\ \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0), \tilde\beta_t\mathbf{I}\ ) \implies \mathbf{x}_{t-1} = \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0) + \sqrt{\tilde\beta_t}\mathbf{z}_t \quad \text{, where} \tag{6.4.2}$$

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0) = \frac{\sqrt{\bar\alpha_{t-1}}\beta_t}{1-\bar\alpha_t}\hat{\mathbf{x}}_0 + \frac{\sqrt{1-\beta_t}(1-\bar\alpha_{t-1})}{1-\bar\alpha_t}\mathbf{x}_t \ ; \ \tilde\beta_t = \frac{1-\bar\alpha_{t-1}}{1-\bar\alpha_t}\beta_t \ ; \ \mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \ . \tag{6.4.3}$$

The **objective** to train $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ is simply an expected reconstruction loss with the true $\boldsymbol{\epsilon}$:

$$\mathcal{L}_{\boldsymbol{\epsilon}}(\boldsymbol{\theta}) = \mathbb{E}_{t \sim \mathcal{U}(1,\cdots,L), \mathbf{x}_0 \sim p(\mathbf{x}_0), \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})} \left[\left\Vert \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left(\sqrt{\bar\alpha_t}\mathbf{x}_0 + \sqrt{1-\bar\alpha_t}\boldsymbol{\epsilon}, t\right) \right\Vert_2^2 \right] . \tag{6.4.4}$$

From the perspective of score matching, the **score** of the DDPM forward process is:

$$\text{Score} \quad \mathbf{s} = \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t \mid \mathbf{x}_0) = -\frac{1}{(1-\bar\alpha_t)}(\mathbf{x}_t - \sqrt{\bar\alpha_t}\mathbf{x}_0) = -\frac{1}{\sqrt{1-\bar\alpha_t}}\boldsymbol{\epsilon}. \tag{6.4.5}$$

Thus, the overall **score-matching objective** for a score estimation network $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ is the weighted sum of the loss $\ell_{\mathbf{s}}(\boldsymbol{\theta}; t)$ for each t, the weight being the inverse of the score **variance** at t i.e. $(1 - \bar{\alpha}_t)$:

$$\mathcal{L}_{\mathbf{s}}(\boldsymbol{\theta}) = \mathbb{E}_t \ (1 - \bar{\alpha}_t) \ \ell_{\mathbf{s}}(\boldsymbol{\theta}; t) = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\left\| \sqrt{1 - \bar{\alpha}_t} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \boldsymbol{\epsilon} \right\|_2^2 \right]. \tag{6.4.6}$$

When the score network output is redefined as per the **score-noise relationship** in eq. (6.4.5):

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = -\frac{1}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_{\theta}(\mathbf{x}_t, t) \implies \mathcal{L}_{\mathbf{s}}(\boldsymbol{\theta}) = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\left\| -\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \boldsymbol{\epsilon} \right\|_2^2 \right] = \mathcal{L}_{\boldsymbol{\epsilon}}(\boldsymbol{\theta}) \tag{6.4.7}$$

Thus, $\mathcal{L}_{\mathbf{s}} = \mathcal{L}_{\boldsymbol{\epsilon}}$ i.e. the score-matching and noise reconstruction objectives are equivalent.

$$\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, t) = \mathbb{E}\left[\| \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t \mid \mathbf{x}_0) \|_2^2 \right] \left(\mathbf{x}_0^*(\mathbf{x}_t, t) - \mathbf{x}_t \right) = \frac{1}{1 - \bar{\alpha}_t} \left(\mathbf{x}_0^*(\mathbf{x}_t, t) - \mathbf{x}_t \right), \tag{6.4.8}$$

$$\implies \mathbf{x}_0^*(\mathbf{x}_t, t) = \mathbf{x}_t + (1 - \bar{\alpha}_t) \ \mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, t) = \mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \ \boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, t). \tag{6.4.9}$$

The EDS is often used to further improve the quality of the final image at $t = 0$.

6.4.2 Non-isotropic Gaussian denoising diffusion models (NI-DDPM)

We formulate the Non-Isotropic DDPM (**NI-DDPM**) using a non-isotropic Gaussian noise distribution with a positive semi-definite covariance matrix $\boldsymbol{\Sigma}$ in the place of $\mathbf{I}$.

The **forward** noising process is:

$$q_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\boldsymbol{\Sigma}) \implies \mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}. \tag{6.4.10}$$

Thus, the **score** of NI-DDPM is (see Section 6.2.2 for derivation):

$$\text{Score } \ \mathbf{s} = \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t \mid \mathbf{x}_0) = -\boldsymbol{\Sigma}^{-1} \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0}{1 - \bar{\alpha}_t} = -\frac{1}{\sqrt{1 - \bar{\alpha}_t}} \sqrt{\boldsymbol{\Sigma}^{-1}} \boldsymbol{\epsilon}. \tag{6.4.11}$$

The **score-matching objective** for a score estimation network $\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)$ at each noise level t is now:

$$\ell(\boldsymbol{\theta}; t) = \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0), \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\left\| \mathbf{s}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, t) + \frac{1}{\sqrt{1 - \bar{\alpha}_t}} \sqrt{\boldsymbol{\Sigma}^{-1}} \boldsymbol{\epsilon} \right\|_2^2 \right]. \tag{6.4.12}$$

The **variance** of this score is:

$$\mathbb{E}\left[\| \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t \mid \mathbf{x}_0) \|_2^2 \right] = \mathbb{E}\left[\left\| -\frac{1}{\sqrt{1 - \bar{\alpha}_t}} \sqrt{\boldsymbol{\Sigma}^{-1}} \boldsymbol{\epsilon} \right\|_2^2 \right] = \frac{1}{1 - \bar{\alpha}_t} \boldsymbol{\Sigma}^{-1} \mathbb{E}\left[\|\boldsymbol{\epsilon}\|_2^2 \right]. \tag{6.4.13}$$

The **overall objective** is a weighted sum, the weight being the inverse of the score variance $(1 - \bar{\alpha}_t)\boldsymbol{\Sigma}$:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{t \sim \mathcal{U}(1, \cdots, L)} \ (1 - \bar{\alpha}_t)\boldsymbol{\Sigma} \ \ell(\boldsymbol{\theta}; t) = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\left\| \sqrt{1 - \bar{\alpha}_t} \sqrt{\boldsymbol{\Sigma}} \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) + \boldsymbol{\epsilon} \right\|_2^2 \right]. \tag{6.4.14}$$

Following the **score-noise relationship** in eq. (6.4.11):

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\sqrt{\boldsymbol{\Sigma}^{-1}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t). \tag{6.4.15}$$

The **objective function** now becomes (expanding $\mathbf{s}_{\boldsymbol{\theta}}$ as per eq. (6.4.15)):

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{t\sim\mathcal{U}(1,\cdots,L),\mathbf{x}_0\sim p(\mathbf{x}_0),\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\left\|-\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}, t) + \boldsymbol{\epsilon}\right\|_2^2\right]. \tag{6.4.16}$$

This objective function for NI-DDPM seems like $\mathcal{L}_\epsilon$ of DDPM, but DDPM's $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ network cannot be re-used here since their forward processes are different. DDPM produces $\mathbf{x}_t$ from $\mathbf{x}_0$ using eq. (6.4.1), while NI-DDPM uses eq. (6.4.10). See Section 6.2.5 for alternate formulations of the score network.

Sampling involves computing $p_{t-1}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \hat{\mathbf{x}}_0)$ (see Section 6.2.8 for derivation):

$$q_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\boldsymbol{\Sigma}) \implies \hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\Big(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\Big). \tag{6.4.17}$$

$$p_{t-1}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \hat{\mathbf{x}}_0) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0), \tilde{\beta}_t\boldsymbol{\Sigma}) \implies \mathbf{x}_{t-1} = \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0) + \sqrt{\tilde{\beta}_t}\sqrt{\boldsymbol{\Sigma}}\mathbf{z}_t. \tag{6.4.18}$$

where $\tilde{\boldsymbol{\mu}}_t$, $\tilde{\beta}_t$ and $\mathbf{z}_t$ are the same as eq. (6.4.3).

Alternatively, Song et al. [2021b] mentions using β_t instead of $\tilde{\beta}_t$:

$$p_{t-1}^{\beta_t}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \hat{\mathbf{x}}_0) = \mathcal{N}(\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0), \beta_t\boldsymbol{\Sigma}) \implies \mathbf{x}_{t-1} = \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0) + \sqrt{\beta_t}\sqrt{\boldsymbol{\Sigma}}\mathbf{z}_t. \tag{6.4.19}$$

Alternatively, sampling using **DDIM** Song et al. [2021a] invokes the following distribution for $\mathbf{x}_{t-1}$:

$$p_{t-1}^{\text{DDIM}}(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \hat{\mathbf{x}}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_{t-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\hat{\mathbf{x}}_0}{\sqrt{1-\bar{\alpha}_t}},\ \ \mathbf{0}\right). \tag{6.4.20}$$

$$\implies \mathbf{x}_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t-1}}\sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t). \tag{6.4.21}$$

The **Expected Denoised Sample** $\mathbf{x}_0^*(\mathbf{x}_t, t)$ and the optimal score $\mathbf{s}_{\boldsymbol{\theta}^*}$ are now related as:

$$\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, t) = \mathbb{E}\left[\|\nabla_{\mathbf{x}_t}\log q_t(\mathbf{x}_t \mid \mathbf{x}_0)\|_2^2\right]\Big(\mathbf{x}_0^*(\mathbf{x}_t, t) - \mathbf{x}_t\Big) = \frac{1}{1-\bar{\alpha}_t}\boldsymbol{\Sigma}^{-1}\Big(\mathbf{x}_0^*(\mathbf{x}_t, t) - \mathbf{x}_t\Big). \tag{6.4.22}$$

$$\implies \mathbf{x}_0^*(\mathbf{x}_t, t) = \mathbf{x}_t + (1-\bar{\alpha}_t)\ \boldsymbol{\Sigma}\mathbf{s}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, t) = \mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\ \sqrt{\boldsymbol{\Sigma}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}^*}(\mathbf{x}_t, t). \tag{6.4.23}$$

SDE formulation: Score-based diffusion models have also been analyzed as stochastic differential equations (SDEs) Song et al. [2021b]. The SDE version of NI-DDPM, which we call Non-Isotropic Variance Preserving (NIVP) SDE, is (see Section 6.2.11 for derivation):

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\ \mathrm{d}t + \sqrt{\beta(t)}\sqrt{\boldsymbol{\Sigma}}\ \mathrm{d}\mathbf{w}. \tag{6.4.24}$$

$$\implies p_{0t}\Big(\mathbf{x}(t) \mid \mathbf{x}(0)\Big) = \mathcal{N}\left(\mathbf{x}(0)\ e^{-\frac{1}{2}\int_0^t \beta(s)\mathrm{d}s},\ \boldsymbol{\Sigma}(\mathbf{I} - \mathbf{I}e^{-\int_0^t \beta(s)\mathrm{d}s})\right). \tag{6.4.25}$$

Finally, Section 2.4 and Section 6.2 contain more detailed derivations of the above equations for DDPM Ho et al. [2020] and our NI-DDPM. See Section 2.5 and Section 6.3 for the equivalent derivations for Score Matching Langevin Dynamics (SMLD) Song and Ermon [2019, 2020], and our Non-Isotropic SMLD (NI-SMLD).

6.5 Gaussian Free Field (GFF) images

An instantiation of non-isotropic Gaussian noise is Gaussian Free Field [Sheffield, 2007] (GFF). In 2D, this manifests as a GFF image. A GFF image $\mathbf{g}$ can be obtained from a normal noise image $\mathbf{z}$ as follows [Sheffield, 2007]:

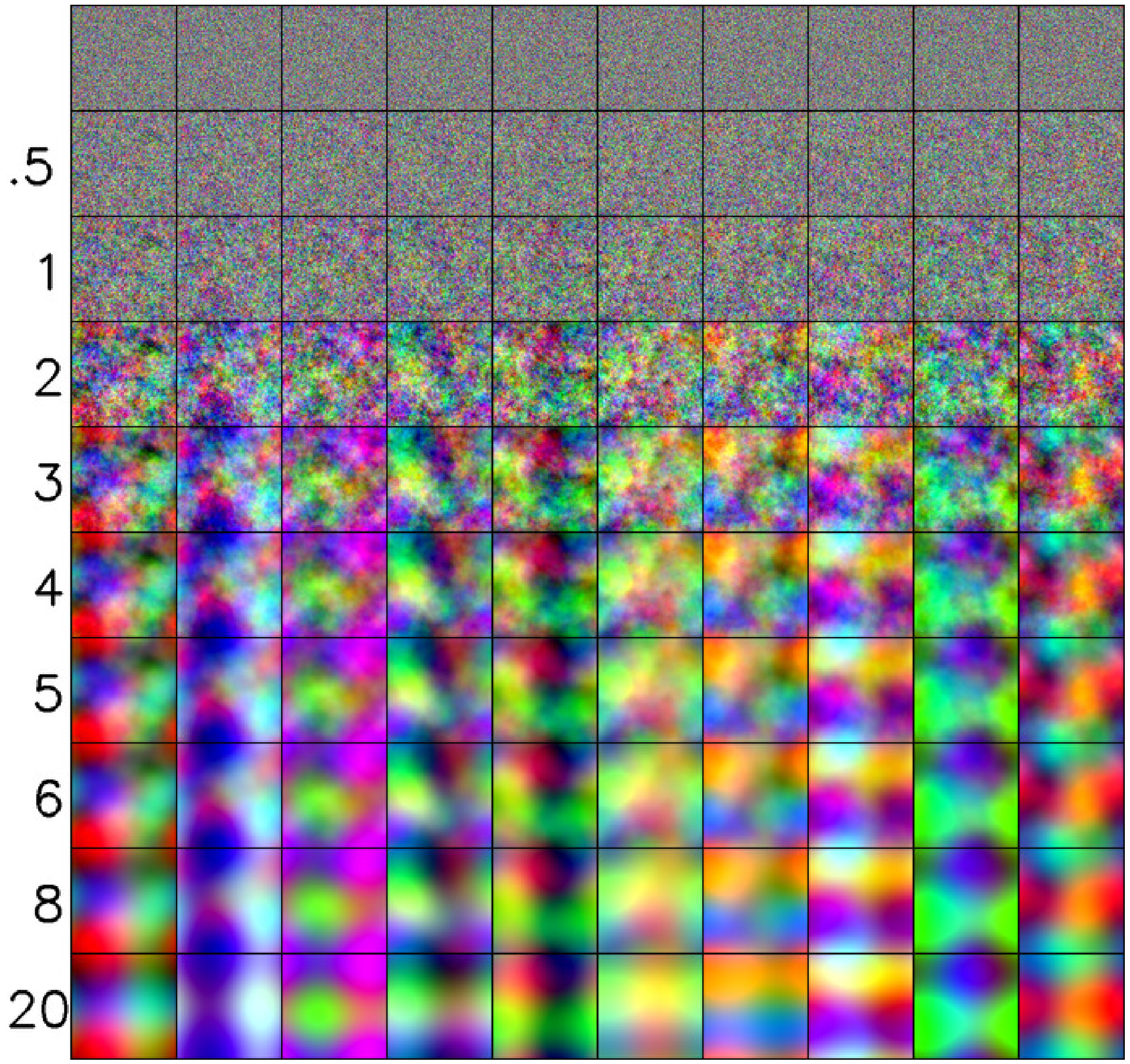


Fig. 6.3. (left to right) 10 GFF images (each varying downwards) as a function of the power γ of the index (mentioned on the left).

(1) First, sample an $n \times n$ noise image $\mathbf{z}$ from the standard complex normal distribution with covariance matrix $\Gamma = \mathbf{I}_N$ where $N = n^2$ is the total number of pixels, and pseudo-covariance matrix $C = \mathbf{0}$: $\mathbf{z} \sim \mathcal{CN}(\mathbf{0}, \mathbf{I}_N, \mathbf{0})$.
The standard **complex** normal distribution is one where the real part $\mathbf{x}$ and imaginary part $\mathbf{y}$ are each distributed as the standard normal distribution with variance $\frac{1}{2}\mathbf{I}_N$. Let $\mathbf{\Sigma_{ab}}$ be the covariance matrix between $\mathbf{a}$ and $\mathbf{b}$. We know that $\mathbf{\Sigma_{xx}} = \mathbf{\Sigma_{yy}} = \frac{1}{2}\mathbf{I}_N$, and $\mathbf{\Sigma_{xy}} = \mathbf{\Sigma_{yx}} = \mathbf{0}_N$ Then:

$$\Gamma = \mathbb{E}_{\mathbf{z}}[\mathbf{z}\mathbf{z}^H] = \mathbf{\Sigma_{xx}} + \mathbf{\Sigma_{yy}} + i(\mathbf{\Sigma_{yx}} - \mathbf{\Sigma_{xy}}) = \mathbf{I}_N. \tag{6.5.1}$$

$$C = \mathbb{E}_{\mathbf{z}}[\mathbf{z}\mathbf{z}^T] = \mathbf{\Sigma_{xx}} - \mathbf{\Sigma_{yy}} + i(\mathbf{\Sigma_{yx}} + \mathbf{\Sigma_{xy}}) = \mathbf{0}_N. \tag{6.5.2}$$

(2) Apply the Discrete Fourier Transform using the $N \times N$ weights matrix $\mathbf{W}_N$: $\mathbf{W}_N\mathbf{z}$.

(3) Consider a diagonal $N \times N$ matrix of the reciprocal of an index value k_{ij} per pixel (i,j) in Fourier space : $\mathbf{K}^{-1} = [1/|k_{ij}|]_{(i,j)}$, and multiply this with the above: $\mathbf{K}^{-1}\mathbf{W}_N\mathbf{z}$.

(4) Take its Inverse Discrete Fourier Transform ($\mathbf{W}_N^{-1}$) to make the raw GFF image: $\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{W}_N\mathbf{z}$. However, this results in a GFF image with a small non-unit variance.

(5) Normalize the above GFF image with the standard deviation σ_N at its resolution N, so that it has unit variance (see Section 6.5.1 for derivation of σ_N):

$$\mathbf{g}_{\text{complex}} = \frac{1}{\sigma_N}\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{W}_N\mathbf{z} \quad \iff \quad \mathbf{z} = \sigma_N\mathbf{W}_N^{-1}\mathbf{K}\mathbf{W}_N\mathbf{g}_{\text{complex}}. \tag{6.5.3}$$

(6) Extract only the real part of $\mathbf{g}_{\text{complex}}$, and normalize (see Section 6.5.1 for derivation):

$$\mathbf{g} = \frac{1}{\sqrt{2N}\sigma_N}\text{Real}\left(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{W}_N\mathbf{z}\right). \iff \mathbf{z} = \sqrt{2N}\sigma_N\text{Real}\left(\mathbf{W}_N^{-1}\mathbf{K}\mathbf{W}_N\mathbf{g}\right). \tag{6.5.4}$$

See Figures 6.1 and 6.3 for examples of GFF images. Effectively, this prioritizes lower frequencies over higher, making noise smoother and correlated.

