Université de Montréal

Natural Image Processing and Synthesis Using Deep Learning

par

Yaroslav Ganin

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

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en informatique

September 23, 2019

SOMMAIRE

Nous étudions dans cette thèse comment les réseaux de neurones profonds peuvent être utilisés dans différents domaines de la vision artificielle. La vision artificielle est un domaine interdisciplinaire qui traite de la compréhension d'images et de vidéos numériques. Les problèmes de ce domaine ont traditionnellement été adressés avec des méthodes ad-hoc nécessitant beaucoup de réglages manuels. En effet, ces systèmes de vision artificiels comprenaient jusqu'à récemment une série de modules optimisés indépendamment. Cette approche est très raisonnable dans la mesure où, avec peu de données, elle bénéficient autant que possible des connaissances du chercheur. Mais cette avantage peut se révéler être une limitation si certaines données d'entré n'ont pas été considérées dans la conception de l'algorithme. Avec des volumes et une diversité de données toujours plus grands, ainsi que des capacités de calcul plus rapides et économiques, les réseaux de neurones profonds optimisés d'un bout à l'autre sont devenus une alternative attrayante. Nous démontrons leur avantage avec une série d'articles de recherche, chacun d'entre eux trouvant une solution à base de réseaux de neurones profonds à un problème d'analyse ou de synthèse visuelle particulier.

Dans le premier article, nous considérons un problème de vision classique: la détection de bords et de contours. Nous partons de l'approche classique et la rendons plus 'neurale' en combinant deux étapes, la détection et la description de motifs visuels, en un seul réseau convolutionnel. Cette méthode, qui peut ainsi s'adapter à de nouveaux ensembles de données, s'avère être au moins aussi précis que les méthodes conventionnelles quand il s'agit de domaines qui leur sont favorables, tout en étant beaucoup plus robuste dans des domaines plus générales. Dans le deuxième article, nous construisons une nouvelle architecture pour la manipulation d'images qui utilise l'idée que la majorité des pixels produits peuvent d'être copiés de l'image d'entrée. Cette technique bénéficie de plusieurs avantages majeurs par rapport à l'approche conventionnelle en apprentissage profond. En effet, elle conserve les détails

de l'image d'origine, n'introduit pas d'aberrations grâce à la capacité limitée du réseau sous-jacent et simplifie l'apprentissage. Nous démontrons l'efficacité de cette architecture dans le cadre d'une tâche de correction du regard, où notre système produit d'excellents résultats.

Dans le troisième article, nous nous éclipsons de la vision artificielle pour étudier le problème plus générale de l'adaptation à de nouveaux domaines. Nous développons un nouvel algorithme d'apprentissage, qui assure l'adaptation avec un objectif auxiliaire à la tâche principale. Nous cherchons ainsi à extraire des motifs qui permettent d'accomplir la tâche mais qui ne permettent pas à un réseau dédié de reconnaître le domaine. Ce réseau est optimisé de manière simultané avec les motifs en question, et a pour tâche de reconnaître le domaine de provenance des motifs. Cette technique est simple à implémenter, et conduit pourtant à l'état de l'art sur toutes les tâches de référence.

Enfin, le quatrième article présente un nouveau type de modèle génératif d'images. À l'opposé des approches conventionnels à base de réseaux de neurones convolutionnels, notre système baptisé SPIRAL décrit les images en termes de programmes bas-niveau qui sont exécutés par un logiciel de graphisme ordinaire. Entre autres, ceci permet à l'algorithme de ne pas s'attarder sur les détails de l'image, et de se concentrer plutôt sur sa structure globale. L'espace latent de notre modèle est, par construction, interprétable et permet de manipuler des images de façon prévisible. Nous montrons la capacité et l'agilité de cette approche sur plusieurs bases de données de référence.

Mots clés: apprentissage profond, vision artificielle, réseaux de neurones, réseaux de neurones convolutionnels, détections de bords, correction du regard, transformateurs spatiaux, adaptation de domaine, adversaire, modèles génératifs, apprentissage par renforcement, graphisme inverse

SUMMARY

In the present thesis, we study how deep neural networks can be applied to various tasks in computer vision. Computer vision is an interdisciplinary field that deals with understanding of digital images and video. Traditionally, the problems arising in this domain were tackled using heavily hand-engineered adhoc methods. A typical computer vision system up until recently consisted of a sequence of independent modules which barely talked to each other. Such an approach is quite reasonable in the case of limited data as it takes major advantage of the researcher's domain expertise. This strength turns into a weakness if some of the input scenarios are overlooked in the algorithm design process.

With the rapidly increasing volumes and varieties of data and the advent of cheaper and faster computational resources end-to-end deep neural networks have become an appealing alternative to the traditional computer vision pipelines. We demonstrate this in a series of research articles, each of which considers a particular task of either image analysis or synthesis and presenting a solution based on a "deep" backbone.

In the first article, we deal with a classic low-level vision problem of edge detection. Inspired by a top-performing non-neural approach, we take a step towards building an end-to-end system by combining feature extraction and description in a single convolutional network. The resulting fully data-driven method matches or surpasses the detection quality of the existing conventional approaches in the settings for which they were designed while being significantly more usable in the out-of-domain situations.

In our second article, we introduce a custom architecture for image manipulation based on the idea that most of the pixels in the output image can be directly copied from the input. This technique bears several significant advantages over the naive black-box neural approach. It retains the level of detail of the original images, does not introduce artifacts due to insufficient capacity of the underlying neural network and simplifies training process, to name a few. We demonstrate the efficiency of the proposed architecture on the challenging gaze correction task where our system achieves excellent results.

In the third article, we slightly diverge from pure computer vision and study a more general problem of domain adaption. There, we introduce a novel training-time algorithm (*i.e.*, adaptation is attained by using an auxilliary objective in addition to the main one). We seek to extract features that maximally confuse a dedicated network called domain classifier while being useful for the task at hand. The domain classifier is learned simultaneosly with the features and attempts to tell whether those features are coming from the source or the target domain. The proposed technique is easy to implement, yet results in superior performance in all the standard benchmarks.

Finally, the fourth article presents a new kind of generative model for image data. Unlike conventional neural network based approaches our system dubbed SPIRAL describes images in terms of concise low-level programs executed by off-the-shelf rendering software used by humans to create visual content. Among other things, this allows SPIRAL not to waste its capacity on minutae of datasets and focus more on the global structure. The latent space of our model is easily interpretable by design and provides means for predictable image manipulation. We test our approach on several popular datasets and demonstrate its power and flexibility.

Keywords: deep learning, computer vision, neural networks, convolutional neural networks, edge detection, gaze correction, spatial transformers, domain adaptation, adversarial, generative models, reinforcement learning, inverse graphics

CONTENTS

Sommaire iii

Summary v

List of Tables xiii

List of Figures xv

Liste des sigles et des abréviations xxv

Acknowledgements xxvii

Chapter 1. Introduction 1

1.1. Structure of the Thesis 2

Chapter 2. Preliminaries 5

2.1. Machine Learning 5
2.1.1. Supervised Learning 6
2.1.2. Reinforcement Learning 7
2.1.3. Unsupervised Learning 8
2.1.4. Learning and Optimization 8
2.1.5. Capacity, Overfitting, Underfitting and Regularization 9
2.1.6. Covariate Shift and Domain Adaptation 10

2.2. Computer Vision and Image Processing 11
2.2.1. Image Classification 12
2.2.2. Image Processing 14

2.3. Neural Networks 16
2.3.1. Feed-forward Neural Networks 16
2.3.2. Optimization in Neural Networks 18

2.3.3. Convolutional Neural Networks 19
2.3.4. Recurrent Neural Networks 24
2.3.5. Adversarial Training of Neural Networks 27
2.4. Deep Reinforcement Learning Basics 28
Prologue to the First Article 33
Article Details 33
Context 33
Contributions 34
Recent Developments 34
Chapter 3. Neural Network Nearest Neighbor Fields for Image Transforms 37
3.1. Introduction 37
3.2. Related Work 38
3.3. N^4-Fields 40
3.3.1. Architecture 40
3.3.2. Training 41
3.3.3. Implementation Details 42
3.4. Experiments 44
3.5. Conclusion 51
Prologue to the Second Article 53
3.6. Article Details 53
3.7. Context 53
3.8. Contributions 54
3.9. Recent Developments 55
Chapter 4. Photorealistic Image Resynthesis for Gaze Manipulation .. 57

4.1. Introduction 57
4.2. Related Work 59
4.2.1. Deep Learning and Image Synthesis. 59
4.2.2. Gaze Manipulation. 60
4.3. The Model 61
4.3.1. Data Preparation 61
4.3.2. Warping Modules 62
4.3.3. Input Encoding 63
4.3.4. Lightness Correction Module 64
4.4. Experiments 65
4.4.1. Dataset 65
4.4.2. Training Procedure 66
4.4.3. Quantitative Evaluation 66
4.4.4. Perceptual Quality 68
4.4.5. Horizontal Redirection 69
4.4.6. Incorporating registration. 70
4.5. Discussion 71
Prologue to the Third Article 75
Article Details 75
Context 76
Contributions 77
Recent Developments 77
Chapter 5. Domain-Adversarial Training of Neural Networks 79
Introduction 79
5.1. Related work 81
5.2. Domain Adaptation 83
5.2.1. Domain Divergence 83
5.2.2. Proxy Distance 84

5.2.3. Generalization Bound on the Target Risk . . . 85
5.3. Domain-Adversarial Neural Networks . . . 85
5.3.1. Shallow Neural Networks . . . 86
5.3.2. Generalization to Arbitrary Architectures . . . 88
5.4. Experiments . . . 90
5.4.1. Baselines . . . 90
5.4.2. CNN Architectures and Training Procedure . . . 91
5.4.3. Visualizations . . . 93
5.4.4. Results On Image Datasets . . . 93
5.5. Conclusion . . . 97
Prologue to the Fourth Article . . . 99
Article Details . . . 99
Context . . . 100
Contributions . . . 101
Chapter 6. Synthesizing Programs for Images using Reinforced Adversarial Learning . . . 103
6.1. Introduction . . . 103
6.2. Related Work . . . 106
6.3. The SPIRAL Agent . . . 107
6.3.1. Overview . . . 107
6.3.2. Objectives . . . 107
6.3.3. Conditional Generation . . . 109
6.3.4. Distributed Learning . . . 109
6.4. Experiments . . . 110
6.4.1. Datasets . . . 110
6.4.2. Environments . . . 111
6.4.3. MNIST . . . 112
6.4.4. Omniglot . . . 113

6.4.5. CelebA 114
6.4.6. MuJoCo Scenes 115
6.5. Discussion 116
Chapter 7. Conclusions and Future Work 119
Future Work 121
Bibliography 125
Appendix A. Additional Details to Chapter 6 ..... A-i
A.1. Optimal D for Conditional Generation ..... A-i
A.2. Network Architectures ..... A-iii
A.3. Training Details ..... A-iii

LIST OF TABLES

3. I Edge detection results on BSDS500 (Arbeláez et al., 2011) (both for the original ground-truth annotation and "consensus" labels) and NYU RGBD (Silberman and Fergus, 2011). Our approach (N^4-fields) achieves performance which is better or comparable to the state-of-the-art. We also observe that the relative performance of the methods in terms of perceptual quality are not adequately reflected by the standard performance measures. The table list the quantitative results for the following methods: [1] Arbeláez et al. (2011), [2] Xiaofeng and Bo (2012), [3] Dollár and Zitnick (2013), [4] Kivinen et al. (2014), [5] Isola et al. (2014). 47

4. I Results of the **user study**. Each of the 16 test subjects was presented with 160 quadruplets containing 3 real images and one synthesized image. The task was to click on the latter as quickly as possible. The first three sections of the table contain the numbers of correct guesses (the smaller the better). The last row shows the mean time the participants spent to make a guess (the larger the better). Our full system (coarse-to-fine warping with lightness correction) outperforms the baselines. See Section 4.4.4 and (Ganin et al., 2016b) for details. 70

5. I Digit image classifications accuracies. The first row corresponds to the lower performance bound (no adaptation is performed). The last row corresponds to training on the target data with known labels (upper bound on the DA performance). For each of the two DA methods (ours and by Fernando et al. (2013)) we show how much of the gap between the lower and the upper bounds was covered (in brackets). 94

5. II Accuracy evaluation of different DA approaches on the standard Office (Saenko et al., 2010) dataset. All methods (except SA) are evaluated in the

"fully-transductive" protocol (some results are reproduced from (Long and Wang, 2015)). 94

LIST OF FIGURES

2.1 Some tasks are quite **easy for humans** but **very hard for machines**. For instance, in image classification, the computer has to deal with a raw RGB or binary representation of visual data (shown in the middle) that has almost no structure. 6

2.2 Relationship between **underfitting**, **overfitting** and **capacity**. Increasing capacity tends to reduce overfitting but increases the effect of overfitting. In the overfitting regime, the generalization gap is typically large so the training error poorly reflects the performance on the test set. We seek to find the optimal capacity (marked with a dashed red line) corresponding both to small generalization gap and to low training error. See Section 2.1.5 for details. . . . 9

2.3 Computation of the **SIFT** keypoint descriptor. Gradients for each the small square cells are weighted by a Gaussian window (blue circle) and accumulated into 4 histograms. The histograms are then concatenated to form the final descriptor. See Section 2.2.1 for details. Figure credit: Lowe (2004). 11

2.4 Training data for the **edge detection** task. Images are taken from the BSDS500 (Arbeláez et al., 2011) dataset. Each photo (on the left) has multiple ground-truth segmentations produced by different human annotators following the same guidelines. The ambiguity of task is reflected in a significant disagreement in annotations. Figure credit: Arbeláez et al. (2011). 14

2.5 **Multi-layer perceptron** (MLP). Each hidden neuron activation is computed as a non-linear function of a weighted sum of all neuron activations from the previous layer (hence the dense connection patterns in the diagram). See Section 2.3.1 for details. 16

2.6 **Receptive field** of a neuron in a higher layer of a convolutional neural network interacts with a large portion of the input despite local connectivity between adjacent layers. 20

2.7 The `AlexNet` architecture (Krizhevsky et al., 2012) used by the researchers from the University of Toronto to win the ILSVRC2012 (Russakovsky et al., 2015). It's a typical example of a convolutional neural network described in Section 2.3.3. Figure credit: Krizhevsky et al. (2012). 22

2.8 Two examples of transformations that can be performed by a **spatial transformer** module. The regular grid (red dots) in the output O is mapped to a sampling grid in the input I via operator $\mathcal{T}$. **(Left)** $\mathcal{T}$ is the *identity*. **(Right)** $\mathcal{T}$ is an *affine* transform. 23

2.9 An example of a **simple single hidden layer RNN** taking a sequence of inputs $(\mathbf{x}_1, \mathbf{x}_2, \ldots)$ and producing a sequence of outputs $(y_1, y_2, \ldots)$. **(Left)** The folded representation. The small hollow circle indicates a delay of a single time step. **(Right)** The unfolded (unrolled) form of the same network. The RNN becomes a feed-forward model with its weights shared across different time steps. 24

2.10 The architecture of an **LSTM** cell. The hollow circles indicate a delay of one time step. At the heart of the architecture lies the memory state $\mathbf{c}_t$ which contains an aggregate of the information the has flown through the cell so far. The gate $\mathbf{i}_t$ controls how much of the current input $(\mathbf{x}_t, \mathbf{h}_{t-1})$ is taken into account. The forget gate $\mathbf{f}_t$ allows to reset the memory state, and $\mathbf{o}_t$ limits the amount of the emitted signal. 26

2.11 The **agent-environment** interaction in a Markov decision process. The agent receives the current state of the environment s_t and a reward r_{t-1} and performs an action a_t that changes the state of the environment to s_{t+1}. 29

3.1 N^4-Fields can be applied to a range of complex image processing tasks, such as natural edge detection (left) or vessel segmentation (right). The proposed architecture combines the convolutional neural networks with the nearest neighbor search and is generic. It achieves state-of-the-art performance on standard benchmarks for these two rather different applications with very little customization or parameter tuning. 38

3.2 The N^4 architecture for natural edge detection. The input image is processed patch-by-patch. An input patch is first passed through a pretrained

convolutional neural network (CNN). Then, the output of the CNN is matched against the dictionary of sample CNN outputs that correspond to training patches with known annotations. The annotation corresponding to the nearest neighbor is transferred to the output. Overall, the output is obtained by averaging the overlapping transferred annotations. 40

3.3 The CNN architecture used in our experiments. See Section 3.3.3 for details. 42

3.4 Examples of nearest neighbor matchings of query patches to dictionary patches. For all patches, the ground truth annotations (edge sets) of the central parts are shown alongside. The righthand panels show the results of the nearest neighbor searches for different combinations of the query encoding and the dictionary patch encoding. *"Neural"* corresponds to the encoding with top-layer activations $\mathrm{CNN}(\mathcal{P})$ of the CNN, while *"Target"* corresponds to the "ground truth"' encoding $\mathrm{PCA}(\mathbf{A}(\mathcal{P}))$ that the CNN is being trained to replicate. Matching neural codes to target codes (Neural/Target) works poorly thus highlighting the gap between the neural codes and the target PCA codes (which is the manifestation of the underfitting during the CNN training). By using neural codes for both the queries and the dictionary patches, our approach is able to overcome such underfitting and to match many patches to correct annotations (see Neural/Neural matching). 45

3.5 Performance scores for different tolerance thresholds (default value is $\mathbf{0.75 \cdot 10^{-2}}$) used in the BSDS500 benchmark (Arbeláez et al., 2011). Algorithms' performance (ODS and OIS measures plotted as *dashed* and *solid* lines respectively) is going down as the tolerance threshold is decreased. N^4-fields (*blue* lines) handles more stringent thresholds better, which suggests that cleaner edges are produced, as is also evidenced by the qualitative results. See Section 3.4 for details. 46

3.6 The validation score (average precision) of the full N^4-fields and error rates (loss) of the underlying CNN measured throughout the training process. The strong correlation between the values suggests the importance of large-scale learning for the good performance of N^4-fields. This experiment was performed for the BSDS500 edge detection (hold out validation set included 20 images). 46

3.7 Representative results on the BSDS500 dataset. For comparison, we give the results of the best previously published method (Dollár and Zitnick, 2013). The red numbers correspond to Recall/Precision/F-measure. We give two examples where N^4-fields perform better than (Dollár and Zitnick, 2013), and one example (bottom row) where (Dollár and Zitnick, 2013) performs markedly better according to the quantitative measure. 48

3.8 Results on the NYU RGBD dataset. For comparison, we give the results of the best previously published method (SE) (Dollár and Zitnick, 2013) and the CNN baseline. We show a representative result where the N^4-fields perform better (top), similarly (middle), or worse (bottom) than the baseline, according to the numberic measures shown in red (recall/precision/F-measure format). We argue that the numerical measures do not adequately reflect the relative perceptual performance of the methods. 50

3.9 Representative results on the DRIVE dataset. A close match to the human expert annotation is observed. The last column corresponds to the algorithm proposed in (Dollár and Zitnick, 2013). 51

3.10 Results for the DRIVE dataset (Staal et al., 2004) in the form of the recall/precision curves. Our approach matches the performance of the current state-of-the-art method by SFLCSS (Becker et al., 2013) and performs much better than baselines and (Dollár and Zitnick, 2013). 51

4.1 Gaze redirection with our model trained for vertical gaze redirection. The model takes an input image (middle row) and the desired redirection angle (here varying between -15 and +15 degrees) and re-synthesize the new image with the new gaze direction. Note the preservation of fine details including specular highlights in the resynthesized images. 58

4.2 Examples of reconstructions produced by a modern **encoder-decoder** architecture (following the approach in (Kulkarni et al., 2015b; Reed et al., 2015; Ghodrati et al., 2015)) trained on our data. In each pair, the left image is the input and the right is the output. Despite our efforts, a noticeable loss of fine-scale details and "regression-to-mean" effect make the result not good

enough for most applications of gaze manipulation. Similar problems can be observed in (Kulkarni et al., 2015b; Reed et al., 2015; Ghodrati et al., 2015). 59

4.3 The **proposed system** takes an input eye region, feature points (`anchors`) as well as a correction `angle` α and sends them to the multi-scale neural network (see Section 4.3.2) predicting a `flow` field. The flow field is then applied to the input image to produce an image of a redirected eye. Finally, the output is enhanced by processing with the lightness correction neural network (see Section 4.3.4). 61