The probability distribution of GFF images $\mathbf{g}$ can be seen as a non-isotropic multivariate Gaussian with mean $\mathbf{0}$, and a non-diagonal covariance matrix $\mathbf{\Sigma}$ (see Section 6.5.1 for derivation):

$$p(\mathbf{g}) = \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}). \tag{6.5.5}$$

$$\mathbf{\Sigma} = \sqrt{\mathbf{\Sigma}}\sqrt{\mathbf{\Sigma}}^T. \tag{6.5.6}$$

$$\sqrt{\mathbf{\Sigma}} = \frac{1}{\sqrt{2N}\sigma_N}\text{Real}\left(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{W}_N\right). \tag{6.5.7}$$

$$\implies \mathbf{g} = \sqrt{\mathbf{\Sigma}}\mathbf{z}. \tag{6.5.8}$$

6.5.1 Probability distribution of GFF

Let the probability distribution of GFF images be $\mathcal{G}$. This can be seen as a non-isotropic multivariate Gaussian with a non-diagonal covariance matrix $\mathbf{\Sigma}$:

$$\mathbf{g} \sim \mathcal{G} = \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma}) = \mathcal{N}(\mathbf{0}_N, \mathbf{\Sigma}). \tag{6.5.9}$$

We know from the properties of Discrete Fourier Transform (following the normalization convention of the Pytorch / Numpy implementation) that, given the Discrete Fourier Transform matrix $\mathbf{W}_N$:

$$\mathbf{W}_N = \mathbf{W}_N^T; \mathbf{W}_N^{-1} = \mathbf{W}_N^{-1^T}; \mathbf{W}_N^{-1} = \frac{1}{N}\mathbf{W}_N^{*} = \frac{1}{N}\mathbf{W}_N^{H}. \tag{6.5.10}$$

Here, $\mathbf{g}$ is real and $\mathbf{z}$ is complex, and $\mathbf{g}_\text{c}$ is complex (real+imaginary).

$\boldsymbol{\mu}$ is given by:

$$\begin{aligned}
\boldsymbol{\mu} &= \mathbb{E}_\mathbf{g}[\mathbf{g}] = \mathbb{E}_\mathbf{g}[\ \frac{1}{2\sigma_N}(\mathbf{g}_\text{c} + \mathbf{g}_\text{c}^*)\ ] = \frac{1}{2\sigma_N}\left(\mathbb{E}_\mathbf{g}[\mathbf{g}_\text{c}] + \mathbb{E}_\mathbf{g}[\mathbf{g}_\text{c}^*]\right), \\
&= \frac{1}{2\sigma_N}\Big(\mathbb{E}_\mathbf{z}[\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z}] + \mathbb{E}_\mathbf{z}[(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z})^*]\Big), \\
&= \frac{1}{2\sigma_N}\Big(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbb{E}_\mathbf{z}[\mathbf{z}] + \mathbf{W}_N^{-1*}\mathbf{K}^{-1}\mathbb{E}_\mathbf{z}[\mathbf{z}^*]\Big), \\
\implies \boldsymbol{\mu} &= \mathbf{0}_N. \qquad\qquad [\because \mathbb{E}_\mathbf{z}[\mathbf{z}] = \mathbb{E}_\mathbf{z}[\mathbf{z}^*] = \mathbf{0}_N]
\end{aligned} \tag{6.5.11}$$

$\mathbf{\Sigma}$ is given by:

$$\begin{aligned}
\mathbf{\Sigma} &= \mathbb{E}_\mathbf{g}[\ \mathbf{g}\ \mathbf{g}^T\ ], \\
&= \mathbb{E}_\mathbf{g}[\ \frac{1}{2\sigma_N}(\mathbf{g}_\text{c} + \mathbf{g}_\text{c}^*)\ \frac{1}{2\sigma_N}(\mathbf{g}_\text{c} + \mathbf{g}_\text{c}^*)^T\ ], \\
&= \frac{1}{4\sigma_N^2}\mathbb{E}_\mathbf{g}[\ (\mathbf{g}_\text{c} + \mathbf{g}_\text{c}^*)\ (\mathbf{g}_\text{c}^T + \mathbf{g}_\text{c}^H)\ ], \\
&= \frac{1}{4\sigma_N^2}\mathbb{E}_\mathbf{g}[\ \mathbf{g}_\text{c}\mathbf{g}_\text{c}^T + \mathbf{g}_\text{c}\mathbf{g}_\text{c}^H + \mathbf{g}_\text{c}^*\mathbf{g}_\text{c}^T + \mathbf{g}_\text{c}^*\mathbf{g}_\text{c}^H\ ], \\
&= \frac{1}{4\sigma_N^2}\Big(\mathbb{E}_\mathbf{g}[\mathbf{g}_\text{c}\mathbf{g}_\text{c}^T] + \mathbb{E}_\mathbf{g}[\mathbf{g}_\text{c}\mathbf{g}_\text{c}^H] + \mathbb{E}_\mathbf{g}[\mathbf{g}_\text{c}^*\mathbf{g}_\text{c}^T] + \mathbb{E}_\mathbf{g}[\mathbf{g}_\text{c}^*\mathbf{g}_\text{c}^H]\Big). \\
\mathbb{E}_\mathbf{g}[\mathbf{g}_\text{c}\mathbf{g}_\text{c}^T] &= \mathbb{E}_\mathbf{z}[\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z}\ (\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z})^T], \\
&= \mathbb{E}_\mathbf{z}[\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z}\ \mathbf{z}^T\mathbf{K}^{-1}\mathbf{W}_N^{-T}], \quad [\because \mathbf{K}^{-1} \text{ is diagonal }] \\
&= \mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbb{E}_\mathbf{z}[\mathbf{z}\mathbf{z}^T]\mathbf{K}^{-1}\mathbf{W}_N^{-T}, \\
&= \mathbf{0}_N. \qquad\qquad [\because \mathbb{E}_\mathbf{z}[\mathbf{z}\mathbf{z}^T] = \mathbf{0}_N \text{ (eq. (6.5.2))}] \\
\mathbb{E}_\mathbf{g}[\mathbf{g}_\text{c}\mathbf{g}_\text{c}^H] &= \mathbb{E}_\mathbf{z}[\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z}\ (\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z})^H], \\
&= \mathbb{E}_\mathbf{z}[\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z}\ \mathbf{z}^H\mathbf{K}^{-1}\mathbf{W}_N^{-H}], \quad [\because \boldsymbol{K}^{-1} \text{ is real diagonal }]
\end{aligned}$$

$$
\begin{aligned}
&= \mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbb{E}_{\mathbf{z}}[\mathbf{z}\mathbf{z}^H]\mathbf{K}^{-1}\frac{1}{N}\mathbf{W}_N, \quad [\because \mathbf{W}_N^{-1} = \frac{1}{N}\mathbf{W}_N^H \text{ (eq. (6.5.10))}]\\
&= \frac{1}{N}\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N. \qquad [\because \mathbb{E}_{\mathbf{z}}[\mathbf{z}\mathbf{z}^H] = \mathbf{I}_N \text{ (eq. (6.5.1))}]\\
\mathbb{E}_{\mathbf{g}}[\mathbf{g}_c^*\mathbf{g}_c^T] &= \mathbb{E}_{\mathbf{z}}[(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z})^* \ (\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z})^T],\\
&= \mathbb{E}_{\mathbf{z}}[\mathbf{W}_N^{-1*}\mathbf{K}^{-1}\mathbf{z}^* \ \mathbf{z}^T\mathbf{K}^{-1}\mathbf{W}_N^{-1}],\\
&= \frac{1}{N}\mathbf{W}_N\mathbf{K}^{-1}\mathbb{E}_{\mathbf{z}}[\mathbf{z}^*\mathbf{z}^T]\mathbf{K}^{-1}\mathbf{W}_N^{-1}, \quad [\because \mathbf{W}_N^{-1} = \frac{1}{N}\mathbf{W}_N^* \text{ (eq. (6.5.10))}]\\
&= \frac{1}{N}\mathbf{W}_N\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N^{-1}. \quad [\because \mathbb{E}_{\mathbf{z}}[\mathbf{z}^*\mathbf{z}^T] = \mathbb{E}_{\mathbf{z}}[\mathbf{z}\mathbf{z}^H]^* = \mathbf{I}_N \text{ (eq. (6.5.1))}]\\
\mathbb{E}_{\mathbf{g}}[\mathbf{g}_c^*\mathbf{g}_c^H] &= \mathbb{E}_{\mathbf{z}}[(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z})^* \ (\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{z})^H],\\
&= \mathbb{E}_{\mathbf{z}}[\mathbf{W}_N^{-1*}\mathbf{K}^{-1}\mathbf{z}^* \ \mathbf{z}^H\mathbf{K}^{-1}\mathbf{W}_N^{-H}],\\
&= \mathbf{W}_N^{-1*}\mathbf{K}^{-1}\mathbb{E}_{\mathbf{z}}[\mathbf{z}^*\mathbf{z}^H]\mathbf{K}^{-1}\mathbf{W}_N^{-H},\\
&= \mathbf{0}_N. \qquad [\because \mathbb{E}_{\mathbf{z}}[\mathbf{z}^*\mathbf{z}^H] = \mathbb{E}_{\mathbf{z}}[\mathbf{z}\mathbf{z}^T]^* = \mathbf{0}_N \text{ (eq. (6.5.1))}]\\
\implies \boldsymbol{\Sigma} &= \frac{1}{4\sigma_N^2}\left(\mathbf{0}_N + \frac{1}{N}\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N + \frac{1}{N}\mathbf{W}_N\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N^{-1} + \mathbf{0}_N\right),\\
&= \frac{1}{4N\sigma_N^2}\left(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N + \mathbf{W}_N\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N^{-1}\right),\\
&= \frac{1}{4N\sigma_N^2}\left(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N + (N\mathbf{W}_N^{-1*})\mathbf{K}^{-1}\mathbf{K}^{-1}(\frac{1}{N}\mathbf{W}_N^*)\right),\\
&= \frac{1}{2N\sigma_N^2}\left(\frac{1}{2}\left(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N + (\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N)^*\right)\right),\\
\implies \boldsymbol{\Sigma} &= \frac{1}{2N\sigma_N^2} \ \text{Real}\left(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{K}^{-1}\mathbf{W}_N\right). \qquad (6.5.12)\\
\sqrt{\boldsymbol{\Sigma}} &= \frac{1}{\sqrt{2N}\sigma_N} \ \text{Real}\left(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{W}_N\right), \qquad (6.5.13)\\
\boldsymbol{\Sigma}^{-1} &= 2N\sigma_N^2 \ \text{Real}\left(\mathbf{W}_N^{-1}\mathbf{K}\mathbf{K}\mathbf{W}_N\right), \qquad (6.5.14)\\
\sqrt{\boldsymbol{\Sigma}^{-1}} &= \sqrt{2N}\sigma_N \ \text{Real}\left(\mathbf{W}_N^{-1}\mathbf{K}\mathbf{W}_N\right). \qquad (6.5.15)
\end{aligned}
$$

6.5.2 Log probability of transformation

$$
\begin{aligned}
\mathbf{g} = \sqrt{\boldsymbol{\Sigma}}\mathbf{z} \implies \log p(\mathbf{g}) &= \log p(\mathbf{z}) - \log\left|\det\frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\mathbf{z}}\right| = \log p(\mathbf{z}) - \log\left|\det\sqrt{\boldsymbol{\Sigma}}\right|\\
&= \log p(\mathbf{z}) - \log\left|\det\frac{1}{\sqrt{2N}\sigma_N}\text{Real}\left(\mathbf{W}_N^{-1}\mathbf{K}^{-1}\mathbf{W}_N\right)\right|\\
&= \log p(\mathbf{z}) - \frac{1}{\sqrt{2N}\sigma_N}\log\left|\det\mathbf{K}^{-1}\right|.
\end{aligned}
$$

This is useful for building (normalizing) flows using non-isotropic Gaussian noise.

6.5.3 Varying K

The index matrix $\mathbf{K}$ involves computation of an index value k_{ij} per pixel (i,j). However, this index value could be raised to any power γ i.e. $|k_{ij}|^{\gamma}$. The effect of varying γ can be seen in Figure 6.3 : greater the γ, the more correlated are neighbouring pixels.

6.6 Experimental results

Table 6.1. Image generation metrics FID, Precision (P), and Recall (R) for CIFAR10 using DDPM and NI-DDPM, with different generation steps.

CIFAR10	**steps**	**FID** $\downarrow$	**P** $\uparrow$	**R** $\uparrow$
DDPM	1000	6.05	0.66	0.54
	100	12.25	0.62	0.48
	50	16.61	0.60	0.43
	20	26.35	0.56	0.24
	10	44.95	0.49	0.24
NI-DDPM	1000	6.95	0.62	0.53
	100	12.68	0.60	0.49
	50	16.91	0.57	0.45
	20	30.41	0.52	0.35
	10	60.32	0.43	0.23

We train two models on CIFAR10, one using DDPM and the other using NI-DDPM with the exact same hyperparameters (batch size, learning rate, etc.) for 300,000 iterations. We then sample 50,000 images from each, and calculate the image generation metrics of Fréchet Inception Distance (FID) Heusel et al. [2017], Precision (P), and Recall (R). Although the models were trained on 1000 steps between data and noise, we report these metrics while sampling images using 1000, and smaller steps: 100, 50, 20, 10.

As can be seen from Table 6.1, our non-isotropic variant performs comparable to the isotropic baseline. The difference between them increases with decreasing number of steps between noise and data. This provides a reasonable proof-of-concept that non-isotropic Gaussian noise works just as well as isotropic noise when used in denoising diffusion models for image generation. These conclusions could be generalized to denoising diffusion models of any modality, since the theoretical framework of Non-Isotropic Denoising Diffusion Models remains intact irrespective of the modality of data.

6.7 Conclusion

We have presented the key mathematics behind non-isotropic Gaussian DDPMs, as well as a complete example using a GFF. We then noted quantitative comparison of using GFF noise vs. regular noise on the CIFAR-10 dataset. In the appendix, we also include further derivations for non-isotropic SMLD models. GFFs are just one example of a well known class of models that are a subset of non-isotropic Gaussian distributions. In the same way that other work has examined non-Gaussian distributions such as the Gamma distribution [Nachmani et al., 2021], Poisson distribution [Xu et al., 2022], and Heat dissipation processes [Rissanen et al., 2022], we hope that our work here may lay the foundation for other new denoising diffusion formulations.

Chapter 7

MCVD: Masked Conditional Video Diffusion [Voleti et al., 2022a]

7.0 Prologue to article

7.0.1 Article details

MCVD: Masked Conditional Video Diffusion for Prediction, Generation, and Interpolation. Vikram Voleti*, Alexia Jolicoeur-Martineau*, Christopher Pal (*denotes equal contribution). *Advances in Neural Information Processing Systems (NeurIPS) 2022.*

Personal contribution: The project began with discussions between the authors at Mila. The idea was to apply denoising diffusion models to model the modality of video. Since denoising diffusion models were shown to work very well on images, it was to be seen whether they could be trained to generate videos. Vikram Voleti and Alexia Jolicoeur-Martineau initially discussed some ideas with other members of the research community, however they did not work as expected. Then Vikram Voleti worked on an initial idea, wrote the code, and conducted preliminary experiments using Moving MNIST that proved that it was indeed possible for denoising diffusion models to model video successfully. Vikram Voleti and Alexia Jolicoeur-Martineau then joined forces to scale this up to bigger and more complex datasets. Christopher Pal provided advice and guidance throughout the project, provided the idea of masking the past and future frames so that a single model can solve all video tasks simultaneously, and wrote parts of the paper. Vikram and Alexia both contributed to the code, experimental design, experiments, improvements to the model architecture, metrics for evaluation, sampling techniques, writing of the final publication, rebuttal, release of code and model checkpoints.

7.0.2 Context

Video prediction is a challenging task. The quality of video frames from current state-of-the-art generative models tends to be poor and generalization beyond the training data is difficult.

Furthermore, existing prediction frameworks are typically not capable of simultaneously handling other video-related tasks such as unconditional generation or interpolation. The recently proposed denoising diffusion models, although very successful at generating images, had not been applied to the modality of video yet.

7.0.3 Contributions

In this work, we devise a general-purpose framework called Masked Conditional Video Diffusion (MCVD) for all of these video synthesis tasks using a probabilistic conditional score-based denoising diffusion model, conditioned on past and/or future frames. We train the model in a manner where we randomly and independently mask all the past frames or all the future frames. This novel but straightforward setup allows us to train a single model that is capable of executing a broad range of video tasks, specifically: future/past prediction – when only future/past frames are masked; unconditional generation – when both past and future frames are masked; and interpolation – when neither past nor future frames are masked. Our experiments show that this approach can generate high-quality frames for diverse types of videos. Our MCVD models are built from simple non-recurrent 2D-convolutional architectures, conditioning on blocks of frames and generating blocks of frames. We generate videos of arbitrary lengths autoregressively in a block-wise manner. Our approach yields state-of-the-art results across standard video prediction and interpolation benchmarks, with computation times for training models measured in 1-12 days using ≤ 4 GPUs.

Project page: `https://mask-cond-video-diffusion.github.io`

Code: `https://mask-cond-video-diffusion.github.io/`

7.0.4 Research impact

Denoising diffusion models have quickly become the default generative models for most modalities: images, video, audio, 3D, etc. Since our work was published, several works (including a few concurrent works) have used denoising diffusion models to generate videos [Ho et al., 2022b, Yang et al., 2022, Harvey et al., 2022, Höppe et al., 2022, Yu et al., 2023, He et al., 2022, Nikankin et al., 2022, Luo et al., 2023, Yin et al., 2023]. Others have further conditioned them on text prompts to make text-to-video models, including Make-A-Video by Meta [Singer et al., 2022], ImagenVideo by Google [Ho et al., 2022a], Phenaki by Google [Villegas et al., 2023], MagicVideo by ByteDance [Zhou et al., 2022], Gen2 by RunwayML [Esser et al., 2023], etc. This has been taken to the next level in Make-A-Video3D by Meta [Singer et al., 2023], which Vikram contributed to during an internship at Meta, to generate 4D scenes from text prompts i.e. text-to-4D based on text-to-video models.

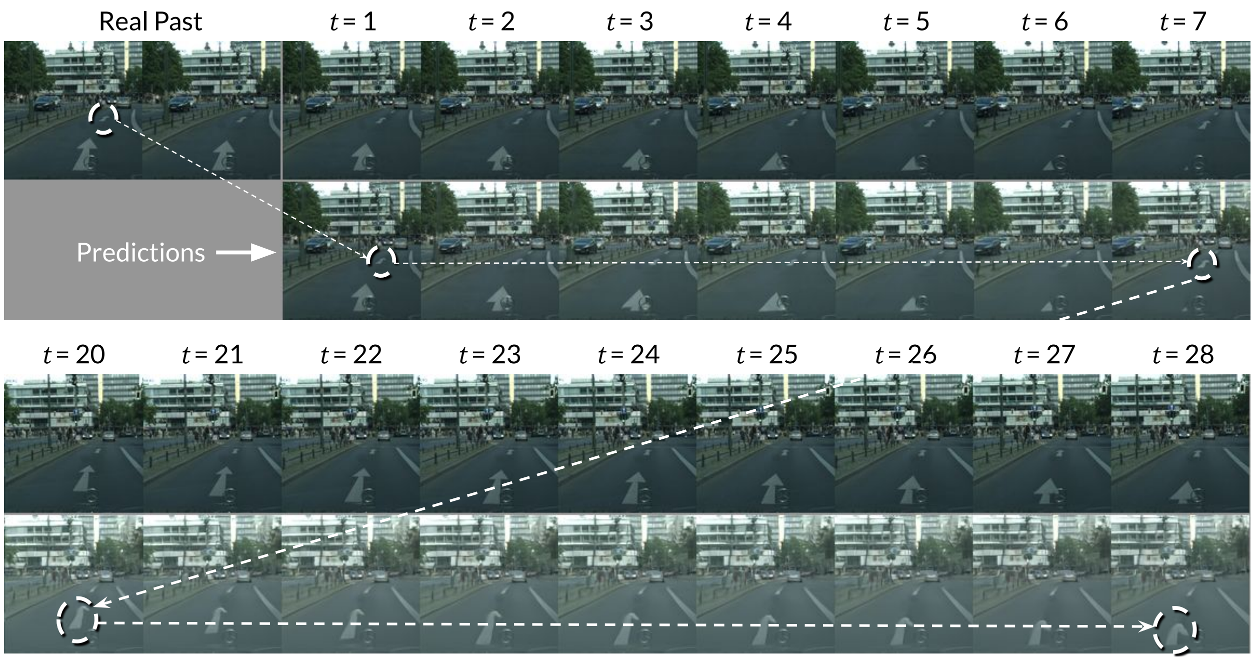


Fig. 7.1. Our approach generates high quality frames many steps into the future: Given two conditioning frames from the Cityscapes [Cordts et al., 2016] validation set (top left), we show 7 predicted future frames in row 2 below, then skip to frames 20-28, autoregressively predicted in row 4. Ground truth frames are shown in rows 1 and 3. Notice the initial large arrow advancing and passing under the car. In frame 20 (the far left of the 3rd and 4th row), the initially small and barely visible second arrow in the background of the conditioning frames has advanced into the foreground. Result generated by our **MCVD** method (concat variant). Note that some Cityscapes videos contain brightness changes, which may explain the brightness change in this sample.

7.1 Introduction

Predicting what one may visually perceive in the future is closely linked to the dynamics of objects and people. As such, this kind of prediction relates to many crucial human decision-making tasks ranging from making dinner to driving a car. If video models could generate full-fledged videos in pixel-level detail with plausible futures, agents could use them to make better decisions, especially safety-critical ones. Consider, for example, the task of driving a car in a tight situation at high speed. Having an accurate model of the future could mean the difference between damaging a car or something worse. We can obtain some intuitions about this scenario by examining the predictions of our model in Figure 7.1, where we condition on two frames and predict 28 frames into the future for a car driving around a corner. We can see that this is enough time for two different painted arrows to pass under the car. If one zooms in, one can inspect the relative positions of the arrow and the Mercedes hood ornament in the real versus predicted frames. Pixel-level models of trajectories, pedestrians, potholes, and debris on the road could one day improve the safety of vehicles.

Although beneficial to decision making, video generation is an incredibly challenging problem; not only must high-quality frames be generated, but the changes over time must be plausible and ideally drawn from an accurate and potentially complex distribution over probable futures. Looking far in time is exceptionally hard given the exponential increase in possible futures. Generating video from scratch or unconditionally further compounds the problem because even the structure of the first frame must be synthesized. Also related to video generation are the simpler tasks of a) video prediction, predicting the future given the past, and b) interpolation, predicting the in-between given past and future. Yet, both problems remain challenging. Specialized tools exist to solve the various video tasks, but they rarely solve more than one task at a time.

Given the monumental task of general video generation, current approaches are still very limited despite the fact that many state of the art methods have hundreds of millions of parameters [Wu et al., 2021, Weissenborn et al., 2019, Villegas et al., 2019, Babaeizadeh et al., 2021]. While industrial research is capable of looking at even larger models, current methods frequently underfit the data, leading to blurry videos, especially in the longer-term future and recent work has examined ways in improve parameter efficiency [Babaeizadeh et al., 2021]. Our objective here is to devise a video generation approach that generates high-quality, time-consistent videos within our computation budget of $\leq$ 4 GPU) and computation times for training models $\leq$ two weeks. Fortunately, diffusion models for image synthesis have demonstrated wide success, which strongly motivated our use of this approach. Our qualitative results in Figure 7.1 also indicate that our particular approach does quite well at synthesizing frames in the longer-term future (i.e., frame 29 in the bottom right corner).

One family of diffusion models might be characterized as Denoising Diffusion Probabilistic Models (DDPMs) [Sohl-Dickstein et al., 2015, Ho et al., 2020, Dhariwal and Nichol, 2021], while another as Score-based Generative Models (SGMs) [Song and Ermon, 2019, Li et al., 2019, Song and Ermon, 2020, Jolicoeur-Martineau et al., 2021b]. However, these approaches have effectively merged into a field we shall refer to as score-based diffusion models, which work by defining a stochastic process from data to noise and then reversing that process to go from noise to data. Their main benefits are that they generate very 1) high-quality and 2) diverse data samples. One of their drawbacks is that solving the reverse process is relatively slow, but there are ways to improve speed [Song et al., 2021a, Jolicoeur-Martineau et al., 2021a, Salimans and Ho, 2022, Liu et al., 2022, Xiao et al., 2022]. Given their massive success and attractive properties, we focus here on developing our framework using score-based diffusion models for video prediction, generation, and interpolation.

Our work makes the following contributions:

(1) A conditional video diffusion approach for video prediction and interpolation that yields state-of-the-art (SOTA) results.

(2) A conditioning procedure based on masking past and/or future frames in a blockwise manner giving a single model the ability to solve multiple video tasks: future/past prediction, unconditional generation, and interpolation.
(3) A sliding window *blockwise autoregressive* conditioning procedure to allow fast and coherent long-term generation (Figure 7.2).
(4) A convolutional U-net neural architecture integrating recent developments with a conditional normalization technique we call SPAce-TIme-Adaptive Normalization (SPATIN) (Figure 7.4).

By conditioning on blocks of frames in the past and optionally blocks of frames even further in the future, we are able to better ensure that temporal dynamics are transferred across blocks of samples, i.e. our networks can learn *implicit* models of spatio-temporal dynamics to inform frame generation. Unlike many other approaches, we do not have explicit model components for spatio-temporal derivatives or optical flow or recurrent blocks.

7.2 Masked conditional diffusion for video

Let $\mathbf{x}_0 \in \mathbb{R}^d$ be a sample from the data distribution p_{data}. A sample $\mathbf{x}_0$ can corrupted from $t = 0$ to $t = T$ through the Forward Diffusion Process (FDP) with the following transition kernel:

$$q_t(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}). \tag{7.2.1}$$

Furthermore, $\mathbf{x}_t$ can be sampled directly from $\mathbf{x}_0$ using the following accumulated kernel:

$$q_t(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \implies \mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \tag{7.2.2}$$

where $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$, and $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Generating new samples can be done by reversing the FDP and solving the Reverse Diffusion Process (RDP) starting from Gaussian noise $\mathbf{x}_T$. It can be shown (Song et al. [2021b], Ho et al. [2020]) that the RDP can be computed using the following transition kernel:

$$p_t(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}),$$

$$\text{where} \quad \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \quad \text{and} \quad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t. \tag{7.2.3}$$

Since $\mathbf{x}_0$ given $\mathbf{x}_t$ is unknown, it can be estimated using eq. (7.2.2): $\hat{\mathbf{x}}_0 = \left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}\right)/\sqrt{\bar{\alpha}_t}$, where $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|t)$ estimates $\boldsymbol{\epsilon}$ using a time-conditional neural network parameterized by θ. This allows us to reverse the process from noise to data. The loss function of the neural network is:

$$L(\theta) = \mathbb{E}_{t, \mathbf{x}_0 \sim p_{\text{data}}, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \mid t)\right\|_2^2\right]. \tag{7.2.4}$$

Note that estimating ϵ is equivalent to estimating a scaled version of the score function (i.e., the gradient of the log density) of the noisy data:

$$\nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t \mid \mathbf{x}_0) = -\frac{1}{1-\bar{\alpha}_t}(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}. \tag{7.2.5}$$

Thus, data generation through denoising depends on the score-function, and can be seen as noise-conditional score-based generation.

Score-based diffusion models can be straightforwardly adapted to video by considering the joint distribution of multiple continuous frames. While this is sufficient for unconditional video generation, other tasks such as video interpolation and prediction remain unsolved.

A conditional video prediction model can be approximately derived from the unconditional model using imputation [Song et al., 2021b]; indeed, the contemporary work of Ho et al. [2022b] attempts to use this technique; however, their approach is based on an approximate conditional model.

7.2.1 Video prediction

We first propose to directly model the conditional distribution of video frames in the immediate future given past frames. Assume we have p past frames $\mathbf{p} = \{\mathbf{p}^i\}_{i=1}^p$ and k current frames in the immediate future $\mathbf{x}_0 = \{\mathbf{x}_0^i\}_{i=1}^k$. We condition the above diffusion models on the past frames to predict the current frames:

$$L_{\text{vidpred}}(\theta) = \mathbb{E}_{t,[\mathbf{p},\mathbf{x}_0]\sim p_{\text{data}},\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \mid \mathbf{p}, t)\right\|^2\right]. \tag{7.2.6}$$

Given a model trained as above, video prediction for subsequent time steps can be achieved by blockwise autoregressively predicting current video frames conditioned on previously predicted frames (see Figure 7.2). We use variants of the network shown in Figure 7.4 to model $\boldsymbol{\epsilon}_\theta$ in Equation 7.2.6 here, and for Equation 7.2.7 and Equation 7.2.8 below.

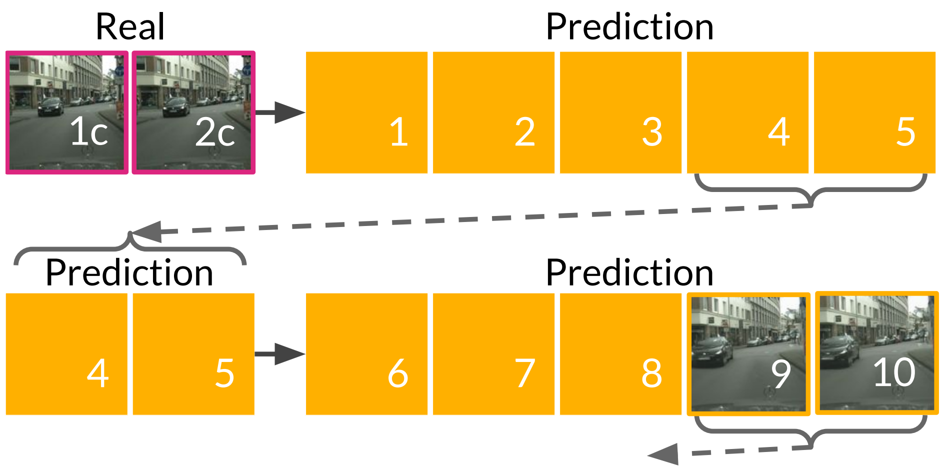


Fig. 7.2. Blockwise autoregressive prediction with our model.

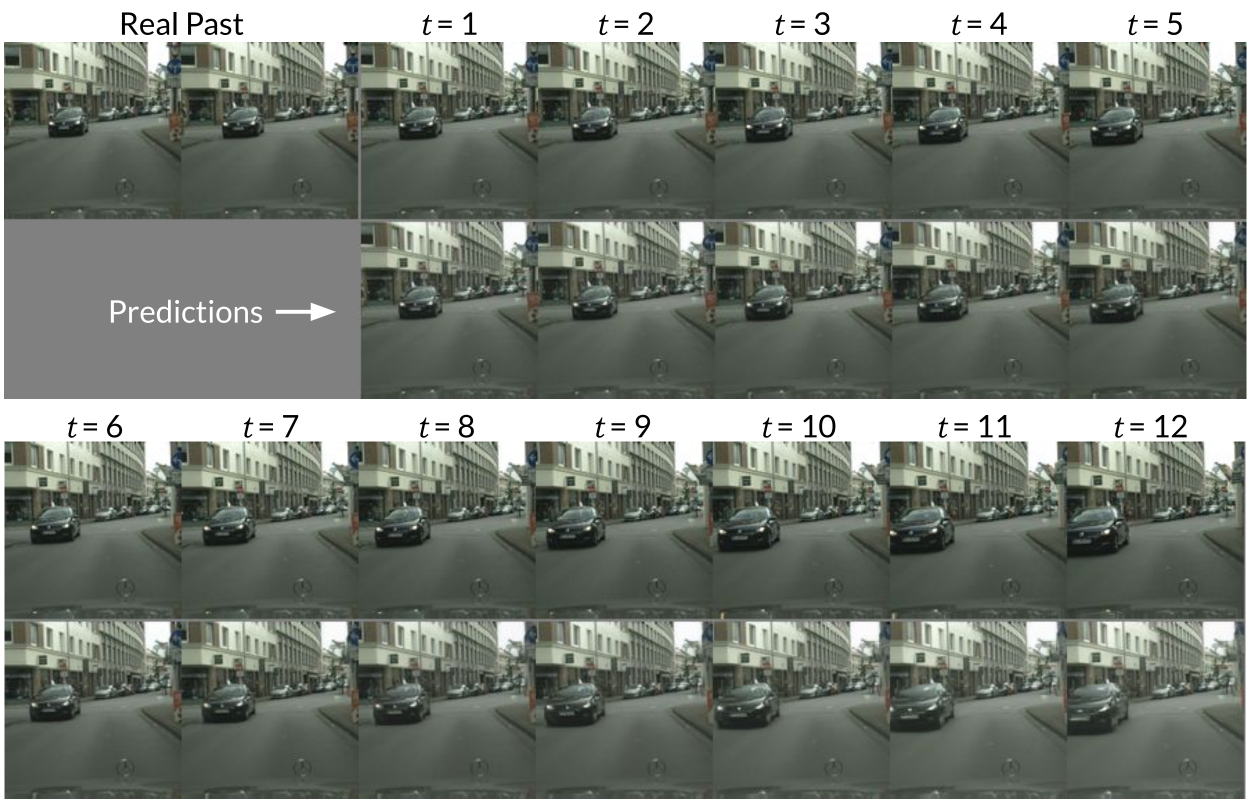


Fig. 7.3. This figure shows our block autoregressive strategy where the top row and third row are ground truth, and the second and fourth rows show the blockwise autoregressively generated frames using our approach.

7.2.2 Video prediction + generation

Our approach above allows video prediction, but not unconditional video generation. As a second approach, we extend the same framework to video generation by masking (zeroing-out) the past frames with probability $p_{\text{mask}} = 1/2$ using binary mask m_p. The network thus learns to predict the noise added without any past frames for context. Doing so means that we can perform conditional as well as unconditional frame generation, i.e., video prediction and generation with the same network. This leads to the following loss ($\mathcal{B}$ is the Bernouilli distribution):

$$L_{\text{vidgen}}(\theta) = \mathbb{E}_{t,[\mathbf{p},\mathbf{x}_0]\sim p_{\text{data}},\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),m_p\sim\mathcal{B}(p_{\text{mask}})}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \mid m_p\mathbf{p}, t)\right\|^2\right]. \quad (7.2.7)$$

We hypothesize that this dropout-like [Srivastava et al., 2014] approach will also serve as a form of regularization, improving the model's ability to perform predictions conditioned on the past. We see positive evidence of this effect in our experiments – see the MCVD past-mask model variants in Tables 7.5 and 7.9 versus without past-masking. Note that random masking is used only during training.

7.2.3 Video prediction + generation + interpolation

We now have a design for video prediction and generation, but it still cannot perform video interpolation nor past prediction from the future. As a third and final approach, we show how to build a general model for solving all four video tasks. Assume we have p past frames, k current frames, and f future frames $\mathbf{f} = \{\mathbf{f}^i\}_{i=1}^{f}$. We randomly mask the p past frames with probability $p_{mask} = 1/2$, and similarly randomly mask the f future frames with the same probability (but sampled separately). Thus, future or past prediction is when only future or past frames are masked. Unconditional generation is when both past and future frames are masked. Video interpolation is when neither past nor future frames are masked. The loss function for this general video machinery is:

$$L(\theta) = \mathbb{E}_{t,[\mathbf{p},\mathbf{x}_0,\mathbf{f}]\sim p_{\text{data}},\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),(m_p,m_f)\sim\mathcal{B}(p_{\text{mask}})}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \mid m_p\mathbf{p}, m_f\mathbf{f}, t)\right\|^2\right]. \tag{7.2.8}$$

The three video tasks and our solutions are visualized in Figure 7.6.

7.2.4 Network architecture

For our denoising network we use a U-net architecture [Ronneberger et al., 2015, Honari et al., 2016, Salimans et al., 2017] combining the improvements from Song et al. [2021b] and Dhariwal and Nichol [2021]. This architecture uses a mix of 2D convolutions [Fukushima and Miyake, 1982], multi-head self-attention [Cheng et al., 2016], and adaptive group-norm [Wu and He, 2018]. We use positional encodings of the noise level ($t \in [0,1]$) and process it using a transformer style positional embedding:

$$\mathbf{e}(t) = \left[\ldots, \cos\left(tc^{\frac{-2d}{D}}\right), \sin\left(tc^{\frac{-2d}{D}}\right), \ldots\right]^{\mathrm{T}}, \tag{7.2.9}$$

where $d = 1, \ldots, D/2$, D is the number of dimensions of the embedding, and $c = 10000$. This embedding vector is passed through a fully connected layer, followed by an activation function and another fully connected layer. Each residual block has an fully connected layer that adapts the embedding to the correct dimensionality.

To provide $\mathbf{x}_t$, $\mathbf{p}$, and $\mathbf{f}$ to the network, we separately concatenate the past/future frames and the noisy current frames in the channel dimension. The concatenated noisy current frames are directly passed as input to the network. Meanwhile, the concatenated conditional frames are passed through an embedding that influences the conditional normalization akin to SPatially-Adaptive (DE)normalization (SPADE) [Park et al., 2019] to account for the effect of time/motion, we call this approach SPAce-TIme-Adaptive Normalization (SPATIN).