4.4 Visualization of three challenging redirection cases where **Lightness Correction Module** helps considerably compared to the system based solely on coarse-to-fine warping (CFW) which is having difficulties with expanding the area to the left of the iris. The 'Mask' column shows the soft mask corresponding to parts where lightness is increased. Lightness correction fixes problems with inpainting disoccluded eye-white, and what is more emphasizes the specular highlight increasing the perceived realism of the result. 64

4.5 Left – dataset collection process. Right – examples of two training pairs (input image with superimposed feature points on top, output image in the bottom). 65

4.6 Ordered errors for 15 redirection. Our multi-scale models (MS, CFW, CFW + LCM) show results that are comparable or superior the Random Forests (RF) (Kononenko and Lempitsky, 2015). 67

4.7 Distribution of errors over different correction angles. 68

4.8 Sample results on a hold-out. The full version of our model (CFW+LCM) outperforms other methods. 69

4.9 **Horizontal redirection** with a model trained for both vertical and horizontal gaze redirection. For the first six rows the angle varies from -15 to 15 relative to the central (input) image. The last two rows push the redirection to extreme angles (up to 45) breaking our model down. 71

4.10 The **basic warping architecture** Figure 4.10a takes an input eye region augmented with eye feature points information (`input`) as well as a correction `angle` and produces an image of the redirected eye. The model contains three main blocks: angle embedding module (`embed angle`) calculating a vector

representation of the correction `angle` and two warping modules (`process` 0.5×`-scale` Figure 4.10b and `process` 1×`-scale` Figure 4.10c) predicting and applying pixel-flow to the input image. 73

4.11 **Lightness Correction Module** increases lightness of selected regions. Figure 4.11a shows the actual architecture of the module. Multi-scale features are processed by the convolutional neural network presented in Figure 4.11b. 74

5.1 The **proposed architecture** includes a *feature extractor* (green) and a *label predictor* (blue), which together form a standard feed-forward architecture. Unsupervised domain adaptation is achieved by adding a *domain classifier* (red) connected to the feature extractor via a *gradient reversal layer* (GRL) that multiplies the gradient by a certain negative constant during the backpropagation-based training. The GRL ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier). 80

5.2 Examples of domain pairs used in the experimental evaluation of DANN. 91

5.3 CNN architectures used in the experiments. Boxes correspond to transformations applied to the data. Color-coding is the same as in Figure 5.1. 92

5.4 The effect of adaptation on the distribution of the extracted features (best viewed in color). The figure shows t-SNE (van der Maaten, 2013) visualizations of the CNN's activations **(a)** when no adaptation was performed and **(b)** when our adaptation procedure was incorporated into training. *Blue* points correspond to the source domain examples, while *red* ones correspond to the target domain. 93

5.5 Results for the traffic signs classification in the semi-supervised setting. *Syn* and *Real* denote available labeled data (100,000 synthetic and 430 real images respectively); *Adapted* means that ≈ 31,000 unlabeled target domain images were used for adaptation. The best performance is achieved by employing both the labeled samples and the large unlabeled corpus in the target domain. 96

6.1 **SPIRAL** takes as input either random noise or images and iteratively produces plausible samples or reconstructions via graphics program synthesis. The first row depicts an unconditional run given random noise. The second, third and fourth rows depict conditional execution given an image with a handwritten character, the Mona Lisa, and objects arranged in a 3D scene... 103

6.2 **The SPIRAL architecture. (A)** An execution trace of the SPIRAL agent. The policy outputs program fragments which are rendered into an image at each step via a graphics engine $\mathcal{R}$. The agent can make use of these intermediate renders to adjust its policy. The agent only receives a reward in the final step of execution. **(B)** Distributed training of SPIRAL. A collection of actors (in our experiments, up to 64), asynchronously and continuously produce execution traces. This data, along with a training dataset of ground-truth renderings, are passed to a Wasserstein discriminator on a separate GPU for adversarial training. The discriminator assesses the similarity of the final renderings of the traces to the ground-truth. A separate off-policy GPU learner receives batches of execution traces and trains the agent's parameters via policy-gradients to maximize the reward assigned to them by the discriminator, *i.e.*, to match the distribution of the ground truth dataset..... 104

6.3 Illustration of the **agent's action space** in the `libmypaint` environment. We show three different strokes (red, green, blue) that can result from a single instruction from the agent to the renderer. Starting from a position on the canvas, the agent selects the coordinates of the next end point, the coordinates of the intermediate control point, as well as the brush size, pressure and color. See Section 6.4.2 for details..... 111

6.4 **MNIST. (A)** A SPIRAL agent is trained to draw MNIST digits via a sequence of strokes in the `libmypaint` environment. As training progresses, the quality of the generations increases. The final samples capture the multi-modality of the dataset, varying brush sizes and digit styles. **(B)** A conditional SPIRAL agent is trained to reconstruct using the same action space. Reconstructions (left) match ground-truth (right) accurately..... 112

6.5 **Omniglot. (A)** A SPIRAL agent is trained to draw Omniglot characters via a sequence of strokes in the `libmypaint` environment. As training progresses, the quality of the generations increase. The final samples capture the multi-modality of the dataset, varying brush sizes and character styles. **(B)** A conditional SPIRAL agent is trained to reconstruct using the same action space. Reconstructions (left) match ground-truth (right) accurately. 113

6.6 **Image parsing** using the SPIRAL agent trained on Omniglot. All images from test sets. Given a rastered input (a), the agent produces a reconstruction (b) that closely matches the input. (c) Having access to the underlying strokes, we can render the character at a higher resolution, or in a different stroke style. (d) The agent effectively parses the input image into strokes. Each stroke is depicted in a separate color (we show average across 100 samples). . 114

6.7 **CelebA reconstructions.** The SPIRAL agent reconstructs human faces in 20 strokes. Although blurry, the reconstructions closely match the high-level structure of each image, for instance the background color, the position of the face and the color of the person's hair. In some cases, shadows around eyes and the nose are visible. 116

6.8 **3D scene reconstructions.** The SPIRAL agent is trained to reconstruct 3D scenes by emitting sequences of commands for the `MuJoCo` environment. In each pair, the left image corresponds to the model's output while the right one is the target. Our method is capable of acurately inferring the number of objects, their locations, sizes and colors. 116

6.9 **ℓ^2-distance between reconstructions and ground truth** images over the course of training. Across all datasets, we observe that training using a discriminator leads to significantly lower ℓ^2-distances, than when directly minimizing ℓ^2. We also show in (A) that the SPIRAL agent is capable of reconstructing even when it does not have access to the renderer in intermediate steps, however this does lead to a small degradation in performance. 117

6.10 **Blocked Metropolis-Hastings (MCMC) *vs* SPIRAL.** The MuJoCo Scenes dataset has a large combinatorial search space. We ran a general-purpose MCMC algorithm with object based blocked proposals and SPIRAL

on 100 holdout images during inference time. SPIRAL reliably processes every image in a single pass. We ran the MCMC algorithm for thousands of evaluations but it was unable to solve the task. 117

A.1 A toy experiment illustrating the **difference between ℓ^2 and discriminator** training in practice. **(A)** We collect a dataset of images with a single solid circle in all possible locations (top) and pick one of them as a target image (bottom). **(B)** The ℓ_2-distance (in the pixel space) from the input images to the target as a function of the circle location; the surface is flat around the borders since the circles do not overlap. **(C)** We train a discriminative model D that takes a pair of images and tells whether they match or not; just like ℓ^2, the resulting function has a pit centered at the target location, but unlike (b), the surface now has better behaved gradients. . . . . A-ii

A.2 The architecture of the **policy network** for a single step. `FC` refers to a fully-connected layer, `MLP` is a multilayer perceptron, `Conv` is a convolutional layer and `ResBlock` is a residual block. We give the dimensions of the output tensors in the square brackets. ReLU activations between the layers have been omitted for brevity. . . . . A-iv

A.3 The architecture of the **autoregressive decoder** for sampling an element a^i_{t+1} of the action tuple. The initial hidden vector z_0 is provided by an upstream LSTM. Depending on the type of the subaction to be sampled, we use either the `scalar` or the `location` branch of the diagram. . . . . A-v

LISTE DES SIGLES ET DES ABRÉVIATIONS

ANN	Artificial neural network
AP	Average precision
BPTT	Backpropagation through time
CFW	Coarse-to-fine warping
CNN	Convolutional neural network
CV	Computer vision
DA	Domain adaptation
DACG	Directed acyclic computational graph
DANN	Domain-adversarial neural network
FCN	Fully-convolutional neural network
GAN	Generative adversarial network
GMM	Gaussian mixture model
GRL	Gradient reversal layer
HMM	Hidden Markov model
kNN	k-nearest neighbor, nearest neighbor
LSTM	Long short-term memory
MCMC	Markov chain Monte Carlo

ML	Machine learning
MLP	Multi-layer perceptron
MS	Multi-scale
MSE	Mean squared error
NN	Neural network
ODS	Optimal dataset scale
OIS	Optimal image scale
RKHS	Reproducing kernel Hilbert space
PBT	Population-based training
RF	Random forest
RL	Reinforcement learning
SA	Subspace alignment
SE	Structured edge detector
SGD	Stochastic gradient descent
SS	Single-scale
ST	Spatial transformer
STN	Spatial transformer network
SVM	Support vector machine

ACKNOWLEDGEMENTS

My Ph.D. has been a challenging but also a very fulfilling journey. I wouldn't be able to achieve this milestone without the help of many great people with whom I had a great pleasure to interact along the way.

I want to extend my special gratitude to my co-advisors, Victor Lempitsky and Yoshua Bengio. Victor introduced me to Machine Learning and Computer Vision and through that helped me find my passion in life. Yoshua has always been an inspiration and provided immense support in the later stages of my studies. Thanks to him, I managed to cross the finish line.

I wish to thank my colleagues at Skoltech and Mila with whom I had many illuminating conversations about research and beyond. DeepWarp would not exist if not for the hard work of Daniil Kononenko and Diana Sungatullina. I'm grateful to DeepMind for giving me the opportunity to do an internship there back in 2017. Thanks to my co-authors, Tejas Kulkarni, Igor Babuschkin, Ali Eslami, and Oriol Vinyals, a fun internship project turned into a publication of which I'm really proud. I'm also thankful to Koray Kavukcuoglu for letting me focus on fiddling with neural networks without having to worry about the time constraints.

Thanks to Mike Chrzanowski, Faruk Ahmed, and Junyoung Chung for proofreading my thesis and giving valuable feedback. And thanks to the awesome staff of Caravane Cafe, who made my thesis writing experience much less painful and even enjoyable (to the extent possible).

I'm dedicating this work to my family, for their constant support and enabling my success.

Chapter 1

INTRODUCTION

Over the past 5-8 years deep *artificial neural networks* (ANNs) have become ubiquitous in the field of *computer vision* (CV). Although the basic machinery behind this family of models was first introduced in the mid-1950s (McCulloch and Pitts, 1943; Rosenblatt, 1958) and adapted for the visual domain in the early 1980s (Fukushima and Miyake, 1982), it was not widely adopted by the research community and practitioners until several decades later. Major challenges in training neural networks for larger-scale tasks (*e.g.*, vanishing and exploding gradients (Hochreiter, 1991; Bengio et al., 1994; Hochreiter et al., 2001), underfitting) outweighed the appealing promise of automatically discovered distributed representations.

Two key factors that led to the ultimate success of ANNs (recently rebranded as *deep learning*) were the enormous growth of the amount of available data (Russakovsky et al., 2015) and computational firepower (Raina et al., 2009; Krizhevsky et al., 2012; Cireşan et al., 2012). Together with architectural (Glorot et al., 2011b; Simonyan and Zisserman, 2014; Szegedy et al., 2015; He et al., 2015a) and algorithmic (Bengio et al., 2007; Kingma and Ba, 2014; Ioffe and Szegedy, 2015) advances, these allowed the researchers to not only experiment with larger models but also iterate over ideas at a much faster pace.

As a result, cumbersome hand-engineered vision pipelines are rapidly becoming a thing of the past. They are being replaced by purely data-driven end-to-end deep learning systems that require minimal human intervention when a new task is considered. Flexible and elegant, these new systems demonstrate state-of-the-art performance on a variety of computer vision benchmarks (*e.g.*, Krizhevsky et al. (2012); He et al. (2015b); Ganin and Lempitsky (2015); Girshick (2015); Long et al. (2015); Zbontar and LeCun (2016) to name a few).

In this thesis, we show how one can use ANNs to achieve excellent results in challenging image processing and synthesis tasks. We also present a method for leveraging large amounts of unlabeled synthetic data in the absence of proper training annotation, which is a common scenario in the CV domain.

1.1. Structure of the Thesis

This document is structured as follows. **Chapter 2** gives a short introduction into the relevant disciplines such as machine learning and computer vision. We pay a special attention to ANNs and particularly to *convolutional neural networks* (CNNs) since they are used extensively for solving various tasks discussed in the thesis.

Chapter 3 describes a general-purpose architecture for difficult image processing operations, such as natural edge detection or thin object segmentation. The architecture is based on a simple combination of convolutional neural networks with the nearest neighbor search.

We focus on situations when the desired image transformation is hard for a neural network to learn explicitly. We show that in such cases, the use of a hybrid model employing nearest neighbor search allows us to improve the results considerably and to account for underfitting in the network. This approach is validated on three challenging benchmarks, where the performance of the proposed method matches or exceeds the prior art.

In **Chapter 4**, we consider the task of synthesizing highly realistic images of a given face with a redirected gaze. We treat this problem as a specific instance of conditional image generation and suggest a new deep architecture that can handle this task very well as revealed by a numerical comparison with the state-of-the-art and a user study. Our deep architecture performs coarse-to-fine warping with an additional per-pixel intensity correction. All these operations are stacked into a single feed-forward network, and the parameters associated with different modules are learned jointly in an end-to-end fashion. The resulting model is capable of synthesising images with manipulated gaze, while the redirection angle can be selected arbitrarily from a certain range and provided as an input to the network.

Chapter 5 presents an adversarial domain adaptation framework useful for a wide variety of machine learning tasks beyond image processing and computer vision in general.

Top-performing deep networks are usually trained on massive amounts of labeled data. In the absence of such data, domain adaptation provides an attractive option given that annotated data, of similar nature but from a different domain, is available. For example, one could use training images synthesized with graphics software. Unfortunately, the resulting non-adapted network will likely underperform on the real data it is originally intended for. We propose a new approach to domain adaptation in deep architectures that can alleviate this situation by training the model on a large amount of labeled data from the source domain and a large amount of unlabeled data from the target domain.

As the training progresses, our method promotes the emergence of "deep" features that are both discriminative for the main learning task on the source domain and invariant to the domain shift. We show that this adaptation behaviour can be achieved in almost any feed-forward model by augmenting it with an additional branch. The augmented architecture is then trained via standard backpropagation.

Overall, our approach can be implemented with little effort using any deep learning package. The method performs very well in a series of image classification experiments, achieving an adaptation effect in the presence of significant domain shifts and outperforming previous state-of-the-art on the Office datasets.

In **Chapter 6** we use the core principle behind our domain adaptation technique to train a new kind of a generative model for visual data.

Advances in deep generative networks have led to impressive results in recent years. Nevertheless, state-of-the-art models can often waste their capacity on the minutiae of datasets, presumably due to weak inductive biases in their decoders. This is where graphics engines may come in handy since they abstract away low-level details and represent images as high-level programs.

Current methods that combine deep learning and renderers are limited by hand-crafted likelihood or distance functions, a need for large amounts of supervision, or difficulties in scaling their inference algorithms to richer datasets. **Chapter 6** presents a new framework that mitigates these issues. At its heart lies an adversarially trained agent, SPIRAL, that generates a program which is executed by a graphics engine to interpret and sample images. The goal of this agent is to fool a discriminator network that distinguishes between real and rendered data, trained with a distributed reinforcement learning setup without any supervision.

A surprising finding is that using the discriminator's output as a reward signal is the key to allow the agent to make meaningful progress at matching the desired output rendering. To the best of our knowledge, this is the first demonstration of an end-to-end, unsupervised and adversarial inverse graphics agent in challenging real-world (MNIST, Omniglot, CelebA) and synthetic 3D datasets.

Finally, **Chapter 7** gives several potential avenues for the future work and concludes the thesis.

Chapter 2

PRELIMINARIES

2.1. Machine Learning

Machine learning (shortened to *ML*) is a subfield of computer science dealing with algorithms capable of learning from data. Although it is difficult come up with a clear definition of what learning from data means, we usually restrict ourselves to the situations where there is a specific task T and some method of obtaining a set of training experiences E which we believe can be useful for solving T and we would like to design a computer program that improves its performance on T (as measured by a predefined metric P) as it consumes E (Mitchell et al., 1997). Machine learning is often contrasted with so-called "hand-engineered" solutions which define a sequence of interpretable steps computing the desired output for given input data I. While the steps performed by a ML program can sometimes be interpreted by humans, it is an optional property rather than a strict requirement. Developers of such algorithms still write standard program code but the goal is quite different: this code defines how to modify the algorithm given a new experience E in such a way it produces better output for I. In order to motivate the ML approach, we present several key drawbacks of manually designed systems:

- *For some tasks it is really difficult to come up with a clear algorithm that achieves the goal.* For example, one may want to classify photos of different breeds of cats. Even if the breeds are very different in appearence and it's relatively easy to describe how to distinguish between them using human language, it's not immediately obvious how to translate this knowledge into an executable code operating with the RGB representation of photos (see Figure 2.1).
- *Lack of robustness.* In the previous example, suppose that an expert managed to write a program that works well for several studio photos she has in her collection. This program is likely to fail for mobile phone shots because the expert might have

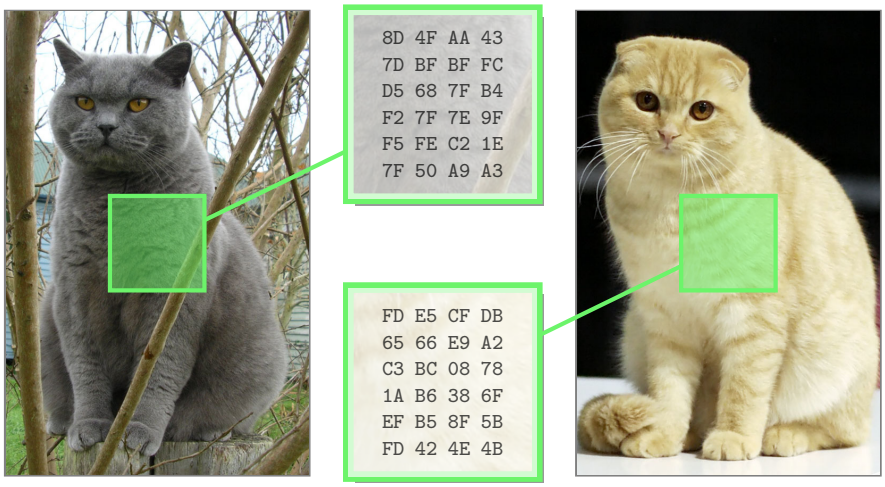


FIGURE 2.1. Some tasks are quite **easy for humans** but **very hard for machines**. For instance, in image classification, the computer has to deal with a raw RGB or binary representation of visual data (shown in the middle) that has almost no structure.

overlooked the effect of lighting conditions, viewpoint and image compression to the appearence of the same breed.

- *Lack of scalability.* For any new task (even if it is related to the previous ones) we have to manually devise a new set of rules. This makes our system unsuitable for the scenarios when the goals are frequently changing. For instance, after we have dealt the cat classifier somebody might ask us to develop a similar algorithm for dog breeds and then for flowers.

Although the examples above are related to vision (which is the main focus of this thesis), the same issues arise in other fields, *e.g.* natural language processing or speech recognition.

Arguably, the easiest way to grasp the essence of machine learning is to consider several typical classes of tasks. We can roughly divide ML algorithms into three categories (Hinton et al., 2012a): *supervised*, *unsupervised* and *reinforcement* learning algorithms. We are going to briefly describe each of those types below.