We also try passing the direct concatenation of the conditional and noisy current frames as the input. In our experiments below, we show some results with SPATIN and some with concatenation (concat). For simple video prediction with Equation 7.2.6, we experimented with 3D convolutions and 3D attention. However, this requires an exorbitant amount of

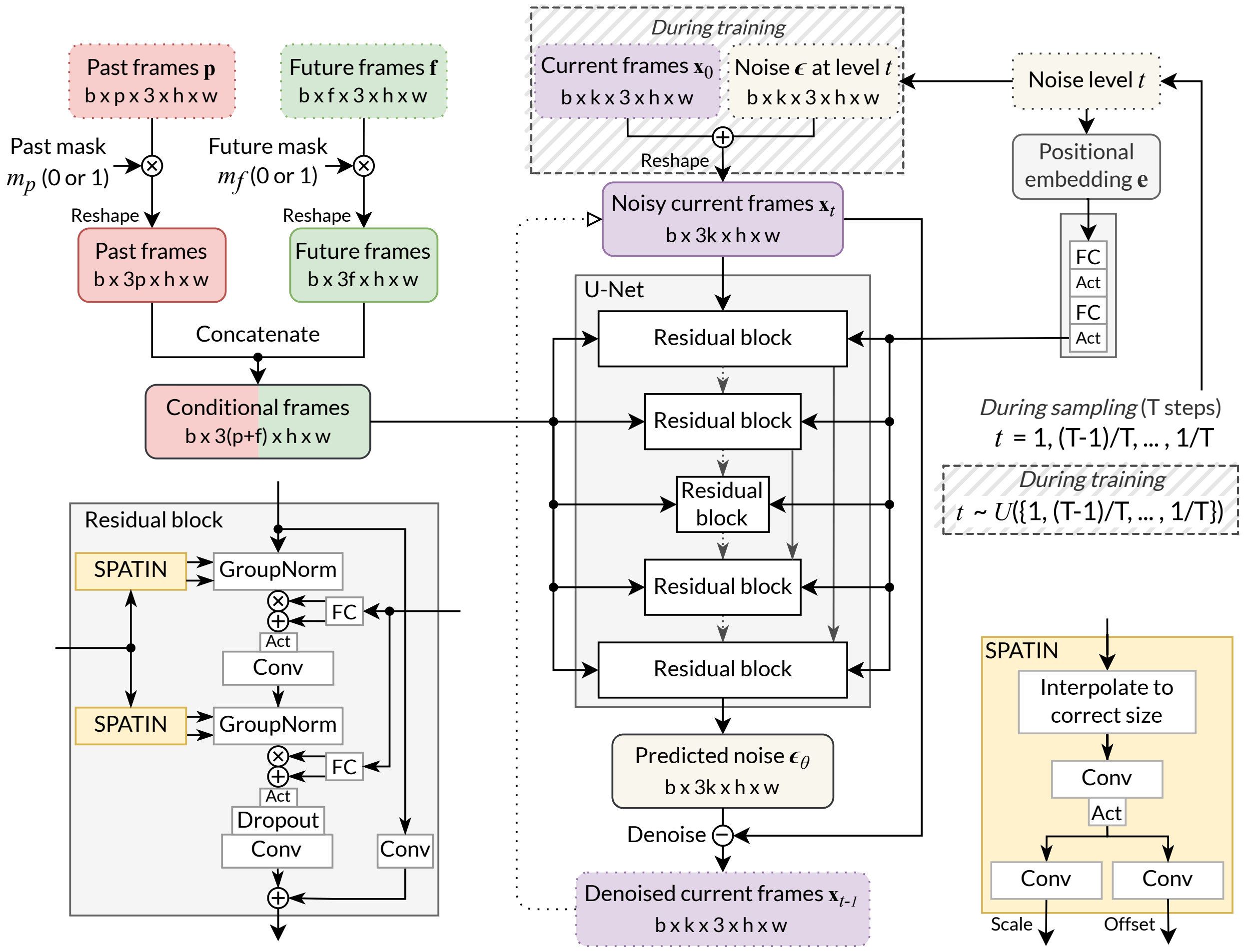

During training
Current frames $\mathbf{x}_0$ b x k x 3 x h x w
Noise ϵ at level t b x k x 3 x h x w
Noise level t
Positional embedding $\mathbf{e}$
FC
Act
FC
Act
Past frames $\mathbf{p}$ b x p x 3 x h x w
Future frames $\mathbf{f}$ b x f x 3 x h x w
Past mask m_p (0 or 1)
Future mask m_f (0 or 1)
Reshape
Past frames b x 3p x h x w
Future frames b x 3f x h x w
Concatenate
Conditional frames b x 3(p+f) x h x w
Noisy current frames $\mathbf{x}_t$ b x 3k x h x w
U-Net
Residual block
Predicted noise ϵ_θ b x 3k x h x w
Denoise
Denoised current frames $\mathbf{x}_{t-1}$ b x k x 3 x h x w
During sampling (T steps) t = 1, (T-1)/T, ... , 1/T
During training $t \sim U(\{1, (T-1)/T, ... , 1/T\})$
SPATIN
GroupNorm
Conv
Dropout
Interpolate to correct size
Scale
Offset

Fig. 7.4. Noisy current frames are given to a U-Net whose residual blocks receive conditional information from past/future frames and noise-level. The output is the predicted noise in the current frames, which is used to denoise the current frames.

memory, and we found no benefit in using 3D layers over 2D layers at the same memory (i.e. the biggest model that fits in 4 GPUs). We also tried and found no benefit from gamma noise [Nachmani et al., 2021], L1 loss, and F-PNDM sampling [Liu et al., 2022].

7.3 Related work

Score-based diffusion models have been used for image editing [Meng et al., 2022, Saharia et al., 2021, Nichol et al., 2021] and our approach to video generation might be viewed as an analogy to classical image inpainting, but in the temporal dimension. The GLIDE or Guided Language to Image Diffusion for Generation and Editing approach of Nichol et al. [2021] uses CLIP-guided diffusion for image editing, while Denoising Diffusion Restoration Models (DDRM) Kawar et al. [2022] additionally condition on a corrupted image to restore the clean image. Adversarial variants of score-based diffusion models have been used to enhance quality [Jolicoeur-Martineau et al., 2021b] or speed [Xiao et al., 2022].

Contemporary work to our own such as that of Ho et al. [2022b] and Yang et al. [2022] also examine video generation using score-based diffusion models. However, the Video Diffusion Models (VDMs) work of Ho et al. [2022b] approximates conditional distributions using a gradient method for conditional sampling from their unconditional model formulation. In contrast, our approach directly works with a conditional diffusion model, which we obtain through masked conditional training, thereby giving us the exact conditional distribution as well as the ability to generate unconditionally. Their experiments focus on: a) unconditional video generation, and b) text-conditioned video generation, whereas our work focuses primarily on predicting future video frames from the past, using our masked conditional generation framework. The Residual Video Diffusion (RVD) of Yang et al. [2022] is only for video prediction, and it uses a residual formulation to generate frames autoregressively one at a time. Meanwhile, ours directly models the conditional frames to generate multiple frames in a block-wise autoregressive manner.

Recurrent neural network (RNN) techniques were early candidates for modern deep neural architectures for video prediction and generation. Early work combined RNNs with a stochastic latent variable (SV2P) Babaeizadeh et al. [2018] and was optimized by variational inference. The stochastic video generation (SVG) approach of Denton and Fergus [2018] learned both prior and a per time step latent variable model, which influences the dynamics of an LSTM at each step. The model is also trained in a manner similar to a variational autoencoder, i.e., it was another form of variational RNN (vRNN). To address the fact that vRNNs tend to lead to blurry results, Castrejón et al. [2019] (Hier-vRNN) increased the expressiveness of the latent distributions using a hierarchy of latent variables. We compare qualitative result of SVG and Hier-vRNN with the concat variant of our MCVD method in Figure 7.5. Other vRNN-based models include SAVP Lee et al. [2018a], SRVP Franceschi et al. [2020], SLAMP Akan et al. [2021].

The well known Transformer paradigm [Vaswani et al., 2017] from natural language processing has also been explored for video. The Video-GPT work of Yan et al. [2021] applied an autoregressive GPT style [Brown et al., 2020] transformer to the codes produced from a VQ-VAE [Van Den Oord et al., 2017]. The Video Transformer work of Weissenborn et al. [2019] models video using 3-D spatio-temporal volumes without linearizing positions in the volume. They examine local self-attention over small non-overlapping sub-volumes or 3D blocks. This is done partly to accelerate computations on TPU hardware. Their work also observed that the peak signal-to-noise ratio (PSNR) metric and the mean-structural similarity (SSIM) metrics [Wang et al., 2004] were developed for images, and have serious flaws when applied to videos. PSNR prefers blurry videos and SSIM does not correlate well to perceptual quality. Like them, we focus on the recently proposed Frechet Video Distance (FVD) [Unterthiner et al., 2018], computed over entire videos and which is sensitive to visual quality, temporal coherence, and diversity of samples. Rakhimov et al. [2020] (LVT) used

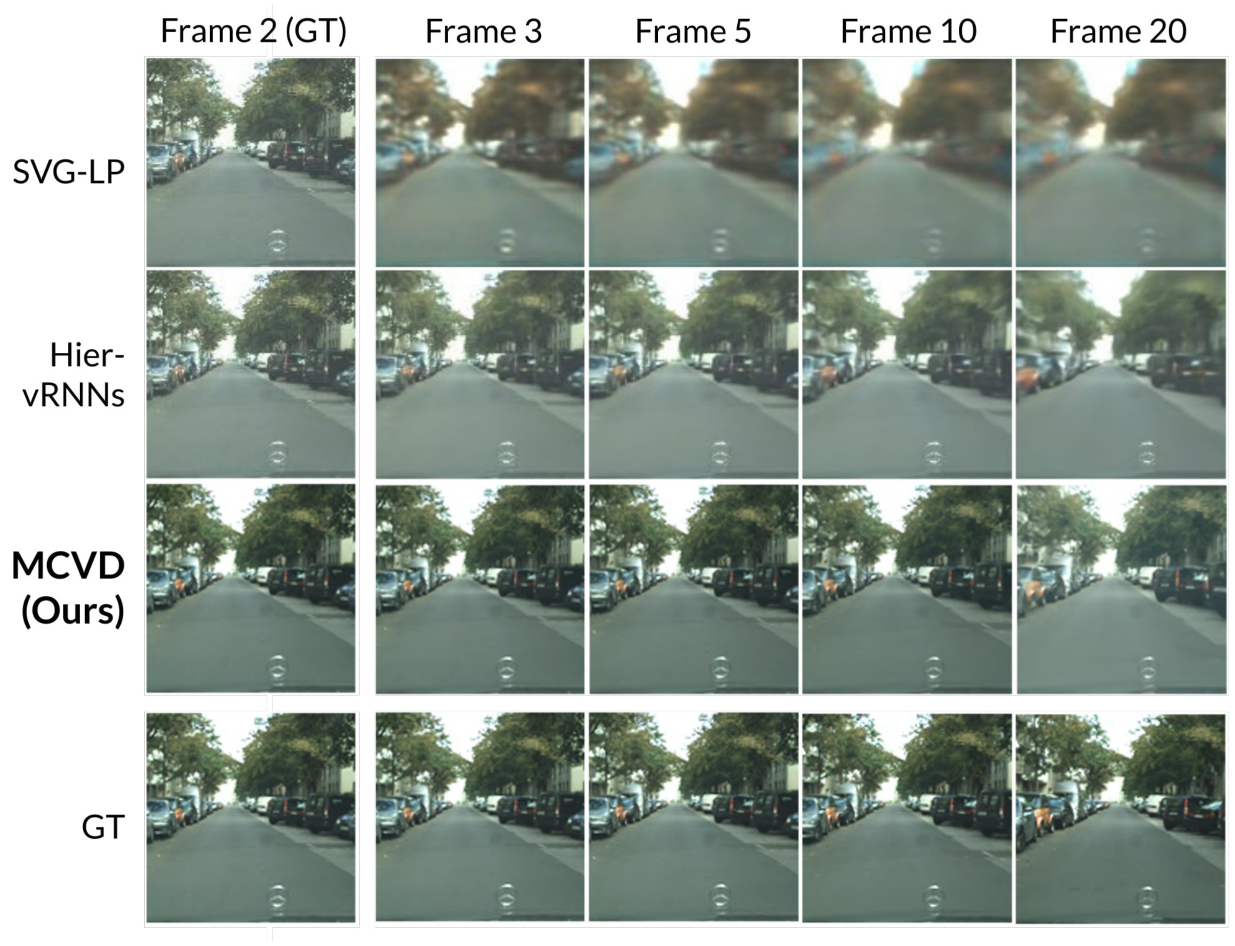


Fig. 7.5. Comparing future prediction methods on Cityscapes: SVG-LP (Top Row), Hier-vRNNs (Second Row), Our Method (Third Row), Ground Truth (Bottom Row). Frame 2, a ground truth conditioning frame is shown in first column, followed by frames: 3, 5, 10 and 20 generated by each method vs the ground truth at the bottom.

transformers to predict the dynamics of video in latent space. Le Moing et al. [2021] (CCVS) also predict in latent space, that of an adversarially trained autoencoder, and also add a learnable optical flow module.

Generative Adversarial Network (GAN) based approaches to video generation have also been studied extensively. Vondrick et al. [2016] proposed an early GAN architecture for video, using a spatio-temporal CNN. Villegas et al. [2017] proposed a strategy for separating motion and content into different pathways of a convolutional LSTM based encoder-decoder RNN. Saito et al. [2017] (TGAN) predicted a sequence of latents using a temporal generator, and then the sequence of frames from those latents using an image generator. TGANv2 Saito et al. [2020] improved its memory efficiency. MoCoGAN Tulyakov et al. [2018] explored style and content separation, but within a CNN framework. Yushchenko et al. [2019] used the MoCoGAN framework by re-formulating the video prediction problem as a Markov Decision Process (MDP). FutureGAN Aigner and Körner [2018] used spatio-temporal 3D convolutions in an encoder decoder architecture, and elements of the progressive GAN Karras et al. [2018] approach to improve image quality. TS-GAN Munoz et al. [2021] facilitated information flow between consecutive frames. TriVD-GAN Luc et al. [2020] proposes a novel recurrent unit in the generator to handle more complex dynamics, while DIGAN Yu et al. [2022] uses implicit neural representations in the generator.

Video interpolation was the subject of a flurry of interest in the deep learning community a number of years ago [Niklaus et al., 2017, Jiang et al., 2018, Xue et al., 2019, Bao et al., 2019]. However, these architectures tend to be fairly specialized to the interpolation task, involving optical flow or motion field modelling and computations. Frame interpolation is useful for video compression; therefore, many other lines of work have examined interpolation from a compression perspective. However, these architectures tend to be extremely specialized to the video compression task [Yang et al., 2020].

The Cutout approach of DeVries and Taylor [2017] has examined the idea of cutting out small continuous regions of an input image, such as small squares. Dropout [Srivastava et al., 2014] at the FeatureMap level was proposed and explored under the name of SpatialDropout in Tompson et al. [2015]. Input Dropout [de Blois et al., 2020] has been examined in the context of dropping different channels of multi-modal input imagery, such as the dropping of the RGB channels or depth map channels during training, then using the model without one of the modalities during testing, e.g. in their work they drop the depth channel.

Regarding our block-autoregressive approach, previous video prediction models were typically either 1) non-recurrent: predicting all n frames simultaneously with no way of adding more frames (most GAN-based methods), or 2) recurrent in nature, predicting 1 frame at a time in an autoregressive fashion. The benefit of the non-recurrent type is that you can generate videos faster than 1 frame at a time while allowing for generating as many frames as needed. The disadvantage is that it is slower than generating all frames at once, and takes up more memory and compute at each iteration. Our model finds a sweet spot in between in that it is block-autoregressive: generating $k < n$ frames at a time recurrently to finally obtain n frames.

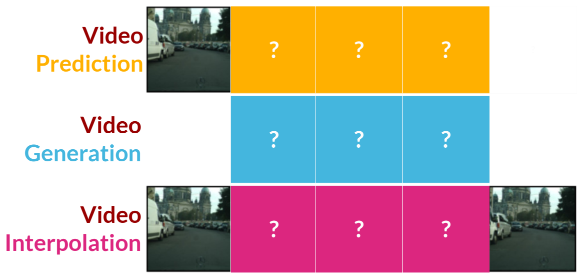
7.4 Experiments

7.4.1 Tasks

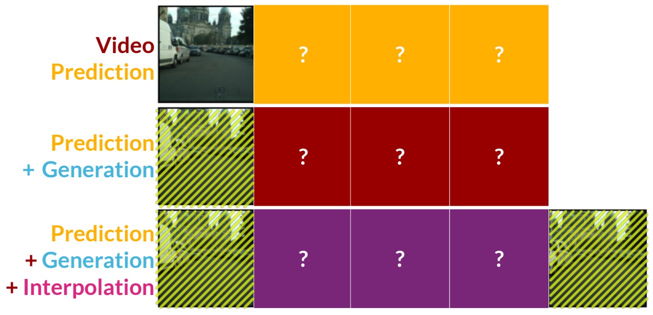
We show state-of-the-art results on three video tasks:

(1) **Video prediction** i.e. prediction of future frames conditioned on past frames,
(2) **Video generation** i.e. unconditional generation of video frames, and
(3) **Video interpolation** i.e. prediction of intermediate frames conditioned on past and future frames

These tasks and our proposed solutions are visualized in Figure 7.6.



(a) Tasks



(b) Masked solutions (green is mask)

Fig. 7.6. Visualization of video (a) tasks and (b) our proposed masked solutions.

7.4.2 Datasets

Our choice of datasets is in order of progressive difficulty: 1) SMMNIST: black-and-white digits; 2) KTH: grayscale single-humans; 3) BAIR: color, multiple objects, simple scene; 4) Cityscapes: color, natural complex natural driving scene; 5) UCF101: color, 101 categories of natural scenes. We process these datasets similarly to prior works. For Cityscapes, each video is center-cropped, then resized to 128×128. For UCF101, each video clip is center-cropped at 240×240 and resized to 64×64, taking care to maintain the train-test splits. We generate 128x128 images for Cityscapes and 64x64 images for the other datasets.

(1) **Video prediction** : We show the results of our video prediction experiments on test data that was never seen during training in Tables 7.3 - 7.5 for Cityscapes [Cordts et al., 2016], Stochastic Moving MNIST (SMMNIST) [Denton and Fergus, 2018, Srivastava et al., 2015], KTH [Schuldt et al., 2004] and BAIR [Ebert et al., 2017] respectively.

(2) **Video generation** : We present unconditional generation results for BAIR in Table 7.6 and UCF-101 [Soomro et al., 2012] in Table 7.7.

(3) **Video interpolation** : We show interpolation results for SMMNIST, KTH, and BAIR in Table 7.8.

7.4.3 Training details

Unless otherwise specified, we set the mask probability to 0.5 when masking was used. For sampling, we report results using the sampling methods DDPM [Ho et al., 2020] or DDIM [Song et al., 2021a] with only 100 sampling steps, though our models were trained with 1000, to make sampling faster. We observe that the metrics are generally better using DDPM than DDIM (except for UCF-101). Using 1000 sampling steps could yield better results.

Note that all our models are trained to predict only 4-5 current frames at a time, unlike other models that predict $\geq$10. We use these models to then autoregressively predict longer sequences for prediction or generation. This was done in order to fit the models in our GPU memory budget. Despite this disadvantage, we find that our MCVD models perform better than many previous SOTA methods.

We provide some additional information regarding model size, memory requirements, batch size and computation times in Table 7.1. This is followed by additional results and visualizations for SMMNIST, KTH, BAIR, UCF-101 and Cityscapes.

Table 7.1. Compute used. "steps" indicates the checkpoint with the best approximate FVD, "GPU hours" is the total training time up to "steps".

Dataset, model	params	CPU mem (GB)	batch size	GPU	GPU mem (GB)	steps	GPU hours
SMMNIST concat	27.9M	3.6	64	Tesla V100	14.5	700000	78.9
SMMNIST spatin	53.9M	3.3	64	RTX 8000	23.4	140000	39.7
KTH concat	62.8M	3.2	64	Tesla V100	21.5	400000	65.7
KTH spatin	367.6M	8.9	64	A100	145.9	340000	45.8
BAIR concat	251.2M	5.1	64	Tesla V100	76.5	450000	78.2
BAIR spatin	328.6M	9.2	64	A100	86.1	390000	50.0
Cityscapes concat	262.1M	6.2	64	Tesla V100	78.2	900000	192.83
Cityscapes spatin	579.1M	8.9	64	A100	101.2	650000	96.0
UCF concat	565.0M	8.9	64	Tesla V100	100.1	900000	183.95
UCF spatin	739.4M	8.9	64	A100	115.2	550000	79.5

7.4.4 Metrics

As mentioned earlier, we primarily use the FVD metric for comparison across models as FVD measures both fidelity and diversity of the generated samples. Previous works compare Frechet Inception Distance (FID) [Heusel et al., 2017] and Inception Score (IS) [Salimans et al., 2016], adapted to videos by replacing the Inception network with a 3D-convolutional network that takes video input. FVD is computed similarly to FID, but using an I3D network trained on the huge video dataset Kinetics-400. We also report PSNR and SSIM.

We tried to add the older FID and IS metrics (as opposed to the newer FVD metric which we used above) for UCF-101 as proposed in Saito et al. [2017], but we had difficulties integrating the chainer [Tokui et al., 2019] based implementation of these metrics into our PyTorch [Paszke et al., 2019] code base.

7.5 Results

Our MCVD concat past-future-mask and past-mask results are of particular interest as they yield SOTA results across many benchmark configurations.

7.5.1 Video prediction

Table 7.2. Video prediction on Cityscapes (128 × 128), 2 past frames, predicting 28.

Cityscapes (128 × 128) [2 → 28; trained on *k*]	*k*	FVD↓	LPIPS↓	SSIM↑
SVG-LP [Denton and Fergus, 2018]	10	1300.26	0.549 ± 0.06	0.574 ± 0.08
vRNN 1L [Castrejón et al., 2019]	10	682.08	0.304 ± 0.10	0.609 ± 0.11
Hier-vRNN [Castrejón et al., 2019]	10	567.51	0.264 ± 0.07	0.628 ± 0.10
GHVAE [Wu et al., 2021]	10	418.00	0.193 ± 0.014	**0.740** ± 0.04
MCVD spatin past-mask (Ours)	**5**	184.81	0.121 ± 0.05	0.720 ± 0.11
MCVD concat past-mask (Ours)	**5**	**141.31**	**0.112** ± 0.05	0.690 ± 0.12

Table 7.3. Video prediction results on SMMNIST (64 × 64) for 10 predicted frames conditioned on 5 past frames. We predicted 10 trajectories per real video, and report the average FVD and maximum SSIM, averaged across 256 test videos.

SMMNIST [5 → 10; trained on *k*]	*k*	**FVD↓**	**SSIM↑**
SVG [Denton and Fergus, 2018]	10	90.81	0.688
vRNN 1L [Castrejón et al., 2019]	10	63.81	0.763
Hier-vRNN [Castrejón et al., 2019]	10	57.17	0.760
MCVD concat (Ours)	**5**	25.63	**0.786**
MCVD spatin (Ours)	**5**	**23.86**	0.780

Table 7.4. Video prediction results on KTH (64 × 64), predicting 30/40 frames from 10 past frames using models trained to predict *k* frames at a time. All models test on 256 videos.

KTH [10 → *pred*; trained on *k*]	*k*	*pred*	FVD↓	PSNR↑	SSIM↑
SV2P [Babaeizadeh et al., 2018]	10	30	636 ± 1	28.2	0.838
SVG-LP [Denton and Fergus, 2018]	10	30	377 ± 6	28.1	0.844
SAVP [Lee et al., 2018a]	10	30	374 ± 3	26.5	0.756
MCVD spatin (Ours)	**5**	30	323 ± 3	27.5	0.835
MCVD concat past-future-mask (Ours)	**5**	30	294.9	24.3	0.746
SLAMP [Akan et al., 2021]	10	30	228 ± 5	29.4	0.865
SRVP [Franceschi et al., 2020]	10	30	222 ± 3	29.7	0.870
Struct-vRNN [Minderer et al., 2019]	10	40	395.0	24.29	0.766
MCVD concat past-future-mask (Ours)	**5**	40	368.4	23.48	0.720
MCVD spatin (Ours)	**5**	40	331.6 ± 5	26.40	0.744
MCVD concat (Ours)	**5**	40	276.6 ± 3	26.20	0.793
SV2P time-invariant [Babaeizadeh et al., 2018]	10	40	253.5	25.70	0.772
SV2P time-variant [Babaeizadeh et al., 2018]	10	40	209.5	25.87	0.782
SAVP [Lee et al., 2018a]	10	40	183.7	23.79	0.699
SVG-LP [Denton and Fergus, 2018]	10	40	157.9	23.91	0.800
SAVP-VAE [Lee et al., 2018a]	10	40	145.7	26.00	0.806
Grid-keypoints [Gao et al., 2021]	10	40	144.2	27.11	0.837

Table 7.5. Video prediction results on BAIR (64×64) conditioning on p past frames and predicting *pred* frames in the future, using models trained to predict k frames at at time.

BAIR [past (p) → pred (pr) ; trained on k]	p	k	pr	FVD↓	PSNR↑	SSIM↑
LVT [Rakhimov et al., 2020]	1	15	15	125.8	–	–
DVD-GAN-FP [Clark et al., 2019]	1	15	15	109.8	–	–
MCVD spatin (Ours)	1	**5**	15	103.8	18.8	0.826
TrIVD-GAN-FP [Luc et al., 2020]	1	15	15	103.3	–	–
VideoGPT [Yan et al., 2021]	1	15	15	103.3	–	–
CCVS [Le Moing et al., 2021]	1	15	15	99.0	–	–
MCVD concat (Ours)	1	**5**	15	98.8	18.8	0.829
MCVD spatin past-mask (Ours)	1	**5**	15	96.5	18.8	0.828
MCVD concat past-mask (Ours)	1	**5**	15	95.6	18.8	0.832
Video Transformer [Weissenborn et al., 2019]	1	15	15	94-96[a]	–	–
FitVid [Babaeizadeh et al., 2021]	1	15	15	93.6	–	–
MCVD concat past-future-mask (Ours)	1	**5**	15	**89.5**	16.9	0.780
SAVP [Lee et al., 2018a]	2	14	14	116.4	–	–
MCVD spatin (Ours)	2	**5**	14	94.1	19.1	0.836
MCVD spatin past-mask (Ours)	2	**5**	14	90.5	19.2	0.837
MCVD concat (Ours)	2	**5**	14	90.5	19.1	0.834
MCVD concat past-future-mask (Ours)	2	**5**	14	89.6	17.1	0.787
MCVD concat past-mask (Ours)	2	**5**	14	**87.9**	19.1	0.838
SVG-LP [Akan et al., 2021]	2	10	28	256.6	–	0.816
SVG [Akan et al., 2021].	2	12	28	255.0	18.95	0.8058
SLAMP [Akan et al., 2021]	2	10	28	245.0	19.7	0.818
SRVP [Franceschi et al., 2020]	2	12	28	162.0	19.6	0.820
WAM [Jin et al., 2020]	2	14	28	159.6	21.0	0.844
SAVP [Lee et al., 2018a]	2	12	28	152.0	18.44	0.7887
vRNN 1L Castrejón et al. [2019]	2	10	28	149.2	–	0.829
SAVP [Lee et al., 2018a]	2	10	28	143.4	–	0.795
Hier-vRNN [Castrejón et al., 2019]	2	10	28	143.4	–	0.822
MCVD spatin (Ours)	2	**5**	28	132.1	17.5	0.779
MCVD spatin past-mask (Ours)	2	**5**	28	127.9	17.7	0.789
MCVD concat (Ours)	2	**5**	28	120.6	17.6	0.785
MCVD concat past-mask (Ours)	2	**5**	28	119.0	17.7	0.797
MCVD concat past-future-mask (Ours)	2	**5**	28	**118.4**	16.2	0.745

[a] 94 on only the first frames, 96 on all subsequences of test frames

7.5.2 Video generation

Table 7.6. Unconditional generation of BAIR video frames.

BAIR (64 × 64) [0 → *pred*; trained on 5]	*pred*	FVD↓
MCVD spatin past-mask (Ours)	16	267.8
MCVD concat past-mask (Ours)	16	**228.5**
MCVD spatin past-mask (Ours)	30	399.8
MCVD concat past-mask (Ours)	30	**348.2**

Table 7.7. Unconditional generation of UCF-101 video frames.

UCF-101 (64 × 64) [0 → 16; trained on *k*]	*k*	FVD↓
MoCoGAN-MDP [Yushchenko et al., 2019]	16	1277.0
MCVD concat past-mask (Ours)	**4**	1228.3
TGANv2 [Saito et al., 2020]	16	1209.0
MCVD spatin past-mask (Ours)	**4**	1143.0
DIGAN [Yu et al., 2022]	16	**655.0**

7.5.3 Video interpolation

Table 7.8. Video Interpolation results (64 × 64). Given p past + f future frames → interpolate k frames. Reporting average of the best metrics out of n trajectories per test sample. $\downarrow(p+f)$ and $\uparrow k$ is harder. We used MCVD spatin past-mask for SMMNIST and KTH, and MCVD concat past-future-mask for BAIR. We also include results on SMMNIST for a "pure" model trained without any masking.

	SMMNIST (64 × 64)					**KTH** (64 × 64)					**BAIR** (64 × 64)				
	p+f	k	n	PSNR↑	SSIM↑	p+f	k	n	PSNR↑	SSIM↑	p+f	k	n	PSNR↑	SSIM↑
SVG-LP	18	7	100	13.543	0.741	18	7	100	28.131	0.883	18	7	100	18.648	0.846
FSTN	18	7	100	14.730	0.765	18	7	100	29.431	0.899	18	7	100	19.908	0.850
SepConv	18	7	100	14.759	0.775	18	7	100	29.210	0.904	18	7	100	21.615	0.877
SuperSloMo	18	7	100	13.387	0.749	18	7	100	28.756	0.893	–	–	–	–	–
SDVI full	18	7	100	16.025	0.842	18	7	100	29.190	0.901	18	7	100	21.432	0.880
SDVI	16	7	100	14.857	0.782	16	7	100	26.907	0.831	16	7	100	19.694	0.852
	10	**10**	100	20.944	0.854	**15**	**10**	100	34.669	0.943	**4**	**5**	100	25.162	0.932
MCVD (Ours)	**10**	**5**	**10**	27.693	0.941	**15**	**10**	**10**	34.068	0.942	**4**	**5**	**10**	23.408	0.914
		pure		18.385	0.802	**10**	**5**	**10**	35.611	0.963					

We compare our MCVD method with previous methods of video interpolation on the standard datasets of SMMNIST, KTH, and BAIR: SVG-LP Denton and Fergus [2018], FSTN Lu et al. [2017], SepConv Niklaus et al. [2017], SuperSloMo Jiang et al. [2018], and SDVI Xu et al. [2020].

7.5.4 Ablation studies

We were able to draw the following conclusions from ablation studies, expanded upon below:

- A model trained on multiple tasks performs better than one trained on individual tasks. This shows that solving tasks like interpolation helps in solving more complex tasks like prediction and generation.
- In general, the concat variant performs better than the spatin variant.

In Table 7.5 we compare models that use concatenated raw pixels as input to U-Net blocks (concat) to SPATIN variants. We also compare no-masking to past-masking variants, i.e. models which are only trained predict the future vs. models which are regularized by being trained for prediction and unconditional generation. It can be seen that our model works across different choices of past frames and generates better quality for shorter videos. This is expected from models of this kind. Moreover, it can be seen that the model trained on the two tasks of Prediction and Generation (i.e., the models with past-mask) performs better than the model trained only on Prediction!

In Table 7.9, we provide results for an ablation study using SMMNIST on the different design choices: concat vs concat past-future-mask vs spatin vs spatin future-mask vs spatin past-future-mask. In Figure 7.7 we provide some visual results for SMMNIST.

Table 7.9. Results on the SMMNIST evaluation, conditioned on 5 past frames, predicting 10 frames using models trained to predict 5 frames at a time.

SMMNIST [$5 \rightarrow 10$; trained on 5]	FVD↓	PSNR↑	SSIM↑	LPIPS↓	MSE↓
MCVD spatin future-mask	44.14 ± 1.73	16.31	0.758	0.141	0.027
MCVD spatin past-future-mask	36.12 ± 0.63	16.15	0.748	0.146	0.027
MCVD concat	25.63 ± 0.69	17.22	0.786	0.117	0.024
MCVD spatin	23.86 ± 0.67	17.07	0.785	0.129	0.025
MCVD concat past-future-mask	**20.77** ± 0.77	16.33	0.753	0.139	0.028

It can be seen that concat is, in general, better than spatin. It can also be seen that the past-future-mask variant, which is a general model capable of all three tasks, performs better at the individual tasks than the models trained only on the individual task. This was demonstrated in Table 7.5 as well. This shows that the model gains helpful insights while generalizing to all three tasks, which it does not while training only on the individual task.

We conducted preliminary experiments with a larger number of frames. Since the models with a larger number of frames were bigger, we could only run them for a shorter time with a smaller batch size than the smaller models. In general, we found that larger models did not substantially improve the results. We attribute this to the fact that using more frames means that the model should be given more capacity, but we could not increase it due to our computational budget constraints. We emphasize that our method works very well with fewer computational resources.

Examining these results we remark that we have SOTA performance for prediction on SMMNIST, BAIR and the challenging Cityscapes evaluation. Our Cityscapes model yields an FVD of 145.5, whereas the best previous result of which we are aware is 418. The quality of our Cityscapes results are illustrated visually in Figure 7.1 and Figure 7.2. While our completely unconditional generation results are strong, we note that when past masking is used to regularize future predicting models, we see clear performance gains in Table 7.5. Finally, in Table 7.8 we see that our interpolation results are SOTA by a wide margin, across experiments on SMMNIST, KTH and BAIR – even compared to architectures much more specialized for interpolation.

It can be seen that our proposed method generates better quality videos, even though it was trained on a shorter number of frames than other methods. It can also be seen that training on multiple tasks using random masking improves the quality of generated frames than training on the individual tasks.

7.6 Conclusion

We have shown how to obtain SOTA video prediction and interpolation results with randomly masked conditional video diffusion models using a relatively simple architecture. We found that past-masking was able to improve performance across all model variants and configurations tested. We believe our approach may pave the way forward toward high quality larger-scale video generation.

7.6.1 Limitations

Videos generated by these models are still small compared to real movies, and they can still become blurry or inconsistent when the number of generated frames is very large. Our unconditional generation results on the highly diverse UCF-101 dataset are still far from perfect. More work is clearly needed to scale these models to larger datasets with more diversity and with longer duration video. As has been the case in many other settings, simply using larger models with many more parameters is a strategy that is likely to improve the quality and flexibility of these models – we were limited to 4 GPUs for our work here. There is also a need for faster sampling methods capable of maintaining quality over time.

Given our strong interpolation results, conditional diffusion models which generate skipped frames could make it possible to generate much longer, but consistent video through a strategy of first generating sparse distant frames in a block, followed by an interpolative diffusion step for the missing frames.

7.6.2 Broader Impacts

High-quality video generation is potentially a powerful technology that could be used by malicious actors for applications such as creating fake video content. Our formulation focuses on capturing the distributions of real video sequences. High-quality video prediction could one day find use in applications such as autonomous vehicles, where the cost of errors could be high. Diffusion methods have shown great promise for covering the modes of real probability distributions. In this context, diffusion-based techniques for generative modelling may be a promising avenue for future research where the ability to capture modes properly is safety critical. Another potential point of impact is the amount of computational resources being spent for these applications involving the high fidelity and voluminous modality of video data. We emphasize the use of limited resources in achieving better or comparable results. Our submission provides evidence for more efficient computation involving fewer GPU hours spent in training time.

7.7 Qualitative results

Below are prediction/generation examples from Stochastic Moving MNIST, KTH, BAIR, UCF-101, and Cityscapes. For more examples including videos, please visit `https://mask-cond-video-diffusion.github.io`

7.7.1 Stochastic Moving MNIST

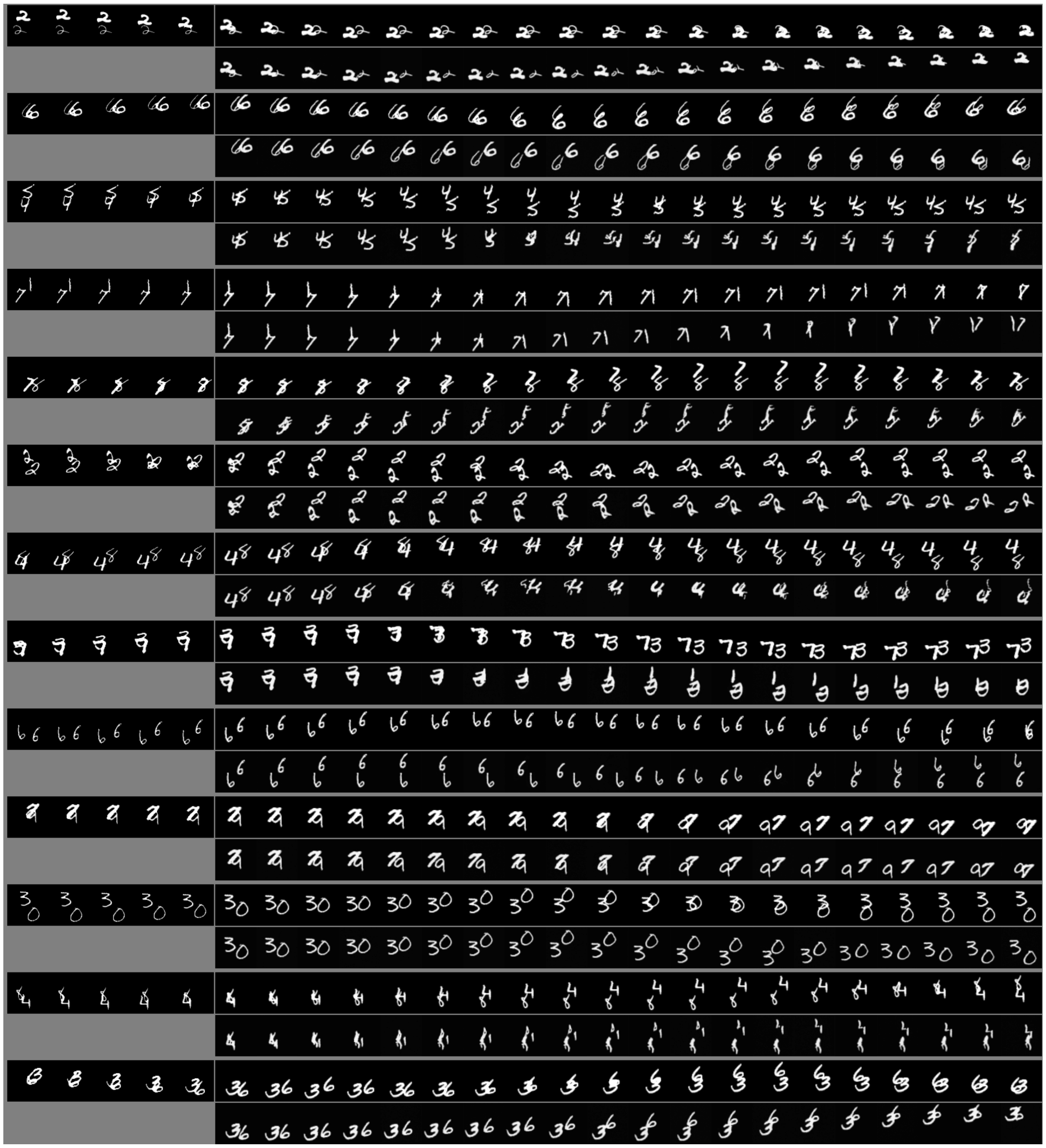


Fig. 7.7. SMMNIST 5 → 10, trained on 5 (prediction). For each sample, top row is real ground truth, bottom is predicted by our MCVD model.