2.1.1. Supervised Learning

Supervised learning is the most common type of a machine learning task. In this setting, one is given a set of training tuples $\{(\mathbf{x}_i, \mathbf{y}_i)\}$, where $\mathbf{x}_i$ corresponds to the input of the algorithm and $\mathbf{y}_i$ denotes the desired output for that particular input. Both $\mathbf{x}_i$ and $\mathbf{y}_i$ can be as simple as scalars or real-valued vectors but they also may have a very rich structure (*e.g.*,

images or graphs). Naturally, the goal of supervised learning is to learn a mapping $\mathbf{x} \to \mathbf{y}$. Some typical examples of such tasks include:

- **Classification.** In this case, the output $\mathbf{y}$ can be treated as a discrete class from $\{1, \ldots, K\}$, where K is the total number of classes. Our example about distiguishing between different breeds of cats falls into this category. We further discuss this problem in Section 2.2.1.
- **Regression** deals with prediction of continuous scalars or vectors. For example, counting the number of cats in a given photo can be posed as a regression problem. One distinctive feature of this task that makes it different from classification is that target values are usually obey some kind of order (*e.g.*, 3 cats is more than 2, 2 is more than a half-cat trying to run away).
- In **structured prediction**, machine learning models predict more complex objects like graphs or text. If we would like to find a contour of a cat in a photo, that would be a structured prediction task.

2.1.2. Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning dealing with sequential decision making (Sutton and Barto, 2016; Schulman, 2016). A typical RL setting is when some *agent* is situated in an *environment* which provides a numerical reward signal that the agent must maximize by performing various *actions*. A concrete example given in (Sutton and Barto, 2016) is *tic-tac-toe*. Here, the agent is one of the players (let's assume it plays with `X`) and at every turn it can place its mark (it's an action) in one of the empty cells following some strategy that is called a *policy*. The environment contains everything that is not under the agent's control, including the other player. The reward is assumed to be zero during the match, positive if `X`s managed to win and negative otherwise. This *delayed feedback* is a distinctive feature of RL making it very different from the standard supervised learning which requires immediate feedback for every decision made by the system. In other words, in RL, we only need to specify the ultimate goal but not the actions that lead to that goal. This simultaneously makes reinforcement learning quite challenging but on the other hand opens up possibilities for discovering interesting and unexpected strategies that can even surpass human-level performance (Silver et al., 2016). Moreover, for some tasks, collecting a ground-truth sequence of decisions is nearly impossible thus ruling out direct application of supervised learning.

We give a short formal introduction in reinforcement learning in Section 2.4.

2.1.3. Unsupervised Learning

Unsupervised learning differs from previously discussed categories in that the learner only experiences inputs $\{\mathbf{x}_i\}$ and has to extract useful information without any external guiding signal. One major aim of unsupervised learning is to create an internal representation of the input that is useful for subsequent supervised or reinforcement learning (Hinton et al., 2012a). It can be used both to compress data into low-dimensional vectors and to extract structured (*e.g.*, sparse) high-dimensional descriptors. Among common unsupervised learning tasks are the following:

- In **synthesis and sampling**, the goal is to generate new examples that are similar to those in the training data, *e.g.*, we could expand our photocollection of cats and may be even invent new breeds by feeding the pictures that we have into a learning algorithm.
- **Density estimation.** In this problem, we would like to model the probability density function of the data distribution. Roughly speaking, given a new photo of a cat, we want to tell how likely it is to find this photo in our collection.
- **Clustering** is the task of grouping objects in such a way that objects of the same group are more similar (in some sense) than the objects from different groups. A clustering algorithm might, for example, form clusters comprised of a single cat breed without knowing anything about the actual breeds and only relying on the visual data.

We will consider an example of unsupervised learning in Chapter 6.

2.1.4. Learning and Optimization

In a typical learning scenario, we aim to maximize some performance measure P for an unseen set of examples (also called *test set*) that is not used to tune our algorithm. Naturally, because we assume we don't have an access to that set (and oftentimes, it's physically the case), we can only formulate the learning objective through the available training experiences (examples) E. The most common objective is just an expectation over the training set distribution $\hat{p}_{data}$:

$$J(\theta) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{data}}\mathcal{L}(f(\mathbf{x};\theta), y)\,, \tag{2.1.1}$$

where θ are parameters of our learning algorithm f that we perform optimization over, and $\mathcal{L}$ is some surrogate *loss function* which we try to minimize hoping it would maximize P (P itself can be hard to formalize, not to mention optimize). If our $\hat{p}_{data}$ is comprised of a finite

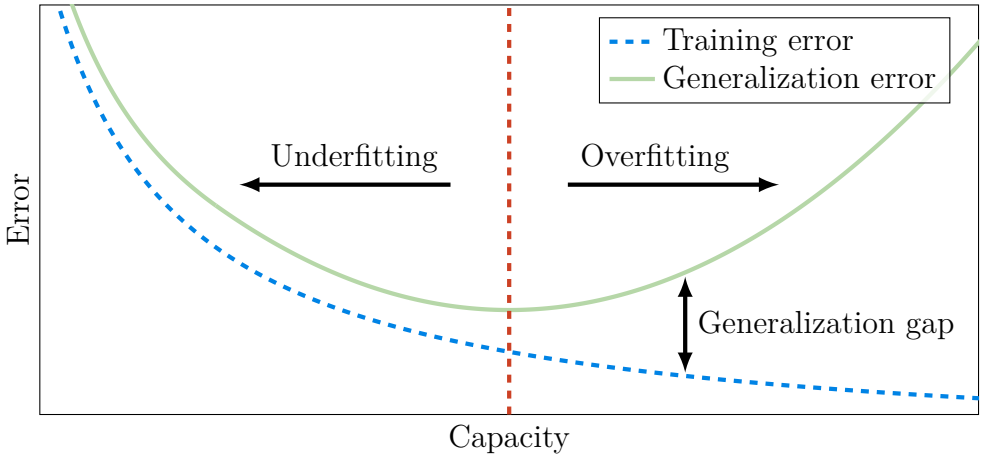


FIGURE 2.2. Relationship between **underfitting**, **overfitting** and **capacity**. Increasing capacity tends to reduce overfitting but increases the effect of overfitting. In the overfitting regime, the generalization gap is typically large so the training error poorly reflects the performance on the test set. We seek to find the optimal capacity (marked with a dashed red line) corresponding both to small generalization gap and to low training error. See Section 2.1.5 for details.

number of examples (say, N), we can rewrite (2.1.1) as

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}^{(i)}; \theta), y^{(i)}) \,. \tag{2.1.2}$$

This expression is called *empirical risk*. Although we optimize (2.1.1), our end goal is to obtain the lowest possible *true risk* J^* (also called *generalization error*), *i.e.*, expectation over the true data generating distribution (indeed, we'd like our algorithm to perform well in all *reasonable* scenarios). By evaluating f on the test set, we get an estimate of that quantity.

2.1.5. Capacity, Overfitting, Underfitting and Regularization

The difference between J and J^* is called *generalization gap*. Usually, we expect the empirical risk to be lower than the true risk, and thus, a large gap between these two values means that our learner has figured out some peculiarities of the training set that are not really useful in general (*e.g.*, if the dataset is small, it could just memorize the correct outputs for all the examples). This phenomenon is called *overfitting* and is closely related to the notion of the model *capacity*. Capacity defines which functions can be well-approximated by the elements from the chosen search (hypothesis) space. For example, a neural net with at least one hidden layer is known to be a universal function approximator (see Section 2.3.1). Overfitting normally occurs when the family of functions in which we perform search is too

rich given the amount of training data and/or the task complexity[1]. At the other extreme, we could fail to construct a model that has enough capacity to reasonably approximate the true data generating process. This regime is called *underfitting*. Here, both J and J^* are typically large. Figure 2.2 show the relationship between all three aforementioned notions. We note, that overfitting and underfitting are not mutually exclusive effects and can be observed within the same model (see Section 3.3.3).

There exist several ways of controlling the model's capacity. In the context of neural networks that will be discussed below, capacity (which can be formally measured by the *VC dimension* (Vapnik and Chervonenkis, 2015)) can be related to the number of trainable weights. We can make our network more powerful, for example, by using wider layers or increasing depth. Overfitting is addressed by replacing fully-connected layers with convolutions, which is a smart way of reducing the number of parameters. These are all examples of so-called model *hyperparameters*, *i.e.*, knobs and dials that are used to define the hypothesis space but not optimized during the actual training process.

Additionally, capacity is sometimes restricted by means of *regularization*. Regularization is a set of techniques for giving the learning algorithm a preference for one solution over another (Goodfellow et al., 2016). For example, widely used *weight decay* biases learning algorithms towards choosing parameters with smaller ℓ^2 norms. In general, regularization aims to reduce the generalization error without increasing the training error too much.

2.1.6. Covariate Shift and Domain Adaptation

Up until now, we have been in the setting when both the training and the test sets were generated from a single distribution p_{data}. In some cases, we want our model to perform well on the test data even if its distribution is different from what we have encountered during training. For example, we might have large amounts of labeled synthetically rendered images of some object types but the final model is intended for the recognition of their real counterparts. Here, a typical assumption that we make is that p_{train} and p_{test} are related through the *covariate shift* (Shimodaira, 2000; Jiang, 2008), *i.e.*, $p_{train}(\mathbf{x}) \neq p_{test}(\mathbf{x})$, while $p_{train}(y \,|\, \mathbf{x}) = p_{test}(y \,|\, \mathbf{x})$. It may seem that the covariate shift does not significantly change the situation comparing to what we considered above as we are often interested in learning conditional distributions $p(y \,|\, \mathbf{x})$ rather than joint. However, this becomes a problem when we deal with *misspecified parametric models* (which we do most of the time). In such models, no setting of the parameters θ reveals the true conditional and, in general, the optimal θ^*_{train}

[1]Here, by the task complexity we mean the complexity of the family containing the true data generating function.

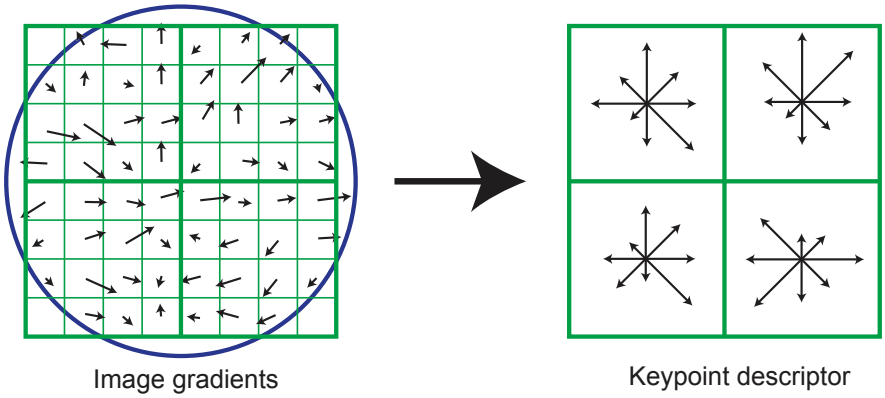


FIGURE 2.3. Computation of the **SIFT** keypoint descriptor. Gradients for each the small square cells are weighted by a Gaussian window (blue circle) and accumulated into 4 histograms. The histograms are then concatenated to form the final descriptor. See Section 2.2.1 for details. Figure credit: Lowe (2004).

under $p_{train}(\mathbf{x})$ does not match θ^*_{test} under $p_{test}(\mathbf{x})$. Thus, we have to devise methods for reducing the effect of the shift. Development of such methods is one of the main focuses of the field called *domain adaption*. In Chapter 5, we give a more detailed overview of the problem and also propose an effective method for solving the task in the case of deep neural networks.

2.2. Computer Vision and Image Processing

Computer vision (CV) is an interdisciplinary field that deals with automatic understanding of digital images and video. It is believed to have emerged in 1966 as a consequence of a failed summer project at the MIT AI laboratory (Papert, 1966). The participants of the project had a goal of constructing a significant part of a visual system but quickly realized that the task was far less straight-forward than it was thought in the beginning. Since that time CV researchers have been working on a large variety of small restricted chunks of the original goal. Some of the typical examples of the CV tasks are: object recognition (which we will discuss below) and detection, semantic segmentation, 3d reconstruction and depth estimation, medical imaging and many others.

2.2.1. Image Classification

One of the most common computer vision tasks is undoubtedly *image classification* (sometimes also referred to as *object recognition*) which we already mentioned in Section 2.1 as an example of a problem that can hardly be solved by conventional hand-engineering. Formally, it is a special case of the supervised learning scenario with a training set containing images (usually in the RGB format), each annotated with a label from some finite collection. For example, a popular toy dataset CIFAR10 (Krizhevsky and Hinton, 2009) consists of 60,000 tiny images that are 32 pixels high and wide. Those images are associated with one of 10 classes, like "automobile", "airplane", "dog", etc. As the name of the task suggests, the goal is to assign correct classes to previously unseen images which are assumed to come from the same distribution as the training examples.

There exist many approaches to solving image classification. One of the simplest and naive ones is non-parametric[2] *nearest neighbor* search (kNN). In this case, no real learning is performed — the classifier takes a test image and compares it to every single element of the training set using some distance measure, like L^1 or L^2. Since the topology of the raw pixel space poorly reflects semantic relations between objects in the images, kNN is rarely appropriate for direct image classification. Nonetheless, in the scenarios when data is relatively low-dimensional, nearest neighbor can be quite effective. We demonstrate this in the context of edge detection in Chapter 3.

Up until 2012 when convolutional neural networks (see Section 2.3.3) demonstrated excellent performance in the ILSVRC (Russakovsky et al., 2015; Krizhevsky et al., 2012), a typical object recognition pipeline was predominantly composed of three steps (Chatfield et al., 2011):

1. Extraction of local features;
2. Aggregation of local features into global image descriptors;
3. Classification of the resulting descriptors.

These steps were performed in isolation from each other with no information flowing from the subsequent to the preceding stages. Moreover, local features and aggregation strategies were mostly designed by hand. This is in contrast to neural networks which attempt to combine all the stages into a single end-to-end learning system and thus are much more flexible. Nevertheless, it is still important to understand how classical computer vision pipelines work as it gives insights into the success of the modern approaches. Below, we give a short introduction into each of the previously listed stages.

[2]Unlike parametric methods, non-parametric ones do not make any strong assumptions on the form of the $\mathbf{x} \to \mathbf{y}$ mapping. They can become more and more complex as the amount of training data increases.

Local features and quantization. One the most widely used methods for computing local image representations is *SIFT* (*scale-invariant feature transform*) developed by David Lowe in 1999 (Lowe, 1999, 2004). Although the full algorithm includes keypoint detection and matching, here we are only focusing on the description step as it is most relevant to the object recognition task.

The SIFT feature is computed by surrounding the point of interest by an appropriately rotated and scaled 8×8 neighborhood (see Figure 2.3). The scale and the orientation of the point are assumed to be given. For each of the 256 cells, one calculates the gradient magnitude and direction. This information is then aggregated into 8-bin histograms for four non-overlapping 4×4 blocks constituting the neighborhood. Intuitively, those histograms characterize how strong the gradient is for a specific quantized direction. The final descriptor is obtaned by concatenating the aforementioned histograms. This pointwise "footprint" is motivated by the work of Hubel & Wiesel (Hubel and Wiesel, 1959, 1962) who found that complex cells in the primary visual cortex are sensitive to the orientation and strength of edges but insensitive to their precise spatial location.

Some other popular keypoint descriptors are SURF (Bay et al., 2006), BRIEF (Calonder et al., 2010) and BRISK (Leutenegger et al., 2011). In case of object recognition, descriptors are typically extracted densely (*i.e.*, for each pixel) for one or more prespecified scales and then compressed using PCA (Ke and Sukthankar, 2004).

In order to gain additional robustness, we often wish to partition the local descriptor space into informative regions whose internal structure is irrelevant to the task and can be safely discarded or simplified. The most common way to do this is to either perform k-means clustering or fit a mixture of Gaussians (GMM). The latter defines soft data-to-cluster assignments (Bis, 2006):

$$q_{ki} = \frac{p(\mathbf{x}_i \,|\, \mu_k, \Sigma_k)\pi_k}{\sum_j^K p(\mathbf{x}_i \,|\, \mu_j, \Sigma_j)\pi_j}, \quad k \in \{1, \ldots, K\}, \tag{2.2.1}$$

where $\{\mathbf{x}_i\}$ is a set of extracted descriptors, $\{(\pi_k, \mu_k, \Sigma_k)\}$ are parameters of the mixture, and K is the number of clusters (often called *visual words*). In this context, k-means and GMM are collectively called *descriptor quantization*.

Feature aggregation. Quantized local descriptors are then *encoded* into a single feature vector for each image. In the simplest case, they are just combined into a bag-of-visual-words (histogram). For a long time, the winning entries for such object recognition competitions as PASCAL VOC (Everingham et al., 2007) and ILSVRC (Russakovsky et al., 2015) were based on so-called *Fisher encoding* (Perronnin and Dance, 2007). The basic idea behind this approach is to compose feature vectors out of the average first and second order differences

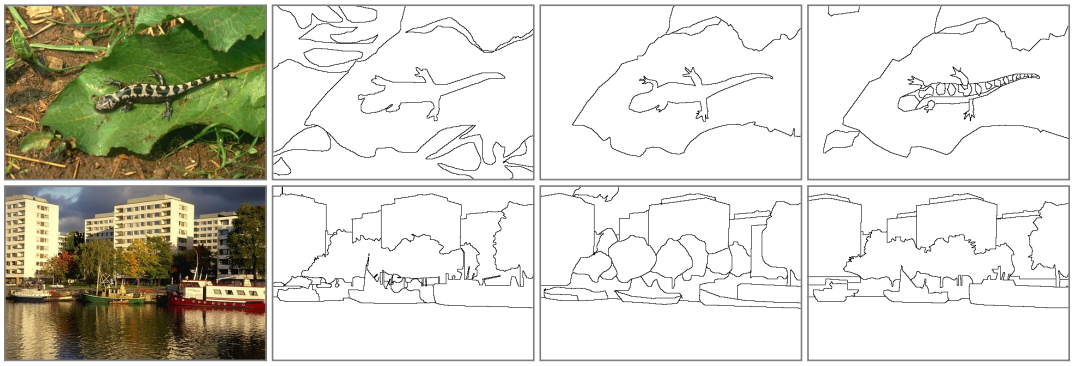


FIGURE 2.4. Training data for the **edge detection** task. Images are taken from the BSDS500 (Arbeláez et al., 2011) dataset. Each photo (on the left) has multiple ground-truth segmentations produced by different human annotators following the same guidelines. The ambiguity of task is reflected in a significant disagreement in annotations. Figure credit: Arbeláez et al. (2011).

between the point descriptors and the centers of a GMM. More concretely, the encoding is defined as:

$$\mathbf{f}_{\text{Fisher}} = \left[\mathbf{u}_1^T, \mathbf{v}_1^T, \ldots, \mathbf{u}_K^T, \mathbf{v}_K^T\right]^T , \tag{2.2.2}$$

where

$$\mathbf{u}_k = \frac{1}{N\sqrt{\pi_k}} \sum_i^N q_{ik} \Sigma_k^{-\frac{1}{2}} (\mathbf{x}_i - \mu_k) , \quad \mathbf{v}_k = \frac{1}{N\sqrt{2\pi_k}} \sum_i^N q_{ik} \left[(\mathbf{x}_i - \mu_k)^T \Sigma_k^{-1} (\mathbf{x}_i - \mu_k) - 1 \right] \tag{2.2.3}$$

with $\{q_{ki}\}$ given by (2.2.1).

In order to introduce weak geometry into the bag-of-words representation, the aforementioned aggregation is sometimes performed within several subregions of the image (*e.g.*, a pyramid of spatial bins). The obtained encodings are then concatenated to form the final feature vector.

Image descriptor classification. Descriptor classification can be performed by any machine learning algorithm such as *random forests* or *support vector machines* (SVM). We note that for SVMs it is crucial to normalize data vectors. As feature vectors usually have high dimensionality, it is sufficient to use linear kernels but sometimes it is possible to gain an additional boost in performance (without sacrificing computational efficiency) by applying special feature maps (Vedaldi and Zisserman, 2012) (*e.g.*, Hellinger kernel).

2.2.2. Image Processing

Image processing is a family of methods and problems closely related to computer vision in which both input and output data are images (Konushin, 2017). Formally, we are concerned

with functions f such that

$$O = f(I)\,, \quad I \in \mathbb{R}^{H_I \times W_I \times C_I}\,, O \in \mathbb{R}^{H_O \times W_O \times C_O}\,, \tag{2.2.4}$$

where $H_I \times W_I$ are the spatial dimensions of the image we would like to process, and $H_O \times W_O$ are the extents of the result. The third dimension corresponds to the number of channels (*e.g.*, typical photos have 3 — R(ed), G(reen) and B(blue)). It should be noted that some algorithms that are traditionally considered as image processing (*e.g.*, image compression) do not exactly fit the definition above. We leave them out of the scope of the present thesis.

Image processing is often used to enchance input data so that it looks aesthetically more pleasing or more entertaining as seen by humans. One other important application is preprocessing for downstream machine analysis tasks. Here, we'll briefly consider an example of such preprocessing called *edge detection*. Edges are curves in the image across which brightness or color changes abruptly. They are caused by shadow boundaries, countours of objects, or changes in surface orientation or reflectance (Tomasi, 2006). Edges are considered to be crucial for scene and object understanding and have been studied since the very formation of the computer vision discipline (Roberts, 1963; Papert, 1966). Standard algorithms for edge detection include Sobel and Prewitt filters as well as the famous Canny edge detector (Canny, 1986). A serious drawback of the aforementioned methods is that they only rely on low-level cues such as color and intensity of images and don't take semantics into account. As a result, their outputs are usually very cluttered, containing an excessive amount of non-informative edges which complicates the subsequent analysis.

Modern edge detection techniques attempt to solve that issue by incorporating learning (Arbeláez et al., 2011; Dollár and Zitnick, 2013) (see (Dollár and Zitnick, 2013) for a review). One of the most popular datasets/benchmarks for the task is BSDS500 (Arbeláez et al., 2011). It contains several hundreds of photos along with ground-truth contours provided by human annotators (see Figure 2.4). Each tested algorithm is required to provide a "soft" binary mask of edges which is then thresholded and compared against reference "hard" binary masks. Different thresholding strategies give rise to different quality metrics. One possibility is to use a fixed threshold for all images in the dataset, calibrated to provide optimal performance on the training set. In this case, the metric is called *optimal dataset scale* (ODS). The *optimal image scale* (OIS) metric is obtained when the optimal threshold is selected by an oracle on a per-image basis (Arbeláez et al., 2011).

In Chapter 3, we present a CNN-based algorithm which is one the earliest examples of how purely data-driven approaches can surpass hand-engineering at the edge detection task.

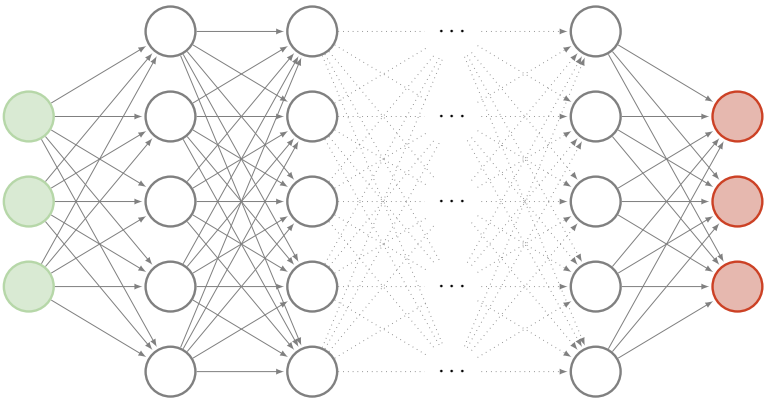


FIGURE 2.5. **Multi-layer perceptron** (MLP). Each hidden neuron activation is computed as a non-linear function of a weighted sum of all neuron activations from the previous layer (hence the dense connection patterns in the diagram). See Section 2.3.1 for details.

2.3. NEURAL NETWORKS

In this section, we describe a powerful family of machine learning models that recently gained a lot of popularity and were behind many advancements in such fields as natural language processing, speech recognition and, of course, computer vision.

2.3.1. Feed-forward Neural Networks

A *feed-forward neural network* is simply any parametrized model that can be represented as a composition of differentiable (or sub-differentiable) functions. A special case of such networks which used to be quite widespread in computer vision systems has a linear structure and can be described by the following expression:

$$f(\mathbf{x}; \theta_1, \theta_2, \ldots, \theta_n) = (f_n \circ f_{n-1} \circ \ldots \circ f_1)(\mathbf{x}) = f_n(\ldots(f_2(f_1(\mathbf{x}; \theta_1); \theta_2); \ldots; \theta_n), \quad (2.3.1)$$

where $\theta = \{\theta_i\}$ is a set parameters, n is called *depth*, while $\{f_i\}$ are usually referred to as *layers* (with f_n being an *output* layer and the rest being *hidden*). Individual components of a vector function f_i, $i \in \{1, \ldots, n-1\}$ are called *hidden units*. The parameters θ are also known as the *weights* of the network. This term is a reference to *synaptic weigths* in biological neural nets found in living organisms. Artificial neurons, although inspired by the real neurons, are quite different from their biological counterpart. For example, they communicate real values rather than discrete spikes of activity. Nevertheless, ANNs have been proven to work suprisingly well despite the idealization and simplifications.

Probably, the simplest example of the feed-forward network with a linear structure is the *linear perceptron*. It has a single affine layer of the form:

$$f(\mathbf{x}; w, b) = \mathbf{x}^T w + b\,, \tag{2.3.2}$$

where b is called *bias*. In order to use the perceptron above for binary classification, its output is thresholded at 0, *i.e.*, positive responses are associated with class 1, and negative with class 0. Perceptrons were popularized by Rosenblatt in early 1960's (Rosenblatt, 1958). At first, they appeared to be a very powerful family of models, however, later, in 1969, Minsky and Papert showed significant limitations of linear perceptrons. For instance, they are incapable of learning the `XOR`-function. Linear models are still quite useful when we can map our input into an approprite high-dimensional space. In practice, one usually computes a large amount of non-linear hand-crafted features hoping that the resulting feature vectors are approximately linearly separable (see Section 2.2.1).

An alternative (and admittedly, cleaner) way of solving the issue is to let the model itself decide which features might be useful for disentangling the input. This can be achieved by stacking several non-linear vector-valued perceptrons on top of each other, giving a concrete form of (2.3.1):

$$f_i(\mathbf{h}; W_i, b_i) = g(W_i^T \mathbf{h} + b_i)\,,\ \forall i \in \{1, \ldots, n\}, \qquad f(\mathbf{x}) = f_n \circ \ldots \circ f_1(\mathbf{x})\,, \tag{2.3.3}$$

where g is an *activation function* (non-linearity). This model is called a *multi-layer perceptron* (MLP) (see Figure 2.5). Some of the most widely used non-linearites are:

- **Sigmoid**. Computed element-wise as $g(z) = \sigma(z) = \frac{1}{1+\exp(-x)}$. This is a popular choice for network output units as it can be interpreted as probability of presence of some pattern. Sigmoids are also an integral part of gating mechanisms in recurrent neural networks. One unpleasant property of such non-linearity is *saturation*, *i.e.*, when the pre-activation z has a large absolute value, the gradient of the sigmoid *w.r.t.* z becomes very small significantly slowing down learning (see Section 2.3.2 for details on learning the ANNs).
- **Hyperbolic tangent** (Tanh) (LeCun et al., 1989b, 2012). Computed element-wise as $g(z) = \tanh(z) = 2\sigma(2z) - 1$. This is basically a sigmoid activation function rescaled to have range of $[-1, 1]$. It is somewhat similar to the identity function in the vicinity of 0 as $g'(0) = 1$ making it convenient for theoretical analysis. It is frequently used in recurrent neural networks, such as LSTMs (Hochreiter and Schmidhuber, 1997).
- **Rectified linear units** (ReLUs) (Nair and Hinton, 2010; Glorot et al., 2011b). Computed element-wise as $g(z) = \max(0, z)$. This seems to be the default recommendation

for modern neural networks. It has been shown to perform well for a wide range of tasks (*e.g.*, (Krizhevsky et al., 2012)) and also fairly easy to train.

- **Softmax**. Computed for the entire vector of pre-activations as:

$$g(\mathbf{z}) = \text{softmax}(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_{i=1}^{k} \exp(z_i)}\,, \quad \mathbf{z} = [z_1, \ldots, z_k]^T\,. \tag{2.3.4}$$

  Softmax is typically used as an output unit in multi-class classification problems. As in the case of binary sigmoid, $g(\mathbf{z})$ can be treated as parameters of a discrete probability distribution (indeed, it's easy to see that components of $g(\mathbf{z})$ sum to 1). Softmax is also used in attention (Bahdanau et al., 2014) and addressing mechanisms (Graves et al., 2014).

It can be shown that a feed-forward network with a linear output layer and at least one hidden layer with any "squashing" non-linearity (*e.g.*, hyperbolic tangent) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network has enough hidden units (Hornik et al., 1989).

In general, feed-forward networks are not restricted to have the linear (chain) structure but can be an arbitrary composition of functions described by a *directed acyclic computational graph* (DACG).

2.3.2. Optimization in Neural Networks

Backpropagation. Given a neural network $f(\cdot;\theta)$ taking $\mathbf{x}$ and producing an output $\hat{\mathbf{y}}$ as well as some scalar cost function J that depends on f, we wish to find θ^* such that

$$\theta^* = \arg\min_{\theta} J(f(\cdot;\theta))\,. \tag{2.3.5}$$

One possible strategy, in this case, is to perform some kind of *gradient descent*, assuming that J is a differentiable function of the network outputs and θ. It turns out that we can compute gradients of J efficiently using a *dynamic programming* algorithm called *backpropagation* (Rumelhart et al., 1988) (or simply *backprop*). This name is used to contrast with the process of computing $\hat{\mathbf{y}} = f(\mathbf{x},\theta)$ which is known as *forward propagation* (or forward pass).

In the nutshell, backpropagation is an implementation of the *chain rule* of calculus that avoids unnecessary recomputations by carefully caching previously obtained intermediate activations and executing operations in a specific order. Let's consider a single layer f_i in (2.3.1):

$$\mathbf{h}_i = f_i(\mathbf{h}_{i-1};\theta_i)\,, \tag{2.3.6}$$

where $\mathbf{h}_{i-1}$ is the layer's input. The derivatives of J *w.r.t.* $\mathbf{h}_{i-1}$ and θ_i are expressed as

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{h}_{i-1}} &= \frac{\partial J}{\partial \mathbf{h}_i} \cdot \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}\,, \\ \frac{\partial J}{\partial \theta_i} &= \frac{\partial J}{\partial \mathbf{h}_i} \cdot \frac{\partial \mathbf{h}_i}{\partial \theta_i}\,. \end{aligned} \tag{2.3.7}$$

If we assume that *Jacobian* $\frac{\partial J}{\partial \mathbf{h}_i}$ is given, and the value of $\mathbf{h}_{i-1}$ is cached from the forward pass, then both formulas above can be computed fast without the need to run through the entire network. This suggests an efficient strategy for obtaining derivatives for all the parameters: we start from the top layer and iteratively apply (2.3.7) until we reach the input of the network. The same algorithm works for arbitrary DACGs if we traverse the graph in topological order. Thus, the backward pass has the same computational complexity as the forward pass. This algorithm is, of course, much faster and more accurate than a naive finite differences method[3] which is still handy for debugging purposes.

Stochastic gradient descent (SGD). Backpropagation only offers a way for computing gradients but is not a full learning algorithm. In order to give a concrete example of such algorithm, we first need to define the cost J. Typically, one considers the empirical risk defined by (2.1.2). *Stochastic gradient descent* (SGD) optimizes (2.1.2) by sampling subsets of size m (*minibatches*) from $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ and iteratively performing the following parameter update:

$$\theta \leftarrow \theta - \varepsilon \cdot \left[\frac{1}{m} \sum_{k=1}^{m} \mathcal{L}(f(\mathbf{x}^{(i_k)}; \theta), y^{(i_k)})\right], \tag{2.3.8}$$

where ε is the *learning rate*. There exist many variants of this algorithm: momentum (Rumelhart et al., 1988), RMSProp (Tieleman and Hinton, 2012), Adam (Kingma and Ba, 2014), to name a few.

2.3.3. Convolutional Neural Networks

Convolutional neural networks (LeCun et al., 1989a) are one of the main building blocks of many recent successful computer vision algorithms. The main component of such models is a special layer called *convolutional layer*. Given an input 3-dimensional tensor of network activations I of size $H \times W \times C_I$, the output value of such a layer at the location (i, j) for the channel k is computed as

$$O(i, j, k) = (I * K)(i, j, k) = \sum_{p=0}^{H_K-1} \sum_{q=0}^{W_K-1} \sum_{r=0}^{C-1} I(i+p, j+q, r) \cdot K(p, q, r, k)\,, \tag{2.3.9}$$

[3]We can compute an approximate derivative *w.r.t.* θ_i by evaluating $[J(\theta + \varepsilon\,\mathbf{e}_i) - J(\theta)]/\varepsilon$, where $\mathbf{e}_i$ is a basis vector corresponding the i-th component of θ and ε is a small positive scalar.

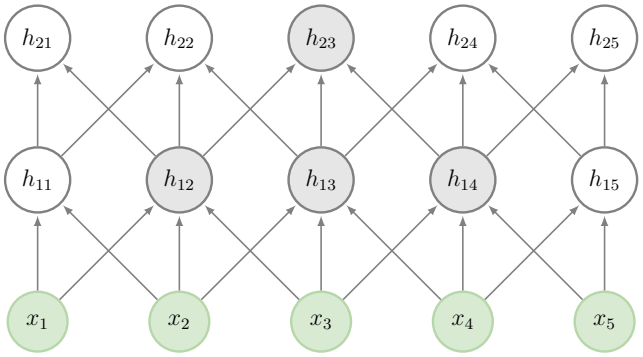


FIGURE 2.6. **Receptive field** of a neuron in a higher layer of a convolutional neural network interacts with a large portion of the input despite local connectivity between adjacent layers.

where K is tensor of size $H_K \times W_K \times C_I \times C_O$ called a *kernel*. Here we are considering the case of multi-channel 2D data (*e.g.* image domain), but the operation above is easily generalised to higher dimensionalities. It should be noted that (2.3.9), strictly speaking, does not correspond to a discrete finite-support convolution but rather to a *cross-correlation*. In order to obtain a true convolution, one has to transpose the kernel. This loose use of terminology within the deep learning community can be explained by the absence of any advantage of picking actual convolution over cross-correlation – we rarely care about commutability and moreover, the orientation of a learned K has no special meaning.

Intuitively, convolution can be seen as a feature detection mechanism. Indeed, we can treat a slice of the kernel tensor $K(\cdot, \cdot, \cdot, k)$ as a specific pattern we wish to find in the input I. The search is then performed by sliding that pattern over I and comparing it against every spatial location by means of a dot product (unnormalized cosine distance). As a result, $O(\cdot, \cdot, k)$ becomes a detection map with large values in positions where the feature appears in the input. This observation is closely related to a property of convolution called *equivariance* to translation. Formally, a function $f(x)$ is equivariant to a function $g(x)$ is $f(g(x)) = g(f(x))$. It's easy to see that applying convolution to an input I' with a pattern shifted from its original location (i, j) by $(\Delta i, \Delta j)$ yields the activation $(I * K)(i, j, k)$ at $(i + \Delta i, j + \Delta j)$.

Due to *local connectivity* of neurons and *weight sharing*, convolutions require far less memory to store the parameters and faster to compute comparing to regular fully-connected layers. One other interesting implication of weight sharing is that it makes the model dramatically more statistically efficient.

One may wonder if replacing full matrix multiplications with convolutions would prevent the model from capturing interactions between distant locations in the input. While this is a

serious issue for shallow models, in deeper networks with many convolutions stacked on top of each other, units at higher levels may still interact indirectly with large portions of the input (see Figure 2.6).

In software implementations of convolution, special care has to be taken when dealing with border pixels. In order to control the spatial dimensions of the output, one typically enlarges the input by means of zero padding. Depending on the amount of padding, we usually distiguish three types of convolutions:

- **Valid.** This is a special case with no zero padding. The output size after convolution is $(H - H_K + 1) \times (W - W_K + 1)$.
- **Same.** In cases when one needs to preserve the spatial dimensions (*i.e.*, get the output of size $H \times W$), a padding of $(H_K - 1)/2$ and $(W_K - 1)/2$ pixels is added to the edges of y and x axes respectively.
- **Full.** This type of convolution is used when one needs every input pixel to influence an equal number (*i.e.*, $H_K \times W_K$) of output pixels. This requires $(H_K - 1) \times (W_K - 1)$ padding resulting in the output of size $(H + H_K - 1) \times (W + W_K - 1)$.

Sometimes it is desirable to skip over some positions of the kernel to reduce the computation cost (Goodfellow et al., 2016) and obtain more spatially compact representation of data. In this case, convolution is said to be *strided* and is computed as

$$O(i, j, k) = (I * K\,;\, s)(i, j, k) = \sum_{p=0}^{H_K-1} \sum_{q=0}^{W_K-1} \sum_{r=0}^{C-1} I(i \cdot s + p, j \cdot s + q, r) \cdot K(p, q, r, k)\,, \quad (2.3.10)$$

where s is the *stride*. This type of convolution has recently gained a lot of popularity as a replacement for pooling operations (Springenberg et al., 2014) (which we will discuss below) and has been found to be important in training good generative models such as VAEs or GANs (Radford et al., 2015).

Pooling and invariance. Much like in best performing hand-engineered features (see Section 2.2.1), robustness and partial *invariance* to small translations in convolutional neural networks are achieved by performing spatial *pooling*. In terms of implementation, pooling is similar to convolution except dot product is now replaced with a special function `pool`, *i.e.*,

$$O(i, j, k) = \texttt{pool}(I\,;\, s)(i, j, k) = \texttt{pool}\left(I([i \cdot s] : [i \cdot s + H_K], [j \cdot s] : [j \cdot s + W_K], \cdot)\right)\,, \quad (2.3.11)$$

where *colon* denotes tensor slice. There are several popular choices for the `pool` function including `max` (computing the maximum activation within the region), `avg` (average activation) and also L^p norm. Just like convolutions, pooling can be applied in a strided fashion, *i.e.*,

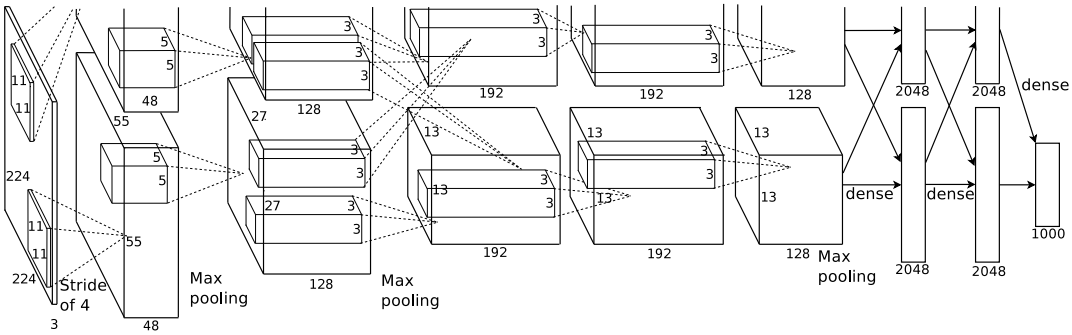


FIGURE 2.7. The `AlexNet` architecture (Krizhevsky et al., 2012) used by the researchers from the University of Toronto to win the ILSVRC2012 (Russakovsky et al., 2015). It's a typical example of a convolutional neural network described in Section 2.3.3. Figure credit: Krizhevsky et al. (2012).

$s > 1$ in (2.3.11). Usually, this does not result in a significant loss of information because pooling responses tend to change slowly when moving from one spatial location to another.

Pooling can also be performed along the channel axis. This way, it is possible to obtain invariances to a wider range of transformations. For example, if the network has learned to produce several detection maps each of which dedicated to a specific orientation of the same feature, then `max` pooling along the channel dimension gives a single map capturing presence or absence of the feature irrespectively of how it is rotated. This principle is extensively employed in *maxout networks* (Goodfellow et al., 2013). One other application of pooling is discussed in Section 4.4.6 where we use it to account for imperfect alignment between inputs and corresponding ground-truth outputs.