7.7.2 KTH

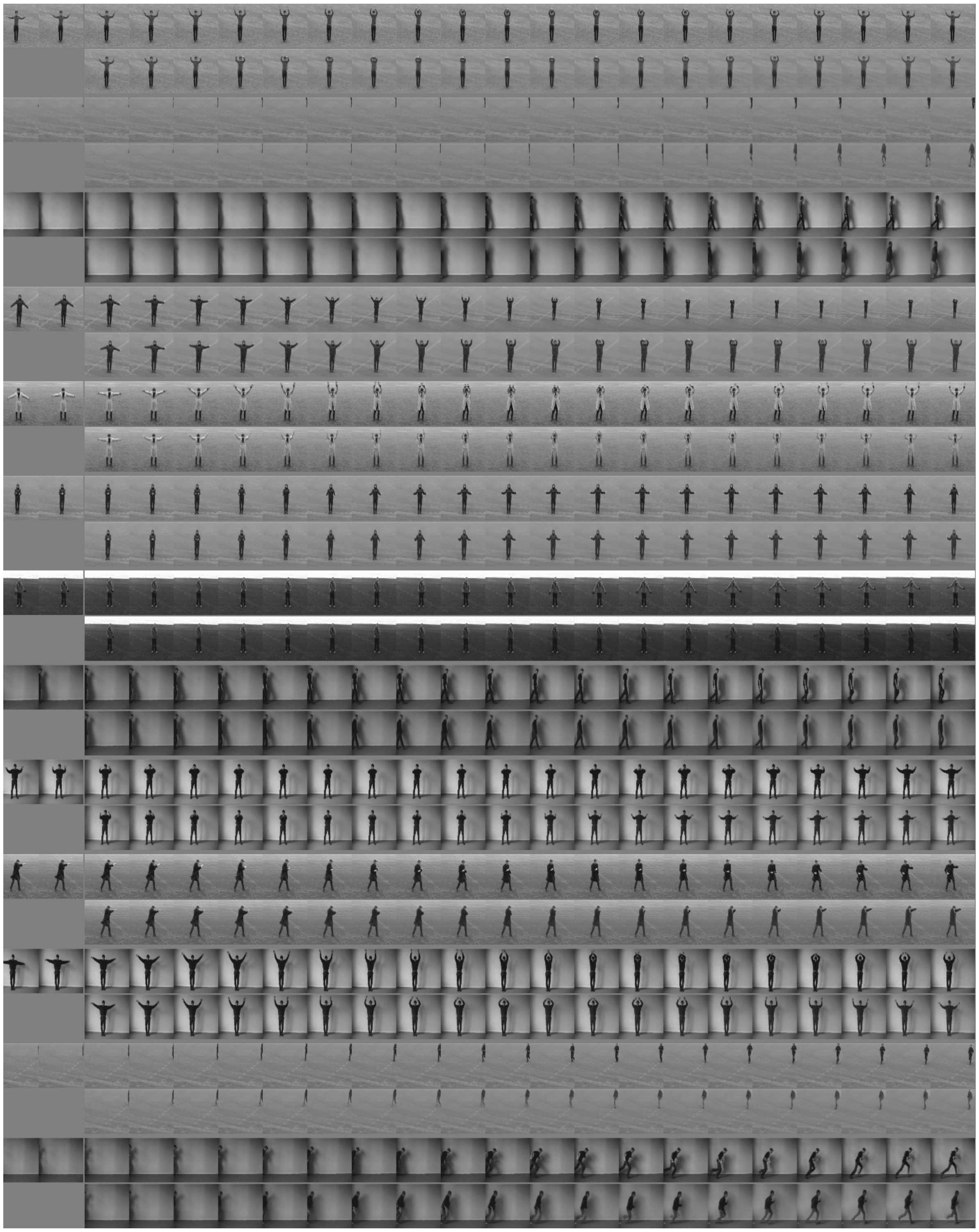


Fig. 7.8. KTH 5 → 20, trained on 5 (prediction). For each sample, top row is real ground truth, bottom is predicted by our MCVD model. (We show only 2 conditional frames here)

7.7.3 BAIR

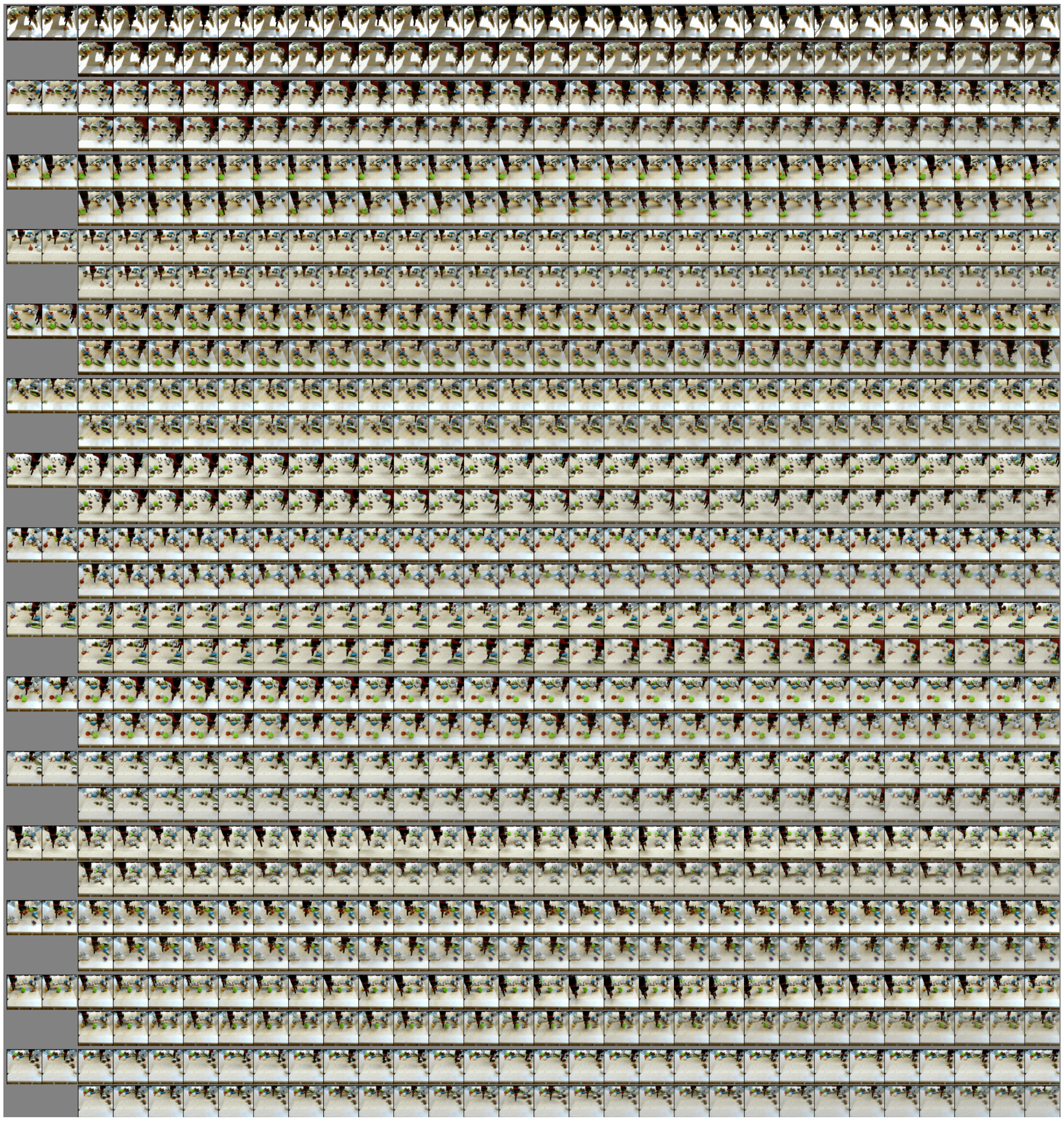


Fig. 7.9. BAIR 2 → 28, trained on 5 (prediction). For each sample, top row is real ground truth, bottom is predicted by our MCVD model.

7.7.4 UCF-101

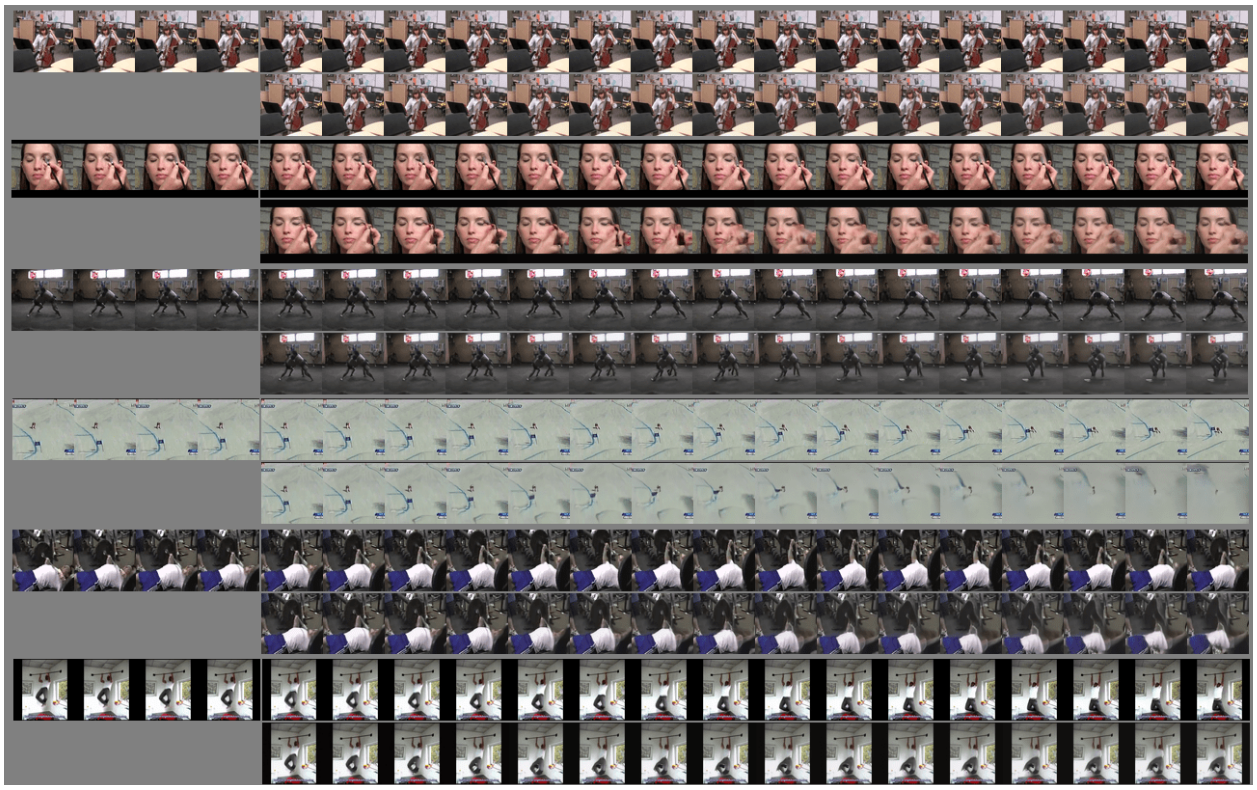


Fig. 7.10. UCF-101 4 → 16, trained on 4 (prediction). For each sample, top row is real ground truth, bottom is predicted by our MCVD model.

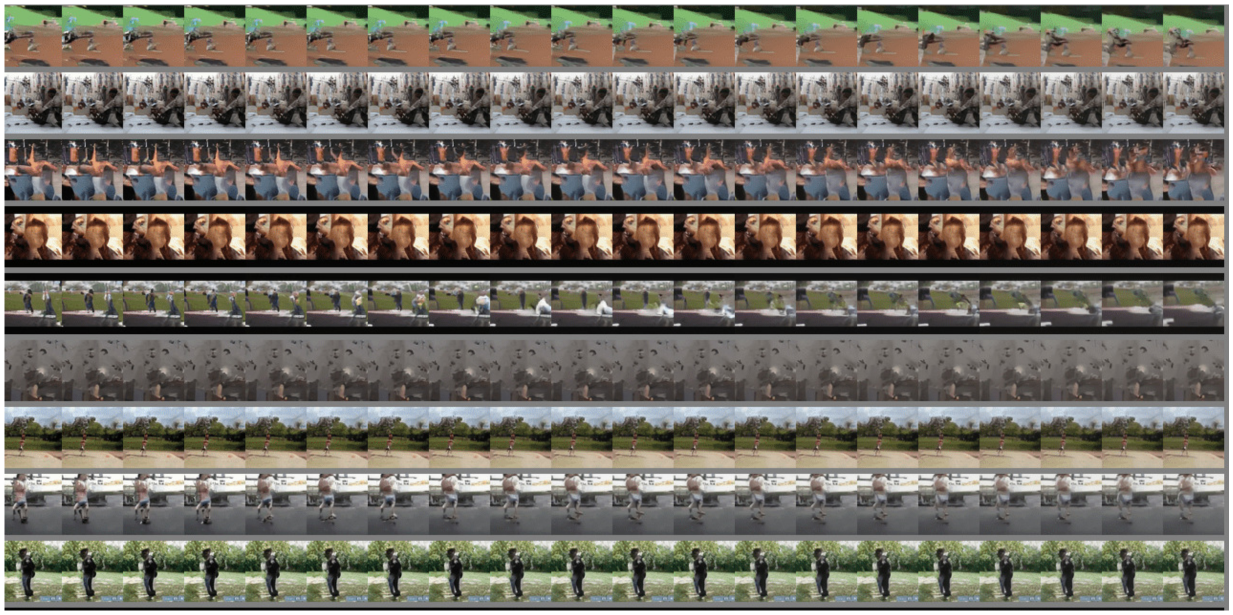


Fig. 7.11. UCF-101 0 → 4 (generation)

7.7.5 Cityscapes

Here we provide some examples of future frame prediction for Cityscapes sequences conditioning on two frames and predicting the next 7 frames.

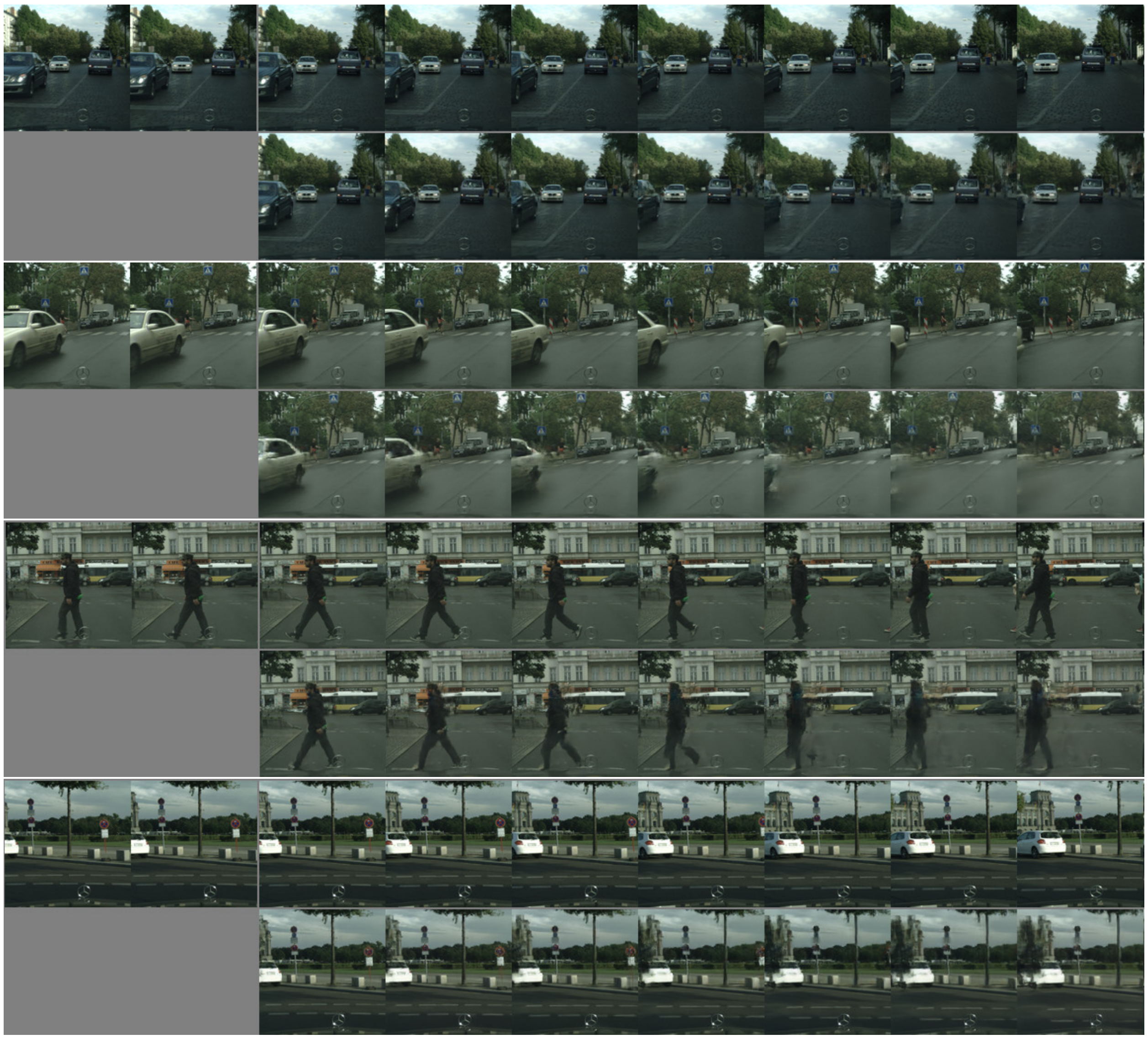


Fig. 7.12. Cityscapes: $2 \rightarrow 7$, trained on 5 (prediction); Conditioning on the two frames in the top left corner of each block of two rows of images, we generate the next 7 frames. The top row is the true frames, bottom row contains the generated frames. We use the MCVD concat model variant.

Chapter 8

Conclusion

The articles presented in this thesis reside at the intersection of generative modeling and computer vision, focusing on inventing novel ideas and combinations of techniques to advance the state of the art in image generation, improved 3D animation, and enhanced video prediction, generation, and interpolation. Our endeavors of the diverse projects within this thesis have embraced a multifaceted notion of improvement, encompassing aspects such as quality enhancement as well as computational efficiency. Leveraging the abundant intellectual (and other) resources at Mila, University of Montreal, we delved into various successful projects, contributing to the advancement of generative modeling in computer vision and paving the way for future exploration and innovation in this exciting field:

(1) **Neural ODEs for video prediction**: novel use of Neural ODEs to model the dynamics of a video [Voleti et al., 2019].
(2) **Multi-Resolution Continuous Normalizing Flows (MRCNF)**: a normalizing flow approach that uses fewer parameters and significantly less time to train than existing works [Voleti et al., 2019].
(3) **Neural inverse kinematics with 3D human pose prior**: integrating the most useful inverse kinematics approach for animators (ProtoRes) with the best 3D human pose prior (SMPL) trained on the biggest 3D human pose dataset (AMASS) [Voleti et al., 2022b].
(4) **Non-isotropic denoising diffusion models**: a novel formulation of denoising diffusion models that expands their capabilities beyond the original formulation [Voleti et al., 2022c].
(5) **Denoising diffusion models for video prediction, generation, interpolation**: novel use of denoising diffusion models to solve all three video tasks simultaneously with the same model [Voleti et al., 2022a].

Towards the end of the last project on video prediction using denoising diffusion models, diffusion-based text-to-image models gained significant traction, followed by text-to-3D object, text-to-video, and text-to-4D. The success of denoising diffusion models has sparked a surge of interest in this area, with many researchers exploring novel ideas and techniques to improve upon existing methods and further advance the field of generative modeling. As the field continues to grow, there are several areas of focus that researchers are likely to explore in the future.