Convolutional networks for computer vision. Deep CNNs have recently achieved several breakthroughs in a variety of computer vision benchmarks and keep on attracting a very strong interest within the CV community. The most impressive results have been attained for image (Krizhevsky et al., 2012) and pixel (Ciresan et al., 2012) classification. Krizhevsky et al. (2012) with the architecture called `AlexNet` (see Figure 2.7) managed to win the ILSVRC2012 (Russakovsky et al., 2015) beating the competitors by a large margin. This effectively ended the domination of traditional CV pipelines that we discussed in Section 2.2.1. The CNN-based systems that have been developed over the past several years significantly raised the bar for such tasks as object detection (Girshick, 2015; Ren et al., 2015), semantic segmentation (Farabet et al., 2013; Long et al., 2015), depth estimation (Zbontar and LeCun, 2016), image captioning (Xu et al., 2015) and in some cases even surpassed the human performance (He et al., 2015b) rendering obsolete the former beliefs of what is hard to achieve with machine learning.

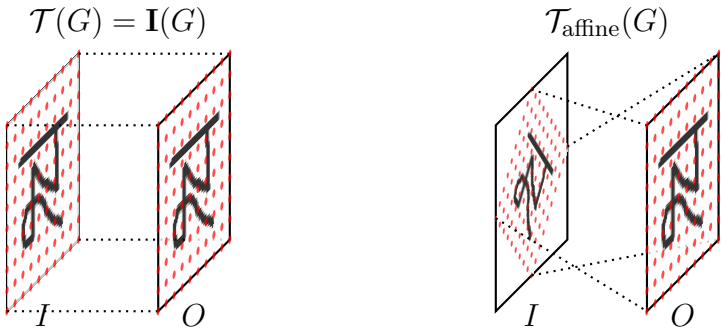


FIGURE 2.8. Two examples of transformations that can be performed by a **spatial transformer** module. The regular grid (red dots) in the output O is mapped to a sampling grid in the input I via operator $\mathcal{T}$. **(Left)** $\mathcal{T}$ is the *identity*. **(Right)** $\mathcal{T}$ is an *affine* transform.

For more references, please see Section 3.2 and Section 4.2.

Enhancing convolutional networks with special modules. Many recent state-of-the-art CV algorithms employing CNNs go beyond standard architectures stacking convolutions and pooling operations. Along with sophisticated wiring of layers (*e.g.*, skip-connections, multiple columns) (Krizhevsky et al., 2012; Cireşan et al., 2012; He et al., 2015a), a typical practice is to add special differentiable modules (Bahdanau et al., 2014; Santoro et al., 2017) that encode prior knowledge about the task at hand and can both speed up training and increase the effectiveness of the network.

One example of such modules that we will use in the context of edge detection (Chapter 4) is the *spatial transformer* (ST) proposed by Jaderberg et al. (2015). The main idea of the paper is to allow spatial manipulations of data within the CNN thus facilitating greater spatial invariance of the model. At the heart of this method lies *differentiable image sampling*.

Let I be the input of the ST module and O – its output. Both I and O are feature maps with the same number of channels but possibly of different spatial dimensions. The values of the output pixels lying on a regular $H' \times W'$ grid G are defined as follows:

$$O(i,j,k) = \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} I(m,n,k) \cdot K(\mathcal{T}(i,j) - (m,n)) \,, \tag{2.3.12}$$

where K is a *sampling kernel* and $\mathcal{T}$ is some transformation of G (Figure 2.8). The original work by Jaderberg et al. (2015) considers $\mathcal{T}$ coming from a restricted family parameterized by a low-dimensional vector (*e.g.*, affine or thin plate spline). In Chapter 4, we take a more direct approach and predict a displacement for each grid node.

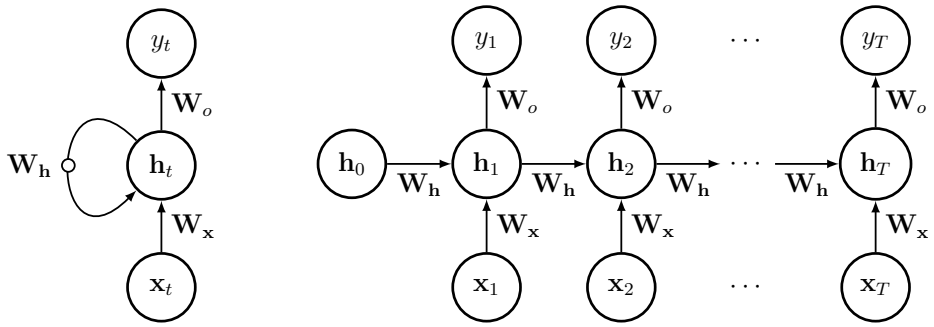


FIGURE 2.9. An example of a **simple single hidden layer RNN** taking a sequence of inputs $(\mathbf{x}_1, \mathbf{x}_2, \ldots)$ and producing a sequence of outputs $(y_1, y_2, \ldots)$. **(Left)** The folded representation. The small hollow circle indicates a delay of a single time step. **(Right)** The unfolded (unrolled) form of the same network. The RNN becomes a feed-forward model with its weights shared across different time steps.

A popular choice for K is a bilinear sampling kernel:

$$K(y, x) = \max(0, 1 - |y|) \cdot \max(0, 1 - |x|)\,. \tag{2.3.13}$$

Assuming that $(u, v) = \mathcal{T}(i, j)$, we can compute the partial (sub-)derivatives *w.r.t.* both the input features and the transformed grid locations as

$$\frac{\partial O(i,j,k)}{\partial I(m,n,k)} = \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} \max(0, 1 - |u - m|) \cdot \max(0, 1 - |v - n|)\,, \tag{2.3.14}$$

$$\frac{\partial O(i,j,k)}{\partial v} = \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} \max(0, 1 - |u - m|) \cdot \begin{cases} 0 & \text{if } |m - v| \geq 1 \\ 1 & \text{if } m \geq v \\ -1 & \text{if } m < v \end{cases}. \tag{2.3.15}$$

An analogous calculation can be done for $\frac{\partial O(i,j,k)}{\partial u}$.

2.3.4. Recurrent Neural Networks

Recurrent neural networks (RNNs) are a special kind of neural networks taylored for processing of sequential data or performing sequential decision making. The main principle behind RNNs is the same as in CNNs, namely, weight sharing. Unlike convolutional nets, however, recurrent neural networks make heavy use of feedback connections between the timesteps which simultaneously makes them more powerful but at the same time hinders efficient parallelization of computations.

In this section, without loss of generality we are restricting ourselves to the case of temporal sequences (*i.e.*, indexed by time steps $t \in \{0, 1, \ldots\}$) but the same principles are applicable to the types of data other than time-series (Kalchbrenner et al., 2015; Oord et al., 2016; Zilly et al., 2016). One of the most basic forms of RNNs was proposed by Elman (1990). Given a sequence of inputs $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots)$, the output y is obtained by repeatedly applying the following rule:

$$\mathbf{h}_t = \sigma(\mathbf{W}_\mathbf{x}\mathbf{x}_t + \mathbf{W}_\mathbf{h}\mathbf{h}_{t-1} + \mathbf{b}_\mathbf{h}) \,, \tag{2.3.16}$$

$$y_t = \sigma(\mathbf{W}_o\mathbf{h}_t + \mathbf{b}_o) \,, \tag{2.3.17}$$

where $\mathbf{h}$ is a sequence of *hidden states* of the RNN and $\mathbf{W}, \mathbf{b}$ with subscripts are learned weights and biases of appropriate dimensionalities. A schematic representation of this network is shown in Figure 2.9. Much like with perceptrons, one can increase the expressive power of the model by stacking multiple one-layer modules described above.

Backpropagation through time (BPTT). RNNs can be converted into feed-forward networks by means of *unfolding* (Goodfellow et al., 2016) (see an example of an unfolded Elman network in Figure 2.9). This makes it possible to obtain all necessary gradients using regular backpropagation which is, in this case, called *backpropagation through time* (BPTT) (Rumelhart et al., 1986; Werbos, 1988).

There are several notable difficulties with performing BPTT for regular RNNs. The first one is related to the computational and memory complexity of this operation — both grow linearly with the length of the sequence. A typical strategy to mitigate this problem is to chunk the sequence into regions of reasonable size and prevent the gradient from flowing through the boundaries of the regions. This approach is called *truncated backpropagation through time* (Williams and Peng, 1990).

The second difficulty stems from the fact that the update rule for the hidden state in (2.3.17) resembles repeated matrix multiplication. To see why this may create problems, consider a learning signal coming into the network at time step T, *i.e.* $\frac{\partial J}{\partial \mathbf{h}_T}$. Backpropagating it all the way to the beginning of the sequence yields

$$\frac{\partial J}{\partial \mathbf{h}_0} = \frac{\partial J}{\partial \mathbf{h}_T} \cdot \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdot \ldots \cdot \frac{\partial \mathbf{h}_1}{\partial \mathbf{h}_0} \,. \tag{2.3.18}$$

Taking into account (2.3.17) we get

$$\frac{\partial J}{\partial \mathbf{h}_0} = \frac{\partial J}{\partial \mathbf{h}_T} \cdot \prod_{i=0}^{T-1} \mathbf{W}_\mathbf{h}^\top \operatorname{diag}(\sigma'(\mathbf{v}_i)) \,, \tag{2.3.19}$$

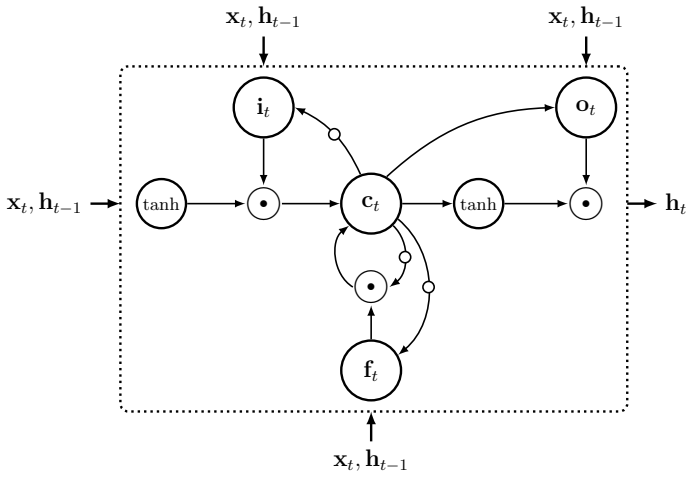


FIGURE 2.10. The architecture of an **LSTM** cell. The hollow circles indicate a delay of one time step. At the heart of the architecture lies the memory state $\mathbf{c}_t$ which contains an aggregate of the information the has flown through the cell so far. The gate $\mathbf{i}_t$ controls how much of the current input $(\mathbf{x}_t, \mathbf{h}_{t-1})$ is taken into account. The forget gate $\mathbf{f}_t$ allows to reset the memory state, and $\mathbf{o}_t$ limits the amount of the emitted signal.

where $\mathbf{v}_i = \mathbf{W}_\mathbf{x}\mathbf{x}_{i+1} + \mathbf{W}_\mathbf{h}\mathbf{h}_i + \mathbf{b}_\mathbf{h}$ and $\operatorname{diag}(\sigma'(\mathbf{v}_i))$ is a diagonalized derivative of the activation function. From the expression above, it is clear that if the *spectral radius* of $\mathbf{W}_\mathbf{h}$ is less than 4 (the derivative of the sigmoid is bounded by $\frac{1}{4}$) the training signal decays exponentially and eventually vanishes. The phenomenon of *vanishing gradients* (Hochreiter, 1991; Bengio et al., 1994) is believed to be one of the main reasons why learning long-term dependencies is hard in vanilla RNNs. On the other hand, if $\mathbf{W}_\mathbf{h}$ has large eigenvalues one can observe the opposite effect — *exploding gradients*, which are typically remedied by *gradient clipping* (Mikolov, 2012; Pascanu et al., 2013).

Long short-term memory (LSTM). In order to mitigate the problem of vanishing and exploding gradients, Hochreiter and Schmidhuber (1997) proposed a special recurrent network architecture called *long short-term memory* (LSTM). LSTM units employ a gating mechanism that allows them to carry information over long time spans, reset hidden states as well as selectively discard inputs. More concretely, the dynamics of such network is defined by the following set of formulas (Zaremba et al., 2014):

$$\left(\mathbf{f}_t^\top, \mathbf{i}_t^\top, \mathbf{o}_t^\top\right)^\top = \sigma(\mathbf{W}_\mathbf{x}\mathbf{x}_t + \mathbf{W}_\mathbf{h}\mathbf{h}_{t-1} + \mathbf{b}_\mathbf{h})\,, \tag{2.3.20}$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_{\tilde{\mathbf{c}},\mathbf{x}}\mathbf{x}_t + \mathbf{W}_{\tilde{\mathbf{c}},\mathbf{h}}\mathbf{h}_{t-1} + \mathbf{b}_{\tilde{\mathbf{c}}})\,, \tag{2.3.21}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t\,, \tag{2.3.22}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)\,, \tag{2.3.23}$$

where $\mathbf{f}_t, \mathbf{i}_t, \mathbf{o}_t$ are the *forget*, the *input*, and the *output* gates respectively, $\mathbf{c}_t$ is the *memory state*, and $\mathbf{h}_t$ is the output of the unit. A diagram depicting the architecture is shown in Figure 2.10. Although there have been proposed several other variants of recurrent units (Cho et al., 2014; Jozefowicz et al., 2015; Greff et al., 2017), LSTMs stood the test of time (Melis et al., 2017) and are still frequently used as a default option for sequential data. For that reason, in Chapter 6, our model employs LSTM for handling recurrent connections.

2.3.5. Adversarial Training of Neural Networks

In the *adversarial* framework, we seek to obtain a set of models $M_1, \ldots, M_k$ possessing certain properties by simultaneously optimizing competing objectives $\mathcal{L}_1, \ldots, \mathcal{L}_k$ respectively. One of the first machine learning works employing this principle was Arthur Samuel's checkers playing agent (Samuel, 1959). In the context of neural networks, Schmidhuber (1992) used a variant of adversarial training for learning of factorial codes.

Most recently, a new surge of interest in the framework has been sparked by *generative adversarial networks* (GANs)[4] (Goodfellow et al., 2014). The goal of GANs is modeling of some data distribution p_d. The architecture contains two networks, G and D, that are pitted against each other. The *generator* network G learns to synthesize samples that best resemble p_d, while the *discriminator* D is trained to distinguish between the outputs produced by G and real data.

More formally, D and G play the following two-player minimax game with value function $V(D, G)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_d}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z}))]\,, \tag{2.3.24}$$

where p_z is a source of randomness (usually taken to be either $\mathcal{N}(0, I)$ or $\mathcal{U}(0, I)$), and D's outputs are assumed to lie in the $[0, 1]$ range. In other words, $M_1 \coloneqq G$, $M_2 \coloneqq D$ and $\mathcal{L}_1 \coloneqq V(D, G)$, $\mathcal{L}_2 \coloneqq -V(D, G)$. It can be proven that at equillibrium, the generated distribution $p_g = G(\mathbf{z})$ matches the target p_d.

In practice, however, (2.3.24) may not provide sufficient gradient for G to learn well. To mitigate this problem, Goodfellow et al. (2014) proposed a non-saturating formulation of the game above which have the same fixed point of the dynamics of G and D but stronger training signal for G:

$$\mathcal{L}_1 = \mathbb{E}_{\mathbf{z}\sim p_z}[-\log(D(G(\mathbf{z}))]\,, \tag{2.3.25}$$

[4]In Chapter 5, we present an independent and concurrent work that shares the same foundational principles.

$$\mathcal{L}_2 = -V(D, G)\,. \tag{2.3.26}$$

Unfortunately, even in their non-saturating form GANs can be difficult to train (Arjovsky and Bottou, 2017). Arjovsky et al. (2017) demonstrate that a slightly different adversarial game derived from the Kantorovich-Rubinstein duality (Villani, 2008) for Wasserstein-1 distance can lead to better learning dynamics of the model:

$$\min_G \max_{D \in \mathrm{Lip}(1)} V(D, G) = \mathbb{E}_{\mathbf{x}\sim p_d}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z}[D(G(\mathbf{z}))]\,, \tag{2.3.27}$$

where Lip(1) is the set of 1-Lipshitz functions and D outputs arbitrary real numbers. We note that this new formulation has one major drawback, namely, it involves constrained optimization *w.r.t.* D. Enforcing the constraint is by no means trivial. One option is to clip weights of the network after each training step (Arjovsky et al., 2017) but this crude trick is shown to yield suboptimal results in several real-world scenarios (Gulrajani et al., 2017).

Gulrajani et al. (2017) proposed a more principled solution based on so-called *gradient penalty*. They observed that the optimal D should have a slope of 1 along the segments connecting optimally coupled pairs of points from p_g and p_d. This idea inspired the following variation of (2.3.24) dubbed WGAN-GP:

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_d}[D(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z}[D(G(\mathbf{z}))] - \lambda\, \mathbb{E}_{\mathbf{x}\sim p_i}[(\|\nabla D(\mathbf{x})\|_2 - 1)^2]\,, \tag{2.3.28}$$

where $\mathbf{x} \sim p_i$ is defined as

$$\mathbf{x} = \alpha\, \mathbf{x}_g + (1 - \alpha)\, \mathbf{x}_d\,, \quad \mathbf{x}_g \sim p_g\,, \quad \mathbf{x}_d \sim p_d\,, \quad \alpha \sim \mathcal{U}(0, 1)\,. \tag{2.3.29}$$

Several sebsequent papers explored the use of the gradient penalty and its modifications for a variety of GAN formulations and verified universal applicability of this method (Roth et al., 2017; Kodali et al., 2018; Dai et al., 2018). Those findings suggest that proper regularization is the key ingredient to making training of GANs stable and reliable.

In Chapter 6, we combine the WGAN-GP objective with basic techniques from reinforcement learning to devise a generative model employing external graphics simulators.

2.4. Deep Reinforcement Learning Basics

In this section, we give a basic technical background on RL necessary for understanding of the method described in Chapter 6.

As it was mentioned before, in reinforcement learning, we are usually concerned with finding a good policy for an agent acting in some environment. Formally, the agent `A` and the

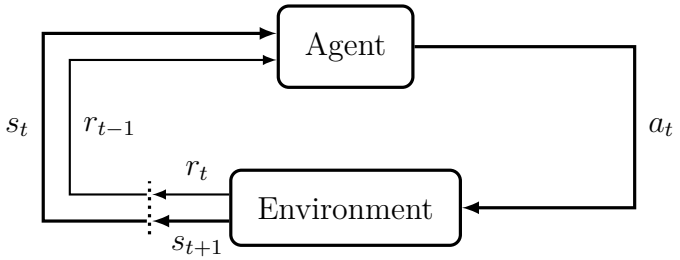


FIGURE 2.11. The **agent-environment** interaction in a Markov decision process. The agent receives the current state of the environment s_t and a reward r_{t-1} and performs an action a_t that changes the state of the environment to s_{t+1}.

environment `E` interact at each of a sequence of discrete time steps $t \in \{0, 1, \ldots\}$. The agent receives some representation of `E`'s state, s_t, and on that basis selects an action a_t which leads to `E` changing its state to s_{t+1} and emitting a scalar reward r_t for `A`. In the framework of *Markov decision processes*, the environment's state is assumed to contain all necessary information for decision making, *i.e.*, knowing the current state makes previous states and actions irrelevant. A sequence of triplets

$$(s_0, a_0, r_0), (s_1, a_1, r_1), \ldots \tag{2.4.1}$$

is called a *trajectory*. The performance of `A` is characterized in terms of expected *return*, *i.e.*, $\mathbb{E}[R_t]$, where $R_t = \sum_{i=t}^{T} r_i$, where T is the length of the trajectory. We operate with the expectation because both the agent and the environment are, in general, stochastic. In the case of finite T, the RL task is called *episodic*. For the *continuing* tasks ($T = \infty$), in order to ensure existence of R_t, we need to introduce the notion of the *discounted return*:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}\,, \tag{2.4.2}$$

where $\gamma \in [0, 1]$ is the *discount factor*.

The agent's policy $\pi(a|s)$ is defined as a mapping from an environment's state to probabilities of selecting each possible action. If `A` follows some π, the following quantity:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_t | s_t = s] \tag{2.4.3}$$

is called the *value* function of that policy. Similarly,

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a] \tag{2.4.4}$$

is the *action-value* function. An access to the latter quantity for an optimal policy, $Q_* \coloneqq Q^{\pi_*}$, gives a straight-forward way of recovering the policy itself: in each state, we just pick the

action corresponding to the highest action-value. One of the most popular ways of obtaining Q_* is *Q-learning* (Watkins, 1989). This algorithm was a backbone of the first successful application of deep reinforcement learning (Mnih et al., 2015).