One area of interest is the development of more efficient and scalable generative models that can handle larger datasets and produce more realistic results. Text has emerged as a powerful input modality in driving the generation of images, 3D models, and videos, showcasing significant progress in recent years. Text-to-image models include DALL-E(2) [Ramesh et al., 2021, dal, 2021], Imagen [Saharia et al., 2022], and the open source Stable Diffusion [Rombach et al., 2022]. Text-to-3D models such as DreamFusion Poole et al. [2022], Magic3D [Lin et al., 2023], Shap-E [Jun and Nichol, 2023] leverage text-to-image models to synthesize 3D objects from text. Recent text-to-video models include Make-A-Video by Meta [Singer et al., 2022], ImagenVideo by Google [Ho et al., 2022a], Gen2 by RunwayML [Esser et al., 2023], etc. Make-A-Video3D [Singer et al., 2023], which Vikram had contributed to, enables the generation of 4D scenes from text prompts, utilizing the foundations of text-to-video models.

Advanced techniques of controlling and editing the synthesized results of generative models have grown immensely in the recent past as well. Diffusion-based controllable methods include Textual Inversion [Gal et al., 2022], Dreambooth [Ruiz et al., 2023], Imagic Kawar et al. [2023], ControlNet [Zhang and Agrawala, 2023], InstructPix2Pix [Brooks et al., 2023], etc. Among recent GAN-based methods, DragGAN [Pan et al., 2023] exhibits promising potential in offering unprecedented levels of controllability in image editing using generative models. A recently developed work called DragDiffusion [Shi et al., 2023] showcases this controllability using denoising diffusion models.

Another area of interest in the research community is the integration of generative models with other machine learning techniques, such as reinforcement learning, to enable the creation of more intelligent and interactive synthetic environments. Additionally, there is likely to be increased focus on the ethical implications of generative models, including issues related to bias, privacy, and intellectual property.

In conclusion, the future of generative modeling holds tremendous promise, offering a multitude of exciting possibilities for further research and innovation in the domains of image generation, 3D modeling, and video synthesis. The advancements achieved thus far have propelled the field forward, pushing the boundaries of what is conceivable and laying the foundation for groundbreaking applications. As the field continues to evolve, we can expect breakthroughs that redefine our understanding and capabilities in generating images, 3D models, and videos. The potential applications are vast, ranging from entertainment and creative industries to healthcare, robotics, and beyond.

I hope this thesis has contributed to the understanding and advancement of generative modeling, exploring its applications in computer vision, discussing various techniques, and highlighting the significance of conditional variants. As the field continues to mature, it is essential to embrace new challenges, address limitations, and explore novel directions. The journey towards harnessing the full potential of generative modeling has just begun, and the future holds tremendous opportunities for further research, innovation, and transformative applications in the world of computer vision.

References

Dall-e 2. 2021. URL `https://openai.com/dall-e-2`.

Unity 3D. Retargeting of humanoid animations. `https://docs.unity3d.com/Manual/Retargeting.html`, 2022. Accessed: 2022-08-01.

Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

Sandra Aigner and Marco Körner. Futuregan: Anticipating the future frames of video sequences using spatio-temporal 3d convolutions in progressively growing gans. *arXiv preprint arXiv:1810.01325*, 2018.

Adil Kaan Akan, Erkut Erdem, Aykut Erdem, and Fatma Güney. Slamp: Stochastic latent appearance and motion prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14728–14737, 2021.

Tom M Apostol. *Calculus, volume 1.* John Wiley & Sons, 1991.

Andreas Aristidou, Joan Lasenby, Yiorgos Chrysanthou, and Ariel Shamir. Inverse kinematics techniques in computer graphics: A survey. In *Computer graphics forum*, volume 37, pages 35–58. Wiley Online Library, 2018.

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

Ilze Amanda Auzina, Çağatay Yıldız, Sara Magliacane, Matthias Bethge, and Efstratios Gavves. Invariant neural ordinary differential equations. *arXiv preprint arXiv:2302.13262*, 2023.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. In *International Conference on Learning Representations*, 2018.

Mohammad Babaeizadeh, Mohammad Taghi Saffar, Suraj Nair, Sergey Levine, Chelsea Finn, and Dumitru Erhan. Fitvid: Overfitting in pixel-level video prediction. *arXiv preprint arXiv:2106.13195*, 2021.

Wenbo Bao, Wei-Sheng Lai, Chao Ma, Xiaoyun Zhang, Zhiyong Gao, and Ming-Hsuan Yang. Depth-aware video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3703–3712, 2019.

A. O. Bebko, A. Thaler, and N. F. Troje. bmlsup - a smpl unity player. In *Proc. IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2021.

Christopher Beckham, Sina Honari, Vikas Verma, Alex M Lamb, Farnoosh Ghadiri, R Devon Hjelm, Yoshua Bengio, and Chris Pal. On adversarial mixup resynthesis. In *Advances in neural information processing systems*, pages 4346–4357, 2019.

Jens Behrmann, Will Grathwohl, Ricky TQ Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582, 2019.

Raphael Bensadoun, Shir Gur, Nitsan Blau, Tom Shenkar, and Liorn Wolf. Neural inverse kinematics. In *Proc. ICML*, 2022.

Hugo Berard, Gauthier Gidel, Amjad Almahairi, Pascal Vincent, and Simon Lacoste-Julien. A closer look at the optimization landscapes of generative adversarial networks. In *International Conference on Machine Learning*, 2020.

Nathanaël Berestycki. Introduction to the gaussian free field and liouville quantum gravity. *Lecture notes*, 2015.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

F. Bocquelet, B. Oreshkin, F. Harvey, L-S Ménard, D. Laflamme, B. Raitt, and J. Cowles. Ai and physics assisted character pose authoring. In *Proc. SIGGRAPH'22 Real-Time Live!* Association for Computing Machinery, 2022.

Botond Bócsi, Duy Nguyen-Tuong, Lehel Csató, Bernhard Schoelkopf, and Jan Peters. Learning inverse kinematics with structured prediction. In *Proc. International Conference on Intelligent Robots and Systems*, pages 698–703. IEEE, 2011.

Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter Gehler, Javier Romero, and Michael J Black. Keep it smpl: Automatic estimation of 3d human pose and shape from a single image. In *Proc. ECCV*, pages 561–578. Springer, 2016.

Sam Bond-Taylor and Chris G Willcocks. ∞-diff: Infinite resolution diffusion with subsampled mollified states. *arXiv preprint arXiv:2303.18242*, 2023.

Lubomir Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *International Conference on Computer Vision*, sep 2009. URL `http://www.eecs.berkeley.edu/~lbourdev/poselets`.

Maury Bramson, Jian Ding, and Ofer Zeitouni. Convergence in law of the maximum of the two-dimensional discrete gaussian free field. *Communications on Pure and Applied Mathematics*, 69(1):62–123, 2016.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer*

Vision and Pattern Recognition, pages 18392–18402, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Adv. Neural Inform. Process. Syst.*, 33:1877–1901, 2020.

Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

Peter J Burt. Fast filter transform for image processing. *Computer graphics and image processing*, 16(1):20–51, 1981.

John Charles Butcher. *Numerical methods for ordinary differential equations.* John Wiley & Sons, 2016.

Lluís Castrejón, Nicolas Ballas, and Aaron C. Courville. Improved conditional vrnns for video prediction. *ArXiv*, abs/1904.12165, 2019.

Jianfei Chen, Cheng Lu, Biqi Chenli, Jun Zhu, and Tian Tian. Vflow: More expressive generative flows with variational data augmentation. In *International Conference on Machine Learning*, 2020.

Nanxin Chen, Yu Zhang, Heiga Zen, Ron J. Weiss, Mohammad Norouzi, and William Chan. Wavegrad: Estimating gradients for waveform generation. In *International Conference on Learning Representations*, 2021a.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018a.

Ricky TQ Chen, Jens Behrmann, David K Duvenaud, and Jörn-Henrik Jacobsen. Residual flows for invertible generative modeling. In *Advances in Neural Information Processing Systems*, pages 9916–9926, 2019.

Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. In *International Conference on Machine Learning*, pages 864–872. PMLR, 2018b.

Xin Chen, Anqi Pang, Wei Yang, Yuexin Ma, Lan Xu, and Jingyi Yu. Sportscap: Monocular 3d human motion capture and fine-grained understanding in challenging sports videos. *arXiv preprint arXiv:2104.11452*, 2021b.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Hyunsun Choi, Eric Jang, and Alexander A Alemi. Waic, but why? generative ensembles for robust anomaly detection. *arXiv preprint arXiv:1810.01392*, 2018.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. *Neural Information*

Processing Systems, 2015.

Aidan Clark, Jeff Donahue, and Karen Simonyan. Adversarial video generation on complex datasets. *arXiv preprint arXiv:1907.06571*, 2019.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

Richard Courant, Fritz John, Albert A Blank, and Alan Solomon. *Introduction to calculus and analysis*, volume 1. Springer, 1965.

Vicente Ruiz De Angulo and Carme Torras. Learning inverse kinematics: Reduced sampling through decomposition into virtual robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(6):1571–1577, 2008.

Sébastien de Blois, Mathieu Garon, Christian Gagné, and Jean-François Lalonde. Input dropout for spatially aligned modalities. In *IEEE Int. Conf. Image Process.*, pages 733–737, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1174–1183, 2018.

Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34, 2021.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. In *International Conference on Learned Representations Workshop*, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learned Representations*, 2017.

Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal. Learning inverse kinematics. In *Proc. International Conference on Intelligent Robots and Systems.*, volume 1, pages 298–303. IEEE, 2001.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In *Advances in Neural Information Processing Systems*, volume 32, pages 7511–7522, 2019. URL https://proceedings.neurips.cc/paper/2019/file/7ac71d433f282034e088473244df8c02-Paper.pdf.

Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. In *CoRL*, pages 344–356, 2017.

Ahmed El-Sherbiny, Mostafa A Elhosseini, and Amira Y Haikal. A comparative study of soft computing methods to solve inverse kinematics problem. *Ain Shams Engineering Journal*, 9(4):2535–2548, 2018.

Patrick Esser, Johnathan Chiu, Parmida Atighehchian, Jonathan Granskog, and Anastasis Germanidis. Structure and content-guided video synthesis with diffusion models. *arXiv preprint arXiv:2302.03011*, 2023.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. *International Conference on Machine Learning*, 2020.

Jean-Yves Franceschi, Edouard Delasalles, Mickaël Chen, Sylvain Lamprier, and Patrick Gallinari. Stochastic latent residual video prediction. In *International Conference on Machine Learning*, pages 3233–3246. PMLR, 2020.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H. Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion, 2022. URL https://arxiv.org/abs/2208.01618.

Xiaojie Gao, Yueming Jin, Qi Dou, Chi-Wing Fu, and Pheng-Ann Heng. Accurate grid keypoint learning for efficient video prediction. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5908–5915. IEEE, 2021.

Leon Gatys, Alexander Ecker, and Matthias Bethge. A neural algorithm of artistic style. *Journal of Vision*, 16(12):326, September 2016. doi: 10.1167/16.12.326. URL https://doi.org/10.1167/16.12.326.

Saeed Ghorbani, Kimia Mahdaviani, Anne Thaler, Konrad Kording, Douglas James Cook, Gunnar Blohm, and Nikolaus F Troje. Movi: A large multi-purpose human motion and video dataset. *Plos one*, 16(6):e0253157, 2021.

Arnab Ghosh, Harkirat Singh Behl, Emilien Dupont, Philip HS Torr, and Vinay Namboodiri. Steer: Simple temporal regularization for neural odes. In *Advances in Neural Information Processing Systems*, 2020.

Michael Gleicher. Retargetting motion to new characters. In *Proc. Annual conference on nomputer graphics and interactive techniques*, pages 33–42, 1998.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

IJ Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, and Y Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 2014.

Erik W Grafarend. *Linear and nonlinear models: fixed effects, random effects, and mixed models*. de Gruyter, 2006.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

Aditya Grover, Manik Dhar, and Stefano Ermon. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *AAAI Conference on Artificial Intelligence*, 2018.

Paul Hagemann, Lars Ruthotto, Gabriele Steidl, and Nicole Tianjiao Yang. Multilevel diffusion: Infinite dimensional score-based diffusion models for image generation. *arXiv preprint arXiv:2303.04772*, 2023.

E Hairer, SP Norsett, and G Wanner. Solving ordinary, differential equations i, nonstiff problems/e. hairer, sp norsett, g. wanner, with 135 figures, vol.: 1. Technical report, 2Ed. Springer-Verlag, 2000, 2000.

William Harvey, Saeid Naderiparizi, Vaden Masrani, Christian Weilbach, and Frank Wood. Flexible diffusion modeling of long videos. *Advances in Neural Information Processing Systems*, 2022.

Yingqing He, Tianyu Yang, Yong Zhang, Ying Shan, and Qifeng Chen. Latent video diffusion models for high-fidelity video generation with arbitrary lengths. *arXiv preprint arXiv:2211.13221*, 2022.

Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*, 2017.

Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. Deep anomaly detection with outlier exposure. In *International Conference on Learning Representations*, 2019.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, 2019a.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*, 2019b.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 2020.

Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022a.

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *arXiv preprint arXiv:2204.03458*, 2022b.

Sina Honari, Jason Yosinski, Pascal Vincent, and Christopher Pal. Recombinator networks: Learning coarse-to-fine feature aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5743–5752, 2016.

Emiel Hoogeboom, Rianne van den Berg, and Max Welling. Emerging convolutions for generative normalizing flows. In *International Conference on Machine Learning*, 2019a.

Emiel Hoogeboom, Jorn Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems*, volume 32, pages 12134–12144, 2019b. URL `https://proceedings.neurips.cc/paper/2019/file/9e9a30b74c49d07d8150c8c83b1ccf07-Paper.pdf`.

Emiel Hoogeboom, , Victor Garcia Satorras, Jakub Tomczak, and Max Welling. The convolution exponential and generalized sylvester flows. In *Advances in Neural Information Processing Systems*, 2020.

Tobias Höppe, Arash Mehrjou, Stefan Bauer, Didrik Nielsen, and Andrea Dittadi. Diffusion models for video prediction and infilling. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL `https://openreview.net/forum?id=lf0lr4AYM6`.

Anders Høst-Madsen, Elyas Sabeti, and Chad Walton. Data discovery and anomaly detection using atypicality: Theory. *IEEE Transactions on Information Theory*, 65(9):5302–5322, 2019.

Chin-Wei Huang, Laurent Dinh, and Aaron Courville. Augmented normalizing flows: Bridging the gap between generative flows and latent variable models. *arXiv preprint arXiv:2002.07101*, 2020.

Han-Hsien Huang and Mi-Yen Yeh. Accelerating continuous normalizing flow with trajectory polynomial regularization. *AAAI Conference on Artificial Intelligence*, 2021.

Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–1339, jul 2014.

Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 9000–9008, 2018.

Jianbin Jiang, Tan Wang, He Yan, and Junhui Liu. Clothformer: Taming video virtual try-on in all module. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530—-1538, 2015.

Beibei Jin, Yu Hu, Qiankun Tang, Jingyu Niu, Zhiping Shi, Yinhe Han, and Xiaowei Li. Exploring spatial-temporal multi-frequency analysis for high-fidelity and temporal-consistency video prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4554–4563, 2020.

Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. *arXiv preprint arXiv:2105.14080*, 2021a.

Alexia Jolicoeur-Martineau, Rémi Piché-Taillefer, Rémi Tachet des Combes, and Ioannis Mitliagkas. Adversarial score matching and improved sampling for image generation. *International Conference on Learning Representations*, 2021b. URL `arXivpreprintarXiv:2009.05475`.

Heewoo Jun and Alex Nichol. Shap-e: Generating conditional 3d implicit functions, 2023.

Heewoo Jun, Rewon Child, Mark Chen, John Schulman, Aditya Ramesh, Alec Radford, and Ilya Sutskever. Distribution augmentation for generative modeling. In *International Conference on Machine Learning*, pages 10563–10576, 2020.

Mahdi Karami, Dale Schuurmans, Jascha Sohl-Dickstein, Laurent Dinh, and Daniel Duckworth. Invertible convolutional flow. In *Advances in Neural Information Processing Systems*, volume 32, pages 5635–5645, 2019. URL `https://proceedings.neurips.cc/paper/2019/file/b1f62fa99de9f27a048344d55c5ef7a6-Paper.pdf`.

Animesh Karnewar and Oliver Wang. Msg-gan: Multi-scale gradients for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7799–7808, 2020.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *International Conference on Learned Representations*, 2018.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. Denoising diffusion restoration models. *arXiv preprint arXiv:2201.11793*, 2022.

Bahjat Kawar, Shiran Zada, Oran Lang, Omer Tov, Huiwen Chang, Tali Dekel, Inbar Mosseri, and Michal Irani. Imagic: Text-based real image editing with diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6007–6017, 2023.

Mitsuo Kawato, Hiroaki Gomi, Masazumi Katayamat, and Yasuharu Koike. Supervised learning for coordinative motor control. *Computational learning & cognition*, pages 126–161, 1993.

Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *ArXiv*, abs/1705.06950, 2017.

Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, and David Duvenaud. Learning differential equations that are easy to solve. In *Advances in Neural Information Processing Systems*, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pages 10215–10224, 2018.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper/2016/file/ddeebdeefdb7e7e7a697e1c3e3d8ef54-Paper.pdf`.

Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Why normalizing flows fail to detect out-of-distribution data. In *Advances in neural information processing systems*, volume 33, 2020.

Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. In *International Conference on Learning Representations*, 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, 2009a. URL `https://www.cs.toronto.edu/~kriz/cifar.html`.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). *Tech report*, 2009b. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

Wilhelm Kutta. Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

Christoph Lassner, Javier Romero, Martin Kiefel, Federica Bogo, Michael J. Black, and Peter V. Gehler. Unite the people: Closing the loop between 3d and 2d human representations. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, July 2017. URL `http://up.is.tuebingen.mpg.de`.

Guillaume Le Moing, Jean Ponce, and Cordelia Schmid. Ccvs: Context-aware controllable video synthesis. *Advances in Neural Information Processing Systems*, 34, 2021.

Alex X. Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *ArXiv*, abs/1804.01523, 2018a.

Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 7167–7177, 2018b.

Sang-gil Lee, Sungwon Kim, and Sungroh Yoon. Nanoflow: Scalable normalizing flows with sublinear parameter complexity. In *Advances in Neural Information Processing Systems*, 2020.

Jiefeng Li, Chao Xu, Zhicun Chen, Siyuan Bian, Lixin Yang, and Cewu Lu. Hybrik: A hybrid analytical-neural inverse kinematics solution for 3d human pose and shape estimation. In *Proc. CVPR*, pages 3383–3393, 2021.

Zengyi Li, Yubei Chen, and Friedrich T Sommer. Learning energy-based models in high-dimensional spaces with multi-scale denoising score matching. *arXiv preprint arXiv:1910.07762*, 2019.

Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*, 2018.

Jae Hyun Lim, Nikola B Kovachki, Ricardo Baptista, Christopher Beckham, Kamyar Azizzadenesheli, Jean Kossaifi, Vikram Voleti, Jiaming Song, Karsten Kreis, Jan Kautz, et al. Score-based diffusion models in function space. *arXiv preprint arXiv:2302.07400*, 2023.

Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 300–309, 2023.

Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. In *Advances in neural information processing systems*, pages 1498–1507, 2018.

Tony Lindeberg. Scale-space for discrete signals. *IEEE transactions on pattern analysis and machine intelligence*, 12(3):234–254, 1990.

Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. *arXiv preprint arXiv:2202.09778*, 2022.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M.J. Black. Smpl: A skinned multi-person linear model. *ACM TOG*, 34(6):1–16, 2015.

Chaochao Lu, Michael Hirsch, and Bernhard Scholkopf. Flexible spatio-temporal networks for video prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern*

Recognition, pages 6523–6531, 2017.