In Chapter 6, we take a different approach. We represent π directly as a neural network parametrized by θ. If we take $J(\theta) = -V^\pi(s_0)$ as our objective (*i.e.*, we want to maximize expected return from the initial state) then using the *policy gradient theorem* (Sutton et al., 2000) we can establish that

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a Q^\pi(s,a) \nabla_\theta \pi(a|s)\,, \tag{2.4.5}$$

where $\mu(s)$ is a probability of being in state s while following policy π. Rearranging the expression above yields:

$$\begin{aligned}
\nabla J(\theta) &\propto \mathbb{E}_\pi \left[\sum_a Q^\pi(s_t,a) \nabla_\theta \pi(a|s_t) \right] = \\
&= \mathbb{E}_\pi \left[\sum_a \pi(a|s_t) Q^\pi(s_t,a) \frac{\nabla_\theta \pi(a|s_t)}{\pi(a|s_t)} \right] = \\
&= \mathbb{E}_\pi \left[Q^\pi(s_t,a_t) \frac{\nabla_\theta \pi(a_t|s_t)}{\pi(a_t|s_t)} \right] = \\
&= \mathbb{E}_\pi \left[R_t \, \nabla_\theta \log \pi(a_t|s_t) \right] .
\end{aligned} \tag{2.4.6}$$

We therefore can improve the agent's policy by sampling trajectories and performing gradient updates of the form:

$$\theta \leftarrow \theta + \lambda \, R_t \, \nabla_\theta \log \pi(a_t|s_t)\,, \tag{2.4.7}$$

where $\lambda > 0$ is a learning rate. Intuitively, this update increases the probability of taking action a_t according to how well the agent performed afterwards (the higher the return the larger the increase). The algorithm we have just presented is called *REINFORCE* and was first proposed by Williams (1992).

In their most basic form REINFORCE updates suffer from high variance which may significantly slow down training. The most common way to address this issue is to use a *baseline* for returns, *i.e.*, replace R_t with $R_t - b(s_t)$. The baseline b can be any function, even a random variable, as long as it does not depend on a (this ensures that the subtracted quantity is zero on average). In practice, b is typically taken to be a value function estimate: $b(s_t) = \hat{V}^\pi(s_t) \approx V^\pi(s_t)$.

In order to further decrease variance (albeit by introducing some bias), we can substitute full Monte-Carlo returns with their bootstrapped versions:

$$R_t = \sum_{k=0}^{n-1} \gamma^k \, r_t + \gamma^n \, \hat{V}^\pi(s_{t+n}) \, . \tag{2.4.8}$$

Here, we replace the tail of the sum with its estimate. This variation of REINFORCE is called the *advantage actor-critic* (A2C) algorithm (Sutton and Barto, 2016). One additional benefit of using A2C is that we do not need to run the episode until completion therefore updating the policy more frequently with smaller computational and memory footprint. Advantage actor-critic serves as basis for several recent large-scale RL frameworks, *e.g.*, A3C (Mnih et al., 2016) and IMPALA (Espeholt et al., 2018). In Chapter 6, we use the latter to perform efficient training of our generative model on a multi-node GPU cluster.

PROLOGUE TO THE FIRST ARTICLE

Article Details

N^4-Fields: Neural Network Nearest Neighbor Fields for Image Transforms. Yaroslav Ganin and Victor Lempitsky, *Proceedings of the 12th Asian Conference on Computer Vision 2014 (ACCV 2014).*

Personal Contributions. The idea of using a non-parametric nearest neighbor extension for a segmentation network I was developing at the time was first discussed in a meeting with Victor Lempitsky in the fall of 2013. I made all the design choices for the actual model (such as network architectures, the use of ensembling and so on), implemented it and ran all the experiments for the project. Victor Lempitsky and I worked on writing a draft of the paper for ECCV 2014. I mostly focused on the method description, experimental section and the related work, while Victor Lempitsky contributed the introduction and the conclusion sections as well as did final editing of the document. For the ACCV 2014 re-submission, I wrote a code for a fast test-time inference and ran additional baseline experiments on the NYU RGBD (Silberman and Fergus, 2011) and the DRIVE (Staal et al., 2004) datasets.

Context

Our work on the neural network-based edge detector started in the summer of 2013. At the time, all the existing state-of-art methods were were either prohibitively slow or required a large amount of memory.

Shortly after we began experimenting with several ideas, Dollár and Zitnick (2013) published a paper describing a very fast and efficient method for edge detection. Their approach employed random forests operating on a set of hand-engineered features. The quantitative results were excellent, and we decided to adopt some of the techniques from the paper in our

own project. As a result, we ended up with a system that relied on a specific description of the edge maps but used approximate nearest neighbour instead of random tree traversal. The latter allowed us to make our detector more flexible by replacing fixed data-independent features with a learned data-driven convolutional network.

The paper was rejected from ECCV 2014. Although we improved upon the state-of-the-art on the BSDS500 dataset (Arbeláez et al., 2011), the reviewers were not convinced with the results. Their main concerns were the justification of the chosen CNN architecture (a typical critique at the time) and the running time of the full pipeline. Moreover, we failed to emphasize the advantages of our method over the algorithm proposed by Dollár and Zitnick (2013). In the revised version of the manuscript, we addressed the latter by demonstrating poor suitability of the Structured Edge detector for the thin object segmentation task. Ultimately, our paper was accepted to ACCV 2014 as an oral presentation.

Contributions

To our knowledge, this work is the first neural network based approach to match or surpass the state-of-the-art on a range of standard edge detection (BSDS500, NYU RGBD) and thin object segmentation (DRIVE) datasets. We showed how one can improve the performance of a discriminatively trained parametric CNN by combining it with a non-parametric nearest neighbour search. This idea is revisited in some recent deep learning papers (Bansal et al., 2017). The key contribution, however, is that our paper demonstrates the superiority of fully-data-driven systems over pipelines employing hand-engineered features: we use the same architecture in all the experiments and let it extract information relevant to the task.

Recent Developments

After our paper was published in the proceedings of ACCV 2014, there has been several attempts to use neural networks for edge detection. Shen et al. (2015) choose to build upon ideas from a different "pre-neural" paper (Lim et al., 2013). Their approach involves two stages. In the first stage, a set of training patches is clustered into several bins representing different edge types. The problem is thus cast into ordinary classification which is solved by means of a CNN. In the second stage, the resulting network is used as a feature extractor for SE (Dollár and Zitnick, 2013).

Bertasius et al. (2015) employ a more direct approach combining a pre-trained `AlexNet` Krizhevsky et al. (2012) and a bifurcated network predicting both presence or absence of the edge at a particular location and the fraction of human annotators marked that location as positive. Just like N^4-fields and DeepControur (Shen et al., 2015), this method is patch-based.

The introduction of *fully-convolutional* networks (FCN) (Long et al., 2015) greatly advanced the performance of the edge detection and segmentation algorithms. The core idea is to apply a CNN with no fully-connected layers to the entire image. This gives a dense map of decisions for all the locations at once as opposed to a single decision in the case of patch processing. Backpropagating errors for the entire image is not only more efficient (due to reuse of computations) but also leads to faster and more robust training. The most notable methods employing this technique are HED (*Holistically-Nested Edge Detection* by Xie and Tu (2015)) and its variant developed by Kokkinos (2016). The former uses so-called *deep supervision* for a series of side outputs of different spatial resolutions. Kokkinos (2016) consider the aforementioned idea as a means of task simplification and propose gradual annealing of the side losses. Two other improvements over the original HED are the modified loss function inspired by the *Multiple Instance Learning* literature and multi-scale training. We note that N^4-fields also relies on multi-scale but only at inference time.

Chapter 3

NEURAL NETWORK NEAREST NEIGHBOR FIELDS FOR IMAGE TRANSFORMS

3.1. Introduction

In this work, we demonstrate that convolutional neural networks can achieve state-of-the-art results for sophisticated image processing tasks. The complexity of these tasks defies the straightforward application of patch-based CNNs, which perform reasonably well, but clearly below state-of-the-art. In particular, we show that by pairing convolutional networks with a simple non-parametric transform based on nearest-neighbor search state-of-the-art performance is achievable. This approach is evaluated on three challenging and competitive benchmarks (edge detection on Berkeley Segmentation dataset (Arbeláez et al., 2011), edge detection on the NYU RGBD dataset (Silberman and Fergus, 2011), retina vessel segmentation on the DRIVE dataset (Staal et al., 2004)). All the results are obtained with the same meta-parameters, such as the configuration of a CNN, thus demonstrating the universality of the proposed approach.

The two approaches, namely convolutional **N**eural **N**etworks and **N**earest **N**eighbor search are applied sequentially and in a patch-by-patch manner, hence we call the architecture N^4-*fields*. At test time, an N^4-field first passes each patch through a CNN. For a given patch, the output of the first stage is a low-dimensional vector corresponding to the activations of the top layer in the CNN. At the second stage we use the nearest neighbor search within the CNN activations corresponding to patches sampled from the training data. Thus, we retrieve a patch with a known pixel-level annotation that has a similar CNN activation, and transfer its annotation to the output. By averaging the outputs of the overlapping patches, the transformation of the input image is obtained.

FIGURE 3.1. N^4-Fields can be applied to a range of complex image processing tasks, such as natural edge detection (left) or vessel segmentation (right). The proposed architecture combines the convolutional neural networks with the nearest neighbor search and is generic. It achieves state-of-the-art performance on standard benchmarks for these two rather different applications with very little customization or parameter tuning.

Below, we first review the related works (Section 3.2), describe the proposed architecture and the associated training procedures in detail (Section 3.3), and discuss the results of applying it on sample problems (Section 3.4). We conclude with a short discussion of the merits and the potential of the proposed approach (Section 3.5).

3.2. Related Work

There is a very large body of related approaches, as both neural networks and nearest neighbor methods have been used heavily as components within image processing systems. Here, we only review several works that are arguably most related to ours.

Neural Networks for Image Processing. The use of neural networks for image processing goes back for decades (Egmont-Petersen et al., 2002). Several recent works have investigated large-scale training of deep architectures for complex edge detection and segmentation tasks. For instance, Mnih and Hinton (2010) have used a cascade of two deep networks to segment roads in aerial images, while Schulz and Behnke (2012) use CNNs to perform semantic segmentation on standard datasets. Kivinen et al. (2014) proposed using unsupervised features extraction via deep belief net extension of mcRBM (Ranzato and Hinton, 2010) followed by supervised neural net training for boundary prediction in natural images. State-of-the-art results on several semantic segmentation datasets were obtained by Farabet et al. (2013) by using a combination of a CNN classifier and superpixelization-based smoothing. Finally, a large body of work, *e.g.*, Jain et al. (2007); Ciresan et al. (2012) simply frame the segmentation problem as patch classification, making generic CNN-based classification easily

applicable and successful. Below, we compare N^4-fields against such baseline and find them to achieve better results for our applications.

Another series of works (Jain and Seung, 2008; Burger et al., 2012) investigate the use of convolutional neural networks for image denoising. In this specific application, CNNs benefit greatly from virtually unlimited training data that can be synthesized, while the gap between synthetic and real data for this application is small.

Neural networks have also been applied for descriptor learning, which resembles the way they are used within N^4-fields. Chopra et al. (2005) introduced a general scheme for learning CNNs that map input images to multi-dimensional descriptors, suitable among other things for nearest neighbor retrieval or similarity verification. The learning in that case is performed on a large set of pairs of matching images. N^4-fields are different from this group of the approaches in terms of their purpose (image processing) and the type of the training data (annotated images).

Non-parametric Approaches to Image Processing. Nearest neighbor methods have been applied to image processing with a considerable success. Most methods use nearest neighbor relations within the same image, *e.g.*, Dabov et al. (2008) for denoising or Criminisi et al. (2004) for inpainting. More related to our work, Freeman et al. (2000) match patches in a given image to a large dataset of patches from different images, to infer the missing high-frequencies and to achieve super-resolution. All these works use the patches themselves or their band-passed versions to perform the matching.

Another popular non-parametric framework to perform operations with patches are random forests. Our work was in many ways inspired by the recent impressive results in (Dollár and Zitnick, 2013), where random forests are trained on patches with structured annotations. Their emphasis is on natural edge detection, and their system represent the state-of-the-art for this task[1]. N^4-fields match the accuracy of (Dollár and Zitnick, 2013) for natural edge detection, and perform considerably better for the task of vessel segmentation in micrographs, thus demonstrating the ability to adapt to new domains.

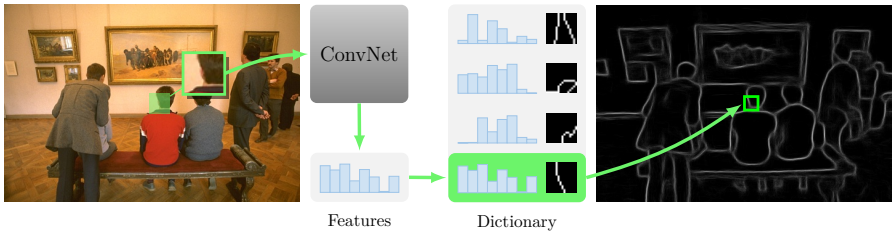


FIGURE 3.2. The N^4 architecture for natural edge detection. The input image is processed patch-by-patch. An input patch is first passed through a pretrained convolutional neural network (CNN). Then, the output of the CNN is matched against the dictionary of sample CNN outputs that correspond to training patches with known annotations. The annotation corresponding to the nearest neighbor is transferred to the output. Overall, the output is obtained by averaging the overlapping transferred annotations.

3.3. N^4-FIELDS

3.3.1. Architecture

We start by introducing the notation, and discussing the way our architecture is applied to images. The N^4-Fields transform images patch-by-patch. Given an image transform application, we wish to map a single or multi-channel (e.g. RGB) image patch $\mathcal{P}$ of size $M \times M$ to a segmentation, an edge map, or some other semantically-meaningful annotation $\mathbf{A}(\mathcal{P})$, which in itself is a single or multi-channel image patch of size $N \times N$. We take N to be smaller than M, so that $\mathbf{A}(\mathcal{P})$ represents a desired annotation for the central part of $\mathcal{P}$. For the simplicity of comparisons, in our experiments we use the sizes proposed in (Dollár and Zitnick, 2013), in particular, $M = 34$ and $N = 16$.

Given annotated data, we learn a mapping $\mathbf{F}$ that maps patches to the desired annotations. At test time, the mapping is applied to all image patches and their outputs are combined by averaging producing the final output. Formally, the resulting transformation of the input image I at pixel (x, y) is defined as:

$$\mathbf{F}(I)[x, y] = \frac{1}{N^2} \sum_{\substack{i,j:|i-x|\leq N/2 \\ |j-y|\leq N/2}} \mathbf{F}\left(I(i, j|M)\right)[x - i, y - i], \tag{3.3.1}$$

where $I(i, j|M)$ denotes the image patch of size $M \times M$ centered at (i, j), and $\mathbf{F}\left(I(i, j|M)\right)[x - i, y - i]$ is a pixel in the output patch at the position $(x - i, y - j)$ assuming the origin in the center of the patch.

[1]At the time of publication.

Obviously, the accuracy of the transform depends on the way the transform $\mathbf{F}$ is defined and learned. Convolutional neural networks (CNNs) provide a generic architecture for learning functions of the multi-channel images and patches exploiting the translational invariance properties of natural images. The direct approach is then to learn a mapping $\mathcal{P} \to \mathbf{A}(\mathcal{P})$ in the form of a CNN. In practice, we found the flexibility of CNNs to be insufficient to learn the corresponding mapping even when large models were considered. For complex transforms, e.g. natural edge detection, we observe a strong underfitting during the training, which results in a suboptimal performance at test time.

Convolutional neural network is a parametric model, albeit with a very large number of parameters. A straightforward way to increase the fitting capacity of the mapping is to consider an auxhillary non-parametric model. We thus combine a simple non-parametric mapping (nearest neighbor) and a complex parametric mapping (convolutional neural network). The input patch $\mathcal{P}$ is first mapped to an intermediate representation $\mathrm{CNN}(\mathcal{P};\Theta)$, where Θ denotes the parameters of the CNN. The output $\mathrm{CNN}(\mathcal{P};\Theta)$ of the CNN mapping (we call it a *neural code*) is then compared to a dictionary dataset of CNN outputs, computed for T patches $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_T$ taken from the training images, and thus having known annotations $\mathbf{A}(\mathcal{P}_1), \mathbf{A}(\mathcal{P}_2), \ldots, \mathbf{A}(\mathcal{P}_T)$. The input patch is then assigned the annotation from the dictionary patch with the closest CNN output, i.e. $\mathbf{A}(\mathcal{P}_k)$, where $k = \arg\min_{i=1}^{T} ||\mathrm{CNN}(\mathcal{P}_i) - \mathrm{CNN}(\mathcal{P})||$ (Figure 3.2). If we denote such nearest neighbor mapping as NNB, then the full two-stage mapping is defined as:

$$\mathbf{F}(\mathcal{P}) = \mathrm{NNB}\left(\mathrm{CNN}(\mathcal{P};\,\Theta) \mid \{(\mathrm{CNN}(\mathcal{P}_i;\Theta)\,;\,\mathbf{A}(\mathcal{P}_i)) \mid i = 1..T\}\right)\,, \tag{3.3.2}$$

where $\mathrm{NNB}(x \mid M = \{(a_i|b_i)\})$ denotes the nearest-neighbor transform that maps x to the value b_i corresponding to the key a_i that is closest to x over the dataset M. In our experiments, the dimensionality of the intermediate representation (i.e. the space of CNN outputs) is rather low (16 dimensions), which makes nearest neighbor search reasonably easy.

In the experiments, we observe that such two-stage architecture can successfully rectify the underfitting effect of the CNN and result in better generalization and overall transform quality compared to single stage architectures that include either CNN or nearest neighbor search on hand-crafted features alone.

3.3.2. Training

The training procedure for an N^4-field requires learning the parameters Θ of the convolutional neural network. Note, that the second stage (nearest neighbor mapping) does not require any training apart from sampling T patches from the training images.

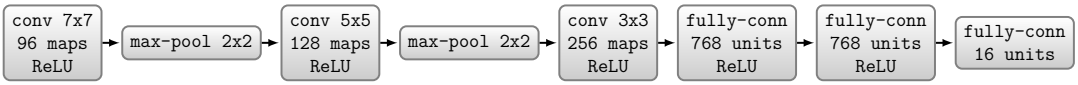


FIGURE 3.3. The CNN architecture used in our experiments. See Section 3.3.3 for details.

The CNN training is performed in a standard supervised way on the patches drawn from the training images $I_1, I_2, \ldots I_R$. For that, we define a surrogate target output for each input patch. Since for each training patch $\mathcal{P}$, the desired annotation $\mathbf{A}(\mathcal{P})$ is known, it is natural to take this annotation itself as a target (although other variants are possible as described in Section 3.3.3), *i.e.*, to train the network on the input-output pairs of the form $(\mathcal{P}, \mathbf{A}(\mathcal{P}))$. In this case, however, the output can be rather high-dimensional (when the output patch size is large) and vary non-smoothly *w.r.t.* small translations and jitter, in particular when our model applications of edge detection or thin object segmentations are considered. To address both problems, we perform dimensionality reduction of the output annotations using PCA. Experimentally, we found that the target dimensionality can be taken rather small, *e.g.*, 16 dimensions for 16×16 patches.

Thus, the overall training process includes the following steps:

1. Learn a PCA projection on a subset of $N \times N$ patches extracted from the training image annotations.
2. Train a convolutional neural network on the input-output pairs $\{(\mathcal{P}, \mathrm{PCA}(\mathbf{A}(\mathcal{P}))\}$ sampled from the training images.
3. Construct a dictionary $\{(\mathrm{CNN}(\mathcal{P}_i; \Theta); \mathbf{A}(\mathcal{P}_i))|i = 1..T\}$ by drawing T random patches from the training images and passing them through the trained network.

After training, the N^4-field can be applied to new images as discussed above.

3.3.3. Implementation Details

Training the CNN. We use a heavily modified variant of the `cuda-convnet` CNN toolbox[2]. The CNN architecture that was used in our experiments is loosely inspired by (Krizhevsky et al., 2012) (it is comprised of the layers shown in Figure 3.3). We also tried a dozen of other CNN designs (deeper ones and wider ones) but the performance always stayed roughly the same, which suggests that our system is somewhat insensitive to the choice of the architecture given the sufficient number of free parameters.

The model was trained on 34×34 patches extracted at randomly sampled locations of the training images. Each patch is preprocessed by subtracting the per-channel mean

[2] https://code.google.com/p/cuda-convnet/

(across all images). Those patches are packed into mini-batches of size 128 (due to the software/hardware restrictions) and presented to the network. The initial weights in the CNN are drawn from Gaussian distribution with zero mean and $\sigma = 10^{-2}$. They are then updated using stochastic gradient descent with momentum set to 0.9. The starting learning rate η is set to 10^{-1} (below in Section 3.4 we introduce an alternative target function which demands smaller initial $\eta = 10^{-3}$). As commonly done, we anneal η throughout training when the validation error reaches its plateau.

As the amount of the training data was rather limited, we observed overfitting (validation error increasing, while training error decreasing) alongside with underfitting (training error staying high). To reduce overfitting, we enrich the training set with various artificial transformations of input patches such as random rotations and horizontal reflections. Those transformations are computed on-the-fly during the training procedure (new batches are prepared in parallel with the network training).

Along with data augmentation we apply two regularization techniques which have become quite common for CNNs, namely dropout (Hinton et al., 2012b) (we randomly discard half of activations in the first two fully-connected layers) and ℓ^2-norm restriction of the filters in the first layer (Hinton et al., 2012b; Zeiler, 2012).

Testing Procedure. At test time, we want to calculate activations for patches centered at all possible locations within input images. A naive approach would be to apply a CNN in a sliding window fashion (separate forward pass for each location). However this solution may be computationally expensive especially in case of deep architectures. Luckily it is rather easy to avoid redundant calculations and to make dense applications efficient by feeding the network with a sequence of shifted test images (Sermanet et al., 2013).

After neural codes for all the patches are computed, we perform nearest-neighbor search using of k-d trees provided as a part of `VLFeat` package (Vedaldi and Fulkerson, 2008). We leave the default settings unchanged except for the maximum number of comparisons which is set to 30.

Our proof-of-concept implementation runs reasonably fast taking about 6 seconds to process an image of size 480×320, although we were not focusing on speed. Computational performance may be brought closer to the real-time by, for example, applying the system in a strided fashion (Dollár and Zitnick, 2013) and/or finding a simpler design for the CNN.

Multi-scale Operation. Following the works (Dollár and Zitnick, 2013; Xiaofeng and Bo, 2012) we apply our scheme at different scales. For each input image we combine detections produced at the original, half and double resolutions to get the final output. While various

blending strategies may be employed, in our case even simple averaging gave remarkably good results.

Committee of N^4-Fields. CNNs are shown (Krizhevsky et al., 2012; Sermanet et al., 2013; Ciresan et al., 2012) to perform better if outputs of multiple models are averaged. We found that this technique works quite well for our system too. One rationale would be that different instances of the neural network produce slightly different neural codes hence nearest-neighbor search may return different annotation patches for the same input patch. In practice we observe that averaging amplifies relevant edges and smoothens noisy regions. The latter is especially important for the natural edge detection benchmarks, as the output of N^4-fields is passed through the non-maximum suppression.

3.4. Experiments

We evaluate our approach on three datasets. In two of them (BSDS500 and NYU RGBD), the goal is to detect natural edges, and in the remaining case (DRIVE), the task is to segment thin vessels in retinal micrographs. Across all the datasets, we provide comparison with baseline methods, with the state-of-the-art on those datasets, illustrate the operation of the method, and demonstrate characteristic results.

CNN Baselines. All three tasks correspond to binary labeling of pixels in the input photographs (boundary/not boundary, vessel/no vessel). It is therefore natural to compare our approach to CNNs that directly predict pixel labels. Given an input patch, a CNN can produce a decision either for the single central pixel or for multiple pixels (*e.g.*, a central patch of size 16×16) hence we have two *CNN baselines*. We call them *CNN-central* and *CNN-patch* respectively. Each of the CNNs has the same architecture as the CNN we use within N^4-fields, except that the size of the last layer is no longer 16 but equals the number of pixels we wish to produce predictions for (*i.e.*, 1 for CNN-central and 256 for CNN-patch). At test time, we run the baseline on every patch and annotate chosen subsets of pixels with the output of the CNN classifier applying averaging in the overlapping regions. As with our main system, to assess the performance of the baseline, we use a committee of three CNN classifiers at three scales.

Nearest Neighbor Baseline. We have also evaluated a baseline that replaces the learned neural codes with "hand-crafted" features. For this, we used SIFT vectors computed over the input $M \times M$ patches as descriptors and use these vectors to perform the nearest-neighbor

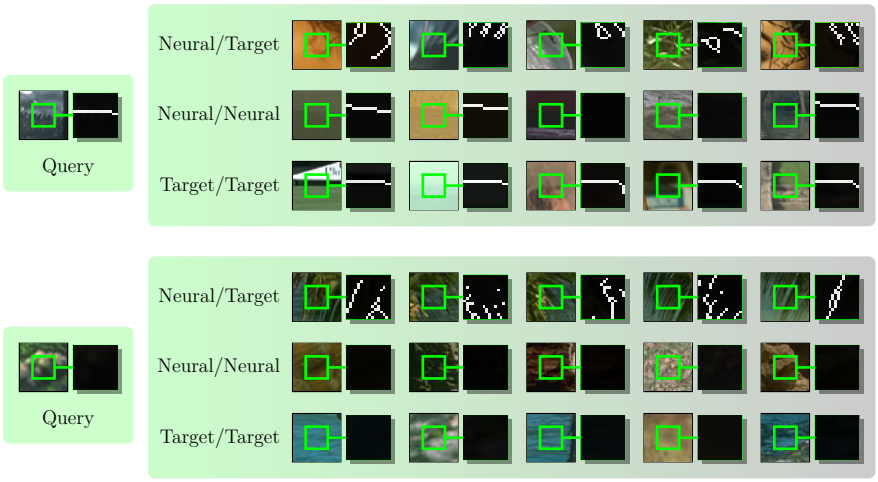


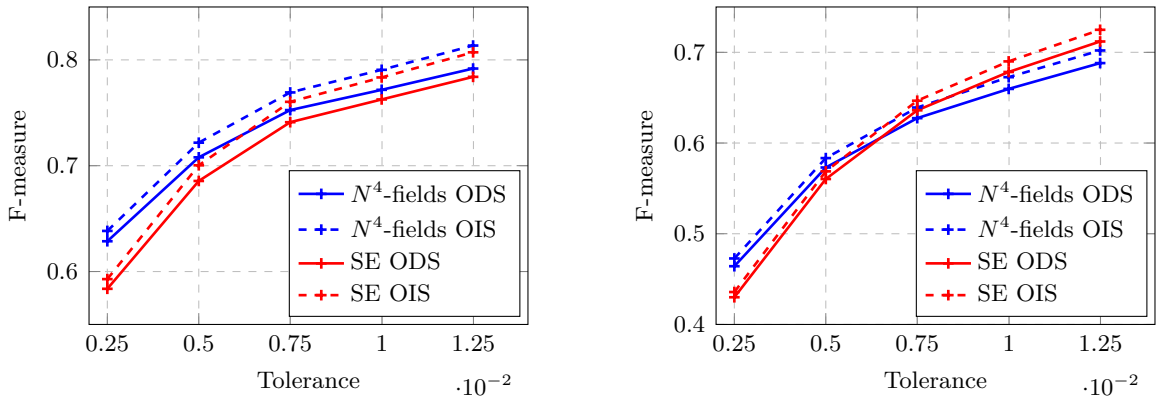
FIGURE 3.4. Examples of nearest neighbor matchings of query patches to dictionary patches. For all patches, the ground truth annotations (edge sets) of the central parts are shown alongside. The righthand panels show the results of the nearest neighbor searches for different combinations of the query encoding and the dictionary patch encoding. *"Neural"* corresponds to the encoding with top-layer activations $\mathrm{CNN}(\mathcal{P})$ of the CNN, while *"Target"* corresponds to the "ground truth"' encoding $\mathrm{PCA}(\mathbf{A}(\mathcal{P}))$ that the CNN is being trained to replicate. Matching neural codes to target codes (Neural/Target) works poorly thus highlighting the gap between the neural codes and the target PCA codes (which is the manifestation of the underfitting during the CNN training). By using neural codes for both the queries and the dictionary patches, our approach is able to overcome such underfitting and to match many patches to correct annotations (see Neural/Neural matching).

search in the training dataset. Since SIFT was designed mainly for natural RGB photographs, we evaluate this baseline for the BSDS500 edge detection only.

Alternative Encoding (AE). Given the impressive results of (Dollár and Zitnick, 2013) on edge detection, we experimented with a variation of our method inspired by their method. We annotate each patch with a long binary vector that looks at the pairs of pixels in the output $N \times N$ patch and assigns it 1 or 0 depending whether it belongs to the object segment. We then apply PCA dimensionality reduction to 16 components. More formally, we define the target annotation vector during the CNN training to be:

$$\mathbf{B}(P) = \mathrm{PCA}((v_1, v_2, \ldots, v_L))\,, \tag{3.4.1}$$

where $L = \binom{N^2}{2}$ and v_i is defined for i-th pair (p_l, p_m) of pixels in the ground truth segmentation $\mathbf{S}(\mathcal{P})$ and is equal to $\mathbb{1}\{\mathbf{S}(\mathcal{P})[p_l] = \mathbf{S}(\mathcal{P})[p_m]\}$. In the experiments, we observe a small improvement for such alternative encoding.



(A) BSDS500 (B) NYU RGBD

FIGURE 3.5. Performance scores for different tolerance thresholds (default value is $\mathbf{0.75 \cdot 10^{-2}}$) used in the BSDS500 benchmark (Arbeláez et al., 2011). Algorithms' performance (ODS and OIS measures plotted as *dashed* and *solid* lines respectively) is going down as the tolerance threshold is decreased. N^4-fields (*blue* lines) handles more stringent thresholds better, which suggests that cleaner edges are produced, as is also evidenced by the qualitative results. See Section 3.4 for details.

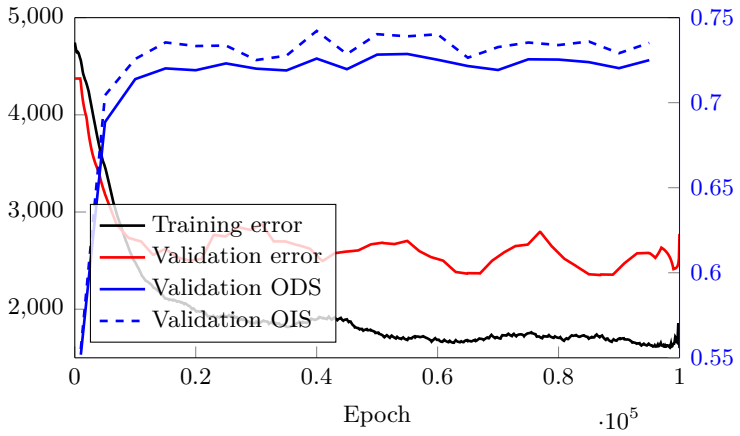


FIGURE 3.6. The validation score (average precision) of the full N^4-fields and error rates (loss) of the underlying CNN measured throughout the training process. The strong correlation between the values suggests the importance of large-scale learning for the good performance of N^4-fields. This experiment was performed for the BSDS500 edge detection (hold out validation set included 20 images).

BSDS500 Experiments. The first dataset is Berkley Segmentation Dataset and Benchmark (BSDS500) (Arbeláez et al., 2011). It contains 500 color images divided into three subsets: 200 for training, 100 for validation and 200 for testing. Edge detection accuracy

	ODS	OIS	AP
SIFT + NNB	.59	.60	.60
CNN-central	.72	.74	.75
CNN-patch	.73	.75	.74
gPb-owt-ucm [1]	.73	.76	.73
SCG [2]	.74	.76	.77
SE-MS, $T = 4$ [4]	.74	.76	**.78**
DeepNet [4]	.74	.76	.76
PMI + sPb, MS [5]	.74	**.77**	**.78**
N^4-fields	**.75**	.76	.77
N^4-fields, AE	**.75**	**.77**	**.78**

(A) BSDS500 (Any)

	ODS	OIS	AP
SE-MS, $T = 4$ [3]	.59	.62	.59
DeepNet [4]	.61	.64	.61
PMI + sPb, MS [5]	.61	**.68**	.56
N^4-fields, AE	**.64**	.67	**.64**

(B) BSDS500 (Consensus)

	ODS	OIS	AP
CNN, central	.60	.62	.55
CNN, patch	.58	.59	.49
gPb [1]	.53	.54	.40
SCG [2]	.62	.63	.54
SE-MS, $T = 4$ [3]	**.64**	**.65**	**.59**
N^4-fields	.61	.62	.56
N^4-fields, AE	.63	.64	.58

(C) NYU RGBD

TABLE 3. I. Edge detection results on BSDS500 (Arbeláez et al., 2011) (both for the original ground-truth annotation and "consensus" labels) and NYU RGBD (Silberman and Fergus, 2011). Our approach (N^4-fields) achieves performance which is better or comparable to the state-of-the-art. We also observe that the relative performance of the methods in terms of perceptual quality are not adequately reflected by the standard performance measures. The table list the quantitative results for the following methods: [1] Arbeláez et al. (2011), [2] Xiaofeng and Bo (2012), [3] Dollár and Zitnick (2013), [4] Kivinen et al. (2014), [5] Isola et al. (2014).

is measured using three scores: fixed contour threshold (ODS), per-image threshold (OIS), and average precision (AP) (Arbeláez et al., 2011; Dollár and Zitnick, 2013). In order to be evaluated properly, test edges must be thinned to one pixel width before running the benchmark code. We use the non-maximum suppression algorithm from (Dollár and Zitnick, 2013) for that.

In general, N^4-fields perform similarly to the best previously published methods (Dollár and Zitnick, 2013; Kivinen et al., 2014; Isola et al., 2014). In particular, the full version of the

FIGURE 3.7. Representative results on the BSDS500 dataset. For comparison, we give the results of the best previously published method (Dollár and Zitnick, 2013). The red numbers correspond to Recall/Precision/F-measure. We give two examples where N^4-fields perform better than (Dollár and Zitnick, 2013), and one example (bottom row) where (Dollár and Zitnick, 2013) performs markedly better according to the quantitative measure.

system (the committee of three N^4-fields applied at three scales) matches the performance of the mentioned algorithms , with the alternative encoding performing marginally better (Table 3. I-A). Following Hou et al. (2013) in order to account for the inherent problems of the dataset we also test our approach against the so-called "consensus" subset of the ground-truth labels. Within this setting our method significantly outperforms other algorithms in terms of ODS and AP (Table 3. I-B).

The benchmark evaluation procedure does not perform strict comparison of binary edge masks but rather tries to find the matching between pixels within certain tolerance level and then analyzes unmatched pixels (Arbeláez et al., 2011). We observed that the default distance matching tolerance threshold, while accounting for natural uncertainty in the exact position of the boundary, often ignores noticeable and unnatural segmentation mistakes such as spurious boundary pixels. Therefore, in addition to the accuracy evaluated for the default matching threshold, we report results for more stringent thresholds (Figure 3.5a-left).

It is also insightful to see whether the outputs of the CNN within the N^4-fields, i.e. CNN($\mathcal{P}$) are reasonably close to the codes PCA($\mathbf{A}(\mathcal{P})$) that were used as target during the learning. To show this, in Figure 3.4 we give several representative results of the nearest neighbor searches where different types of codes are used on the query and on the dictionary dataset sides (alongside the corresponding patches). It can be seen, that there are very accurate matches (in terms of similarity between true annotations) between PCA codes on both sides, and reasonably good matches between neural (CNN) codes on both sides. However, when matching the neural code of an input patch to PCA codes on the dataset side the results are poor. This is especially noticeable for patches without natural boundaries in them as we force our neural network to map all such patches into one point (empty annotation is always encoded with the same vector). This qualitative performance results in a notoriously bad quantitative performance of the system that uses such matching (from the neural codes in the test image to the PCA codes in the training dataset).

While CNN is clearly unable to learn to reproduce the target codes closely, there is still a strong correlation between the training error (the value of the loss function within the CNN) and the performance of the N^4-fields (Figure 3.6). The efficiency of the learned codes and its importance for the good performance of N^4-fields is also highlighted by the fact that the nearest neighbor baseline using SIFT codes performs very poorly. Thus, optimizing the loss functions introduced above really makes edge maps produced by our algorithm agree with ground truth annotations.

NYU RGBD Experiments. We also show results for the NYU Depth dataset (v2) (Silberman and Fergus, 2011). It contains 1,449 RGBD images with corresponding semantic segmentations. We use a script (Xiaofeng and Bo, 2012) to translate the data into BSDS500 format and use the same evaluation procedure, following training/testing split proposed by Xiaofeng and Bo (2012). The CNN architecture stays the same except for the number of input channels which is now equal to four (RGBD) instead of three (RGB).

The results are summarized in Table 3. I-C. Our approach almost ties the state-of-the-art method by Dollár and Zitnick (2013) for the default matching threshold. However, just like in the case of the BSDS500 dataset this difference in scores may be due to the peculiarity of the benchmark. Indeed, Figure 3.5b-right shows that for smaller values of matching thresholds, N^4-fields match or outperform the accuracy of Structured Edge detector (Dollár and Zitnick, 2013).

Note on the Quantitative Performance. During the experiments, we observed a clear disconnect between the relative performance of the methods according to the quantitative measures, and according to the actual perceptual quality. This was especially noticeable for

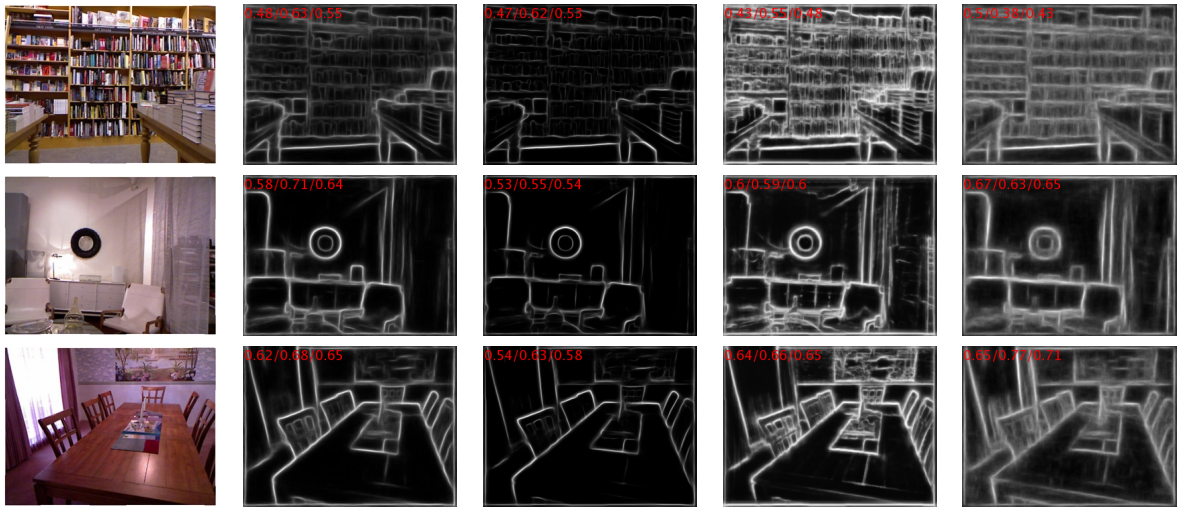


FIGURE 3.8. Results on the NYU RGBD dataset. For comparison, we give the results of the best previously published method (SE) (Dollár and Zitnick, 2013) and the CNN baseline. We show a representative result where the N^4-fields perform better (top), similarly (middle), or worse (bottom) than the baseline, according to the numberic measures shown in red (recall/precision/F-measure format). We argue that the numerical measures do not adequately reflect the relative perceptual performance of the methods.

the NYU RGBD dataset (Figure 3.8). We provide extended uniformly-sampled qualitative results at the project website[3].

DRIVE Dataset. In order to demonstrate wide applicability of our method, we evaluate it on the DRIVE dataset (Staal et al., 2004) containing micrographs obtained in a diabetic retinopathy screening program. It includes forty 768×584 images split evenly into a training and a test sets. Ground truth annotations include manually segmented vasculature as well as ROI masks.