You Lu and Bert Huang. Woodbury transformations for deep generative flows. In *Advances in Neural Information Processing Systems*, 2020.

Pauline Luc, Aidan Clark, Sander Dieleman, Diego de Las Casas, Yotam Doron, Albin Cassirer, and Karen Simonyan. Transformation-based adversarial video prediction on large-scale data. *arXiv preprint arXiv:2003.04035*, 2020.

Zhengxiong Luo, Dayou Chen, Yingya Zhang, Yan Huang, Liang Wang, Yujun Shen, Deli Zhao, Jinren Zhou, and Tieniu Tan. Videofusion: Decomposed diffusion models for high-quality video generation. In *Proc. the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Zhengyi Luo, S. Alireza Golestaneh, and Kris M. Kitani. 3d human motion estimation via motion compression and refinement. In *Proc. the Asian Conference on Computer Vision (ACCV)*, November 2020.

Jianxin Ma, Shuai Bai, and Chang Zhou. Pretrained diffusion models for unified human motion synthesis. *arXiv preprint arXiv:2212.02837*, 2022.

Xuezhe Ma, Xiang Kong, Shanghang Zhang, and Eduard Hovy. Macow: Masked convolutional generative flow. In *Advances in Neural Information Processing Systems*, pages 5893–5902, 2019.

Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision*, pages 5442–5451, October 2019.

Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, and Brendan Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

Stephane G Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, 11(7):674–693, 1989.

Stéphane G. Mallat and Gabriel Peyré. *A wavelet tour of signal processing: the sparse way.* Elsevier, 2009.

David Marr. *Vision: A computational investigation into the human representation and processing of visual information.* MIT press, 2010.

James L McClelland, David E Rumelhart, PDP Research Group, et al. *Parallel distributed processing*, volume 2. MIT press Cambridge, MA, 1986.

Leland McInnes, Jasmine Healy, Nathaniel Saul, and Lukas Großberger. Umap: Uniform manifold approximation and projection. *J. Open Source Software*, 3:861, 2018.

Dushyant Mehta, Helge Rhodin, Dan Casas, Pascal Fua, Oleksandr Sotnychenko, Weipeng Xu, and Christian Theobalt. Monocular 3d human pose estimation in the wild using improved cnn supervision. In *3D Vision (3DV), 2017 Fifth International Conference on*. IEEE, 2017. doi: 10.1109/3dv.2017.00064. URL `http://gvv.mpi-inf.mpg.de/3dhp_dataset`.

Dushyant Mehta, Oleksandr Sotnychenko, Franziska Mueller, Weipeng Xu, Srinath Sridhar, Gerard Pons-Moll, and Christian Theobalt. Single-shot multi-person 3d pose estimation from monocular rgb. In *3D Vision (3DV), 2018 Sixth International Conference on*. IEEE, sep 2018. URL `http://gvv.mpi-inf.mpg.de/projects/SingleShotMultiPerson`.

Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. SDEdit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2022.

Jacob Menick and Nal Kalchbrenner. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. In *International Conference on Learning Representations*, 2019.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

Matthias Minderer, Chen Sun, Ruben Villegas, Forrester Cole, Kevin P Murphy, and Honglak Lee. Unsupervised learning of object structure and dynamics from videos. *Advances in Neural Information Processing Systems*, 32, 2019.

Nykan Mirchi, Vincent Bissonnette, Recai Yilmaz, Nicole Ledwos, Alexander Winkler-Schwartz, and Rolando F. Del Maestro. The virtual operative assistant: An explainable artificial intelligence tool for simulation-based training in surgery and medicine. *PLOS ONE*, 15(2):e0229596, February 2020. doi: 10.1371/journal.pone.0229596. URL `https://doi.org/10.1371/journal.pone.0229596`.

Lucas Mourot, Ludovic Hoyet, François Le Clerc, François Schnitzler, and Pierre Hellier. A survey on deep learning for skeleton-based human animation. In *Computer Graphics Forum*, volume 41, pages 122–157. Wiley Online Library, 2022.

Andres Munoz, Mohammadreza Zolfaghari, Max Argus, and Thomas Brox. Temporal shift gan for large scale video generation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3179–3188, 2021.

Eliya Nachmani, Robin San Roman, and Lior Wolf. Denoising diffusion gamma models. *arXiv preprint arXiv:2110.05948*, 2021.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Do deep generative models know what they don't know? In *International Conference on Learning Representations*, 2019a.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, and Balaji Lakshminarayanan. Detecting out-of-distribution inputs to deep generative models using a test for typicality. *arXiv preprint arXiv:1906.02994*, 5, 2019b.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

Didrik Nielsen and Ole Winther. Closing the dequantization gap: Pixelcnn as a single-layer flow. In *Advances in Neural Information Processing Systems*, 2020.

Yaniv Nikankin, Niv Haim, and Michal Irani. Sinfusion: Training diffusion models on a single image or video. *arXiv preprint arXiv:2211.11743*, 2022.

Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 670–679, 2017.

Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto. Ot-flow: Fast and accurate continuous normalizing flows via optimal transport. *AAAI Conference on Artificial Intelligence*, 2021.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *International Conference on Machine Learning*, 2016.

Boris N. Oreshkin, Florent Bocquelet, Felix G. Harvey, Bay Raitt, and Dominic Laflamme. Protores: Proto-residual network for pose authoring via learned inverse kinematics. In *Proc. ICLR*, 2021.

Ahmed A A Osman, Timo Bolkart, and Michael J. Black. STAR: A sparse trained articulated human body regressor. In *Proc. ECCV*, pages 598–613, 2020.

Xingang Pan, Ayush Tewari, Thomas Leimkühler, Lingjie Liu, Abhimitra Meka, and Christian Theobalt. Drag your gan: Interactive point-based manipulation on the generative image manifold. In *ACM SIGGRAPH 2023 Conference Proceedings*, 2023.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

Sunghyun Park, Kangyeol Kim, Junsoo Lee, Jaegul Choo, Joonseok Lee, Sookyung Kim, and Edward Choi. Vid-ode: Continuous-time video generation with neural ordinary differential equation. *arXiv preprint arXiv:2010.08188*, page online, 2021.

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2337–2346, 2019.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International Conference on Machine Learning*, 2018.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed AA Osman, Dimitrios Tzionas, and Michael J Black. Expressive body capture: 3d hands, face, and body from a single image. In *Proc. CVPR*, pages 10975–10985, 2019.

LS Pontyagin, VG Boltyanskii, RV Gamkrelidze, and EF Mishchenko. The mathematical theory of optimal processes. *Interscience, NY*, 1962.

Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022.

Ben Poole, Ajay Jain, Jonathan T. Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=FjNys5c7VyY`.

Mr Prabhat, Evan Racah, Jim Biard, Yunjie Liu, Mayur Mudigonda, Karthik Kashinath, Christopher Beckham, Tegan Maharaj, Samira Kahou, Chris Pal, et al. Deep learning for extreme weather detection. In *AGU Fall Meeting Abstracts*, 2017.

William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

Alfio Quarteroni, Riccardo Sacco, Fausto Saleri, and P Gervasio. Matematica numerica 2a edizione, 2000.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Jathushan Rajasegaran, Georgios Pavlakos, Angjoo Kanazawa, and Jitendra Malik. Tracking people with 3d representations. *Proc. NIPS*, 34:23703–23713, 2021.

Ruslan Rakhimov, Denis Volkhonskiy, Alexey Artemov, Denis Zorin, and Evgeny Burnaev. Latent video transformer. *arXiv preprint arXiv:2006.10704*, 2020.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in Neural Information Processing Systems*, pages 14866–14876, 2019.

Scott Reed, Aäron van den Oord, Nal Kalchbrenner, Sergio Gómez Colmenarejo, Ziyu Wang, Dan Belov, and Nando De Freitas. Parallel multiscale autoregressive density estimation. In *International Conference on Machine Learning*, 2017.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Severi Rissanen, Markus Heinonen, and Arno Solin. Generative modelling with inverse heat dissipation. *arXiv preprint arXiv:2206.13397*, 2022.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

J. Romero, D. Tzionas, and M.J. Black. Embodied hands: Modeling and capturing hands and bodies together. In *Proc. SIGGRAPH Asia*, volume 36, November 2017.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Azriel Rosenfeld and Avinash C. Kak. Picture processing by computer. *ACM Computing Surveys (CSUR)*, 1(3):147–176, 1969.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22500–22510, 2023.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Elyas Sabeti and Anders Høst-Madsen. Data discovery and anomaly detection using atypicality for real-valued data. *Entropy*, 21(3):219, 2019.

Chitwan Saharia, William Chan, Huiwen Chang, Chris A Lee, Jonathan Ho, Tim Salimans, David J Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. *arXiv preprint arXiv:2111.05826*, 2021.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

Masaki Saito, Eiichi Matsumoto, and Shunta Saito. Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE international conference on computer vision*, pages 2830–2839, 2017.

Masaki Saito, Shunta Saito, Masanori Koyama, and Sosuke Kobayashi. Train sparsely, generate densely: Memory-efficient unsupervised training of high-resolution temporal gan. *International Journal of Computer Vision*, 128(10):2586–2606, 2020.

Seyed Sadegh Mohseni Salehi, Shadab Khan, Deniz Erdogmus, and Ali Gholipour. Real-time deep pose estimation with geodesic loss for image-to-template rigid registration. *IEEE Trans. Medical Imaging*, 38(2):470–481, 2018.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, 2016.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Saeed Saremi and Aapo Hyvarinen. Neural empirical bayes. *Journal of Machine Learning Research*, 20:1–23, 2019.

Simo Särkkä and Arno Solin. *Applied stochastic differential equations*, volume 10. Cambridge University Press, 2019.

Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local svm approach. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 32–36. IEEE, 2004.

Joan Serrà, David Álvarez, Vicenç Gómez, Olga Slizovskaia, José F Núñez, and Jordi Luque. Input complexity and out-of-distribution detection with likelihood-based generative models. In *International Conference on Learning Representations*, 2020.

Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4570–4580, 2019.

Scott Sheffield. Gaussian free fields for mathematicians. *Probability theory and related fields*, 139(3):521–541, 2007.

Yujun Shi, Chuhui Xue, Jiachun Pan, Wenqing Zhang, Vincent Y. F. Tan, and Song Bai. Dragdiffusion: Harnessing diffusion models for interactive point-based image editing. *arXiv preprint arXiv:2306.14435*, 2023.

Leonid Sigal, Alexandru O Balan, and Michael J Black. Humaneva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. *International journal of computer vision*, 87(1-2):4, 2010.

Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*, 2022.

Uriel Singer, Shelly Sheynin, Adam Polyak, Oron Ashual, Iurii Makarov, Filippos Kokkinos, Naman Goyal, Andrea Vedaldi, Devi Parikh, Justin Johnson, et al. Text-to-4d dynamic scene generation. *arXiv preprint arXiv:2301.11280*, 2023.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science, 1986.

Jon Sneyers and Pieter Wuille. Flif: Free lossless image format based on maniac compression. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 66–70. IEEE, 2016.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *International Conference on Learning Representations*, 2021a. URL `https://arxiv.org/abs/2010.02502`.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 2019.

Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in Neural Information Processing Systems*, 2020.

Yang Song, Chenlin Meng, and Stefano Ermon. Mintnet: Building invertible neural networks with masked convolutions. In *Advances in Neural Information Processing Systems*, pages 11004–11014, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*, 2021b. URL `arXivpreprintarXiv:2011.13456`.

Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852. PMLR, 2015.

Gilbert Strang. *Calculus*, volume 1. SIAM, 1991.

Yu Sun, Qian Bao, Wu Liu, Yili Fu, Michael J Black, and Tao Mei. Monocular, one-stage, regression of multiple 3D people. In *Proc. ICCV*, pages 11179–11188, 2021a.

Yu Sun, Qian Bao, Wu Liu, Yili Fu, Michael J Black, and Tao Mei. Monocular, one-stage, regression of multiple 3d people. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11179–11188, 2021b.

Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations*, 2016.

Seiya Tokui, Ryosuke Okuta, Takuya Akiba, Yusuke Niitani, Toru Ogawa, Shunta Saito, Shuji Suzuki, Kota Uenishi, Brian Vogel, and Hiroyuki Yamazaki Vincent. Chainer: A deep learning framework for accelerating the research cycle. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2002–2011, 2019.

Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 648–656, 2015.

Matt Trumble, Andrew Gilbert, Charles Malleson, Adrian Hilton, and John Collomosse. Total capture: 3d human pose estimation fusing video and inertial sensors. In *2017 British Machine Vision Conference (BMVC)*, 2017.

Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1526–1535, 2018.

Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

Arash Vahdat and Jan Kautz. Nvae: A deep hierarchical variational autoencoder. In *Advances in Neural Information Processing Systems*, 2020.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Adv. Neural Inform. Process. Syst.*, 30, 2017.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inform. Process. Syst.*, volume 30, 2017.

Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. In *International Conference on Learning Representations*, 2017.

Ruben Villegas, Arkanath Pathak, Harini Kannan, Dumitru Erhan, Quoc V Le, and Honglak Lee. High fidelity video prediction with large stochastic recurrent neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Ruben Villegas, Mohammad Babaeizadeh, Pieter-Jan Kindermans, Hernan Moraldo, Han Zhang, Mohammad Taghi Saffar, Santiago Castro, Julius Kunze, and Dumitru Erhan.

Phenaki: Variable length video generation from open domain textual descriptions. In *International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=vOEXS39nOF`.

Vikram Voleti, David Kanaa, Samira Ebrahimi Kahou, and Christopher Pal. Neural ordinary differential equations. *Workshop at Advances in Neural Information Processing Systems*, 2019.

Vikram Voleti, Chris Finlay, Adam M Oberman, and Christopher Pal. Improving continuous normalizing flows using a multi-resolution framework. In *International Conference on Machine Learning 2021 Workshop*, 2021.

Vikram Voleti, Alexia Jolicoeur-Martineau, and Christopher Pal. Mcvd: Masked conditional video diffusion for prediction, generation, and interpolation. In *(NeurIPS) Advances in Neural Information Processing Systems*, 2022a. URL `https://arxiv.org/abs/2205.09853`.

Vikram Voleti, Boris Oreshkin, Florent Bocquelet, Félix Harvey, Louis-Simon Ménard, and Christopher Pal. Smpl-ik: Learned morphology-aware inverse kinematics for ai driven artistic workflows. *SIGGRAPH Asia 2022 Technical Communications*, pages 1–7, 2022b.

Vikram Voleti, Christopher Pal, and Adam M Oberman. Score-based denoising diffusion with non-isotropic gaussian noise models. In *Advances in Neural Information Processing Systems 2022 Workshop on Score-Based Methods*, 2022c.

Timo von Marcard, Roberto Henschel, Michael J Black, Bodo Rosenhahn, and Gerard Pons-Moll. Recovering accurate 3d human pose in the wild using imus and a moving camera. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 601–617, 2018.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. *Advances in neural information processing systems*, 29, 2016.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018.

Ze Wang, Seunghyun Hwang, Zichen Miao, and Qiang Qiu. Image generation using continuous filter atoms. *Advances in Neural Information Processing Systems*, 34:17826–17838, 2021.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4): 600–612, 2004.

Dirk Weissenborn, Oscar Täckström, and Jakob Uszkoreit. Scaling autoregressive video models. *arXiv preprint arXiv:1906.02634*, 2019.

Wendelin Werner and Ellen Powell. Lecture notes on the gaussian free field. *arXiv preprint arXiv:2004.04720*, 2020.

Andrew P Witkin. Scale-space filtering. In *Readings in Computer Vision*, pages 329–332. Elsevier, 1987.

Bohan Wu, Suraj Nair, Roberto Martin-Martin, Li Fei-Fei, and Chelsea Finn. Greedy hierarchical variational autoencoders for large-scale video prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2318–2328, 2021.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

Changyi Xiao and Ligang Liu. Generative flows with matrix exponential. In *International Conference on Machine Learning*, 2020.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion GANs. In *International Conference on Learning Representations (ICLR)*, 2022.

SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

Qiangeng Xu, Hanwang Zhang, Weiyue Wang, Peter Belhumeur, and Ulrich Neumann. Stochastic dynamics for video infilling. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2714–2723, 2020.

Yilun Xu, Ziming Liu, Max Tegmark, and Tommi Jaakkola. Poisson flow generative models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 16782–16795. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/6ad68a54eaa8f9bf6ac698b02ec05048-Paper-Conference.pdf`.

Yucheng Xu, Nanbo Li, Arushi Goel, Zijian Guo, Zonghai Yao, Hamidreza Kasaei, Mohammad-Sajad Kasaei, and Zhibin Li. Controllable video generation by learning the underlying dynamical system with neural ode. *ArXiv*, abs/2303.05323, 2023.

Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman. Video enhancement with task-oriented flow. *Int. J. Comput. Vis.*, 127(8):1106–1125, 2019.

Hanshu Yan, Jiawei Du, Vincent Y. F. Tan, and Jiashi Feng. On robustness of neural ordinary differential equations. *International Conference on Learning Representations*, 2020.

Wilson Yan, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. Videogpt: Video generation using vq-vae and transformers. *arXiv preprint arXiv:2104.10157*, 2021.

Ren Yang, Fabian Mentzer, Luc Van Gool, and Radu Timofte. Learning for video compression with recurrent auto-encoder and recurrent probability model. *IEEE Journal of Selected Topics in Signal Processing*, 15(2):388–401, 2020.

Ruihan Yang, Prakhar Srivastava, and Stephan Mandt. Diffusion probabilistic modeling for video generation. *arXiv preprint arXiv:2203.09481*, 2022.

Shengming Yin, Chenfei Wu, Huan Yang, Jianfeng Wang, Xiaodong Wang, Minheng Ni, Zhengyuan Yang, Linjie Li, Shuguang Liu, Fan Yang, et al. Nuwa-xl: Diffusion over diffusion for extremely long video generation. *arXiv preprint arXiv:2303.12346*, 2023.

Jason Yu, Konstantinos Derpanis, and Marcus Brubaker. Wavelet flow: Fast training of high resolution normalizing flows. In *Advances in Neural Information Processing Systems*, 2020.

Sihyun Yu, Jihoon Tack, Sangwoo Mo, Hyunsu Kim, Junho Kim, Jung-Woo Ha, and Jinwoo Shin. Generating videos with dynamics-aware implicit generative adversarial networks. In *International Conference on Learning Representations*, 2022.

Sihyun Yu, Kihyuk Sohn, Subin Kim, and Jinwoo Shin. Video probabilistic diffusion models in projected latent space. *arXiv preprint arXiv:2302.07685*, 2023.

Vladyslav Yushchenko, Nikita Araslanov, and Stefan Roth. Markov decision process for video generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. *arXiv preprint arXiv:2302.05543*, 2023.

Yuxuan Zhang, Huan Ling, Jun Gao, Kangxue Yin, Jean-Francois Lafleche, Adela Barriuso, Antonio Torralba, and Sanja Fidler. Datasetgan: Efficient labeled data factory with minimal human effort. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10145–10155, 2021.

Daquan Zhou, Weimin Wang, Hanshu Yan, Weiwei Lv, Yizhe Zhu, and Jiashi Feng. Magicvideo: Efficient video generation with latent diffusion models. *arXiv preprint arXiv:2211.11018*, 2022.

Yuxiao Zhou. Minimal-ik : https://github.com/calciferzh/minimal-ik, 2020.

Çagatay Yildiz, Markus Heinonen, and Harri Lähdesmäki. Ode2vae: Deep generative second order odes with bayesian neural networks. In *Advances in Neural Information Processing Systems*, 2019.