We use exactly the same CNN architecture as in the BSDS500 experiment. Without any further tuning our system achieves state-of-the-art performance comparable to the algorithm proposed by Becker et al. (2013). Precision/recall curves for both approaches as well as for the baseline neural networks and (Dollár and Zitnick, 2013) (obtained using the authors' code) are shown in Figure 3.10. Notably, there is once again a clear advantage over the CNN baselines. Poor performance of (Dollár and Zitnick, 2013) is likely to be due to the use of default features that are not suitable for this particular imaging modality. This provides an extra evidence for the benefits of fully data-driven approach.

[3] `http://sites.skoltech.ru/compvision/projects/n4/` at the moment of publication.

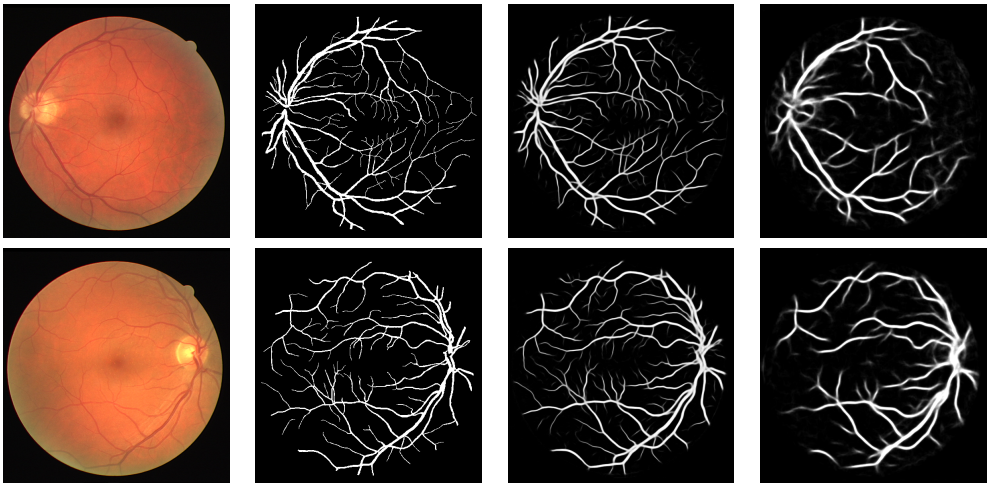


FIGURE 3.9. Representative results on the DRIVE dataset. A close match to the human expert annotation is observed. The last column corresponds to the algorithm proposed in (Dollár and Zitnick, 2013).

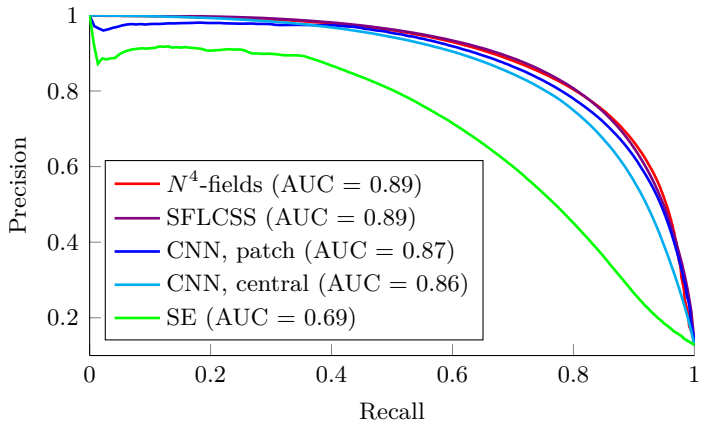


FIGURE 3.10. Results for the DRIVE dataset (Staal et al., 2004) in the form of the recall/precision curves. Our approach matches the performance of the current state-of-the-art method by SFLCSS (Becker et al., 2013) and performs much better than baselines and (Dollár and Zitnick, 2013).

3.5. CONCLUSION

We have presented a new approach to machine-learning based image processing. We have demonstrated how convolutional neural networks can be efficiently combined with the nearest neighbor search, and how such combination can improve the performance of standalone CNNs in the situation when CNN training underfits due to the complexity of a problem at hand.

State-of-the-art results[4] are demonstrated for natural edge detection in RGB and RGBD images, as well as for thin object (vessel) segmentation. Compared to the structured forests method (Dollár and Zitnick, 2013), the proposed approach is slower, but can be adapted to new domains (*e.g.*, micrographs) without manual retuning.

The future work may concern the fact that we employ PCA compression to define direct targets during the CNN training. It's easy to see, that for the the nearest neighbor search we don't really care about the actual values but rather about the topology (*i.e.*, distances between vectors). This suggests that we could replace naive regression with relative distance matching. It remains to be seen whether this approach will bring any improvements to the overall performance of the system.

[4]At the time of publication.

PROLOGUE TO THE SECOND ARTICLE

3.6. Article Details

DeepWarp: Photorealistic Image Resynthesis for Gaze Manipulation. Yaroslav Ganin, Daniil Kononenko, Diana Sungatullina, and Victor Lempitsky. *Proceedings of the 14th European Conference on Computer Vision (ECCV 2016).*

Personal Contributions. The idea of using differential bilinear warping of the input for obtaining redirected gaze came to me during one of the Computer Vision Group meetings (at Skoltech) in the fall of 2015. I quickly implemented an initial version of the model and verified that it was working well. Victor Lempitsky suggested that I should train a unified model for arbitrary redirection angles in both vertical and horizontal directions. The latter variation which was also designed and implemented by me became the core of our ECCV 2016 submission. Daniil Kononenko and Diana Sungatullina provided the data (initially collected for a different project). They were also responsible for the user study that can be found in the experimental section of the paper. I conducted the rest of the experiments and wrote the initial draft of the submission. Daniil Kononenko and Diana Sungatullina contributed to the related work and to the experimental sections. Victor Lempitsky focused on the introduction and the conclusion as well as general editing of the manuscript. I did the final editing for the camera-ready version of the paper after it was accepted to the conference.

3.7. Context

We first started discussing applying neural networks for gaze correction back in the summer of 2013. Victor Lempitsky deemed that task as very important since solving it could have a big impact on how people communicate on the internet. Video conferences lacked the feel

of real conversations partly because the participants would gaze at the screen instead of the camera and therefore appear not to be looking at the interlocutor.

Later in 2013, Daniil Kononenko took the lead of the project. The idea to use NNs was dropped in favor of random forests which were a better fit for real-time processing (Kononenko and Lempitsky, 2015). Having accomplished two successful deep learning efforts, I was naturally interested in seeing how a neural approach would compare to random forests. Following the philosophy of (Kononenko and Lempitsky, 2015), I developed a fully-convolutional architecture that predicted a per-pixel displacement field. Unlike (Kononenko and Lempitsky, 2015), my system represented each output pixel as a weighted sum of neighbouring pixels in the input image. The main problem with this approach was that the predicted weights were often distributed over several distinct relative locations. As a result, the output image looked blurry. We managed to partially remedy this by applying an additional sharpening step but that solution was not satisfactory.

After an extended break during which Jaderberg et al. (2015) released their seminal work on STNs, I made an attempt to improve my gaze correction system by adopting that new technique. The network was now constrained to assign each output pixel to a single location in the input. This seemingly minor difference drastically improved the quality making it nearly photorealistic. We felt this result was worth sharing with the research community not only because it made gaze correction better but also because dense prediction of the flow for an STN was a novel technique that could potentially be used in a wide range of computer vision tasks. We summarized our observations in a paper that was eventually accepted to ECCV 2016.

Since it was impossible to share the source code for the method (due to licensing restrictions), we decided to make our system available in the form of a web-service. I developed a site (both the back and the front-end) that allowed users to upload their photos and manipulate the gaze direction. Much to our surprise, the service got a good media coverage (Plaugic, 2017) and was used quite heavily both for fun and for interesting art/technology fusion projects (Ulicny, 2017).

3.8. Contributions

Although the idea of using differentiable warping modules in CNNs was first explored in (Jaderberg et al., 2015), that work was dealing with the classification task and transformations were carried out on the feature level. In contrast, our paper demonstrated how one

can use dense displacement field prediction for direct image-level resynthesis. One appealing property of this technique is that it makes it easy for the network to retain fine-grained details of the input, something conventional black-box image-to-image CNN-based models struggle with.

Two other main contributions are the multi-scale flow refinement procedure and the correction of the warped image. Both are implemented as modules of a bigger end-to-end system. The former significantly improved the spatial coherency of the predicted field, while the latter helped with difficult cases of occlusion and dis-occlusion which required synthesis of novel pixel that are not present in the input image.

3.9. Recent Developments

In the context of gaze correction, the ideas from our paper were further explored by Kononenko et al. (2017) and Wood et al. (2017). The former use a NN as a teacher for a random forest therefore improving the running time of the system. One caveat of their method is that it can only deal with a single redirection angle. Wood et al. (2017) take a different approach and use a non-neural 3D-model fitting. Transformation is no longer applied to the entire eye region but just to the eyelids.

We also made an attempt to adapt DeepWarp for more general facial expression manipulation (Ganin, 2017). Since we used CelebA as our dataset, we did not have an access to matching input-output pairs. The data was instead presented as a collection of images with corresponding expression labels. Our preliminary experiments show that DeepWarp is capable of learning transformations even in absence of strong supervision. A similar system (albeit trained on matching pairs from Multi-Pie Gross et al. (2010)) was later proposed by Yeh et al. (2016).

The differentiable image warping which is the core of our method was used in a range of CV tasks beyond gaze correction, *e.g.*, novel view synthesis (Zhou et al., 2016; Park et al., 2017), super-resolution (Caballero et al., 2017) and interpolation between video frames (van Amersfoort et al., 2017; Liu et al., 2017). Finally, concurrently with us, Yu et al. (2016) proposed to employ STNs for weakly supervised optical flow prediction.

Chapter 4

PHOTOREALISTIC IMAGE RESYNTHESIS FOR GAZE MANIPULATION

4.1. Introduction

In this work, we consider the task of learning deep neural networks for image editing and resynthesis. Generally, using deep architectures for image generation has become a very active topic of research. While a lot of very interesting results have been reported over recent years and even months, achieving photo-realism beyond the task of synthesizing small patches has proven to be hard.

Previously proposed methods for deep resynthesis usually tackle the resynthesis problem in a general form and strive for universality. Here, we take an opposite approach and focus on a very specific image resynthesis problem (gaze manipulation) that has a long history in the computer vision community (Okada et al., 1994; Yang and Zhang, 2002; Yip and Jin, 2003; Criminisi et al., 2003; Jones et al., 2009; Wolf et al., 2010; Kuster et al., 2012; Giger et al., 2014; Kononenko and Lempitsky, 2015) and some important real-life applications. We show that by restricting the scope of the method and exploiting the specifics of the task, we are indeed able to train deep architectures that handle gaze manipulation well and can synthesize output images of high realism (Figure 4.1).

Generally, few image parts can have such a dramatic effect on the perception of an image like regions depicting eyes of a person in this image. Humans (and even non-humans (Wallis et al., 2015)) can infer a lot of information about of the owner of the eyes, her intent, her mood, and the world around her, from the appearance of the eyes and, in particular, from the direction of the gaze. Generally, the role of gaze in human communication is long known to be very important (Kleinke, 1986).

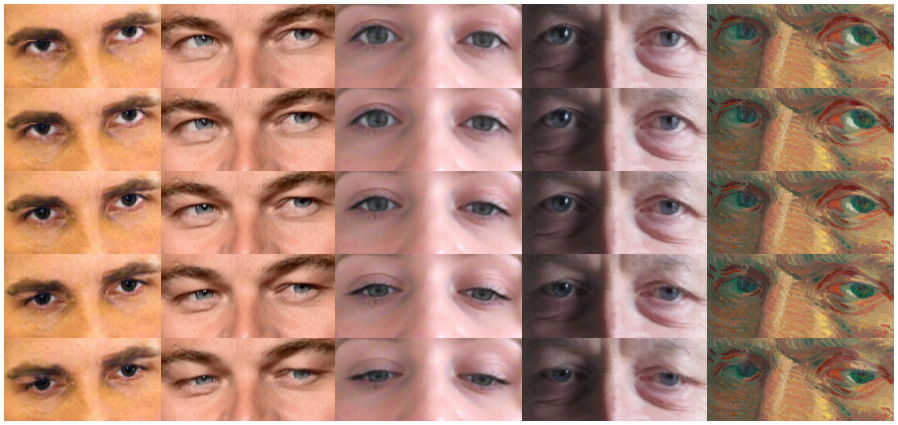


FIGURE 4.1. Gaze redirection with our model trained for vertical gaze redirection. The model takes an input image (middle row) and the desired redirection angle (here varying between -15 and +15 degrees) and re-synthesize the new image with the new gaze direction. Note the preservation of fine details including specular highlights in the resynthesized images.

In some important scenarios, there is a need to digitally alter the appearance of eyes in a way that changes the apparent direction of the gaze. These scenarios include gaze correction in video-conferencing, as the intent and the attitude of a person engaged in a videochat is distorted by the displacement between the face on her screen and the webcamera (e.g. while the intent might be to gaze into the eyes of the other person, the apparent gaze direction in a transmitted frame will be downwards). Another common scenario that needs gaze redirection is "talking head"-type videos, where a speaker reads the text appearing alongside the camera but it is desirable to redirect her gaze into the camera. One more example includes editing of photos (e.g. group photos) and movies (e.g. during postproduction) in order to make gaze direction consistent with the ideas of the photographer or the movie director.

All of these scenarios put very high demands on the realism of the result of the digital alteration, and some of them also require real-time or near real-time operation. To meet these challenges, we develop a new deep feed-forward architecture that combines several principles of operation (coarse-to-fine processing, image warping, intensity correction). The architecture is trained end-to-end in a supervised way using a specially collected dataset that depicts the change of the appearance under gaze redirection in real life.

Qualitative and quantitative evaluation demonstrate that our deep architecture can synthesize very high-quality eye images, as required by the nature of the applications, and does so at several frames per second. Compared to several recent methods for deep image synthesis, the output of our method contains larger amount of fine details (comparable to the

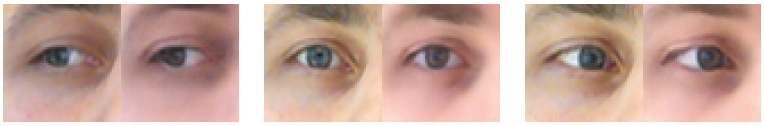


FIGURE 4.2. Examples of reconstructions produced by a modern **encoder-decoder** architecture (following the approach in (Kulkarni et al., 2015b; Reed et al., 2015; Ghodrati et al., 2015)) trained on our data. In each pair, the left image is the input and the right is the output. Despite our efforts, a noticeable loss of fine-scale details and "regression-to-mean" effect make the result not good enough for most applications of gaze manipulation. Similar problems can be observed in (Kulkarni et al., 2015b; Reed et al., 2015; Ghodrati et al., 2015).

amount in the input image). The quality of the results also compares favorably with the results of a random forest-based gaze redirection method (Kononenko and Lempitsky, 2015). Our approach has thus both practical importance in the application scenarios outlined above, and also contributes to an actively-developing field of image generation with deep models.

4.2. Related Work

4.2.1. Deep Learning and Image Synthesis.

Image synthesis using neural networks is receiving growing attention (Mahendran and Vedaldi, 2015; Dosovitskiy et al., 2015; Goodfellow et al., 2014; Denton et al., 2015; Gatys et al., 2015; Gregor et al., 2015). More related to our work are methods that learn to transform input images in certain ways (Kulkarni et al., 2015b; Ghodrati et al., 2015; Reed et al., 2015). These methods proceed by learning internal compact representations of images using encoder-decoder (autoencoder) architectures, and then transforming images by changing their internal representation in a certain way that can be trained from examples. We have conducted numerous experiments following this approach combining standard autoencoders with several ideas that have reported to improve the result (convolutional and up-convolutional layers (Zeiler and Fergus, 2014; Dosovitskiy et al., 2015), adversarial loss (Goodfellow et al., 2014), variational autoencoders (Kingma and Ba, 2014)). However, despite our efforts (see Figure 4.2), we have found that for large enough image resolution, the outputs of the network lacked high-frequency details and were biased towards typical mean of the training data ("regression-to-mean" effect). This is consistent with the results demonstrated in (Kulkarni et al., 2015b; Ghodrati et al., 2015; Reed et al., 2015) that also exhibit noticeable blurring.

Compared to Kulkarni et al. (2015b); Ghodrati et al. (2015); Reed et al. (2015), our approach can learn to perform a restricted set of image transformations. However, the perceptual quality and, in particular, the amount of high-frequency details is considerably better in the case of our method due to the fact that we deliberately avoid any input data compression within the processing pipeline. This is crucial for the class of applications that we consider.

Finally, the idea of spatial warping that lies in the core of the proposed system has been previously suggested in (Jaderberg et al., 2015). In relation to (Jaderberg et al., 2015), parts of our architecture can be seen as spatial transformers with the localization network directly predicting a sampling grid instead of low-dimensional transformation parameters.

4.2.2. Gaze Manipulation.

An early work on monocular gaze manipulation (Wolf et al., 2010) did not use machine learning, but relied on pre-recording a number of potential eye replacements to be copy-pasted at test time. The idea of gaze redirection using supervised learning was suggested in (Kononenko and Lempitsky, 2015), which also used warping fields that in their case were predicted by machine learning. Compared to their method, we use deep convolutional network as a predictor, which allows us to achieve better result quality. Furthermore, while random forests in (Kononenko and Lempitsky, 2015) are trained for a specific angle of gaze redirection, our architecture allows the redirection angle to be specified as an input, and to change continuously in a certain range. Most practical applications discussed above require such flexibility. Finally, the realism of our results is boosted by the lightness adjustment module, which has no counterpart in the approach of Kononenko and Lempitsky (2015).

Less related to our approach are methods that aim to solve the gaze problem in video-conferencing via synthesizing 3D rotated views of either the entire scene (Okada et al., 1994; Criminisi et al., 2003; Yang and Zhang, 2002) or of the face (that is subsequently blended into the unrotated head) (Kuster et al., 2012; Giger et al., 2014). Out of this works only (Giger et al., 2014) works in a monocular setting without relying on extra imaging hardware. The general problem with the novel view synthesis is how to fill disoccluded regions. In cases when the 3D rotated face is blended into the image of the unrotated head (Kuster et al., 2012; Giger et al., 2014), there is also a danger of distorting head proportions characteristic to a person.

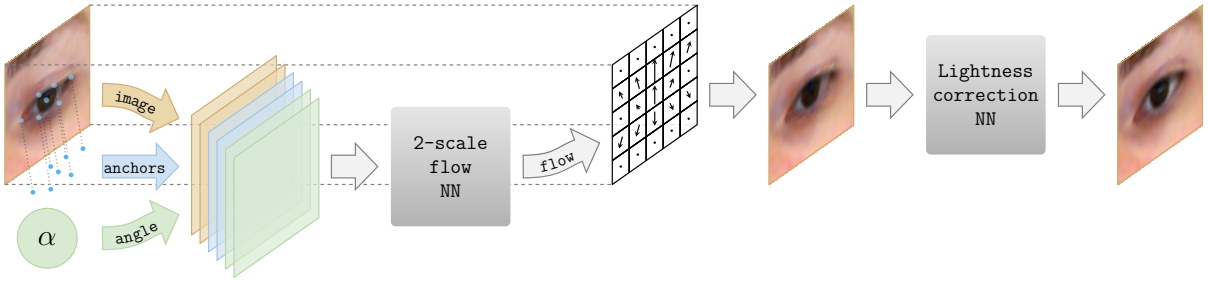


FIGURE 4.3. The **proposed system** takes an input eye region, feature points (`anchors`) as well as a correction `angle` α and sends them to the multi-scale neural network (see Section 4.3.2) predicting a `flow` field. The flow field is then applied to the input image to produce an image of a redirected eye. Finally, the output is enhanced by processing with the lightness correction neural network (see Section 4.3.4).

4.3. The Model

In this section, we discuss the architecture of our deep model for resynthesis. The model is trained on pairs of images containing rectangular eye regions before and after redirection. The redirection angle serves as an additional input parameter that is provided both during training and at test time.

As in (Kononenko and Lempitsky, 2015), the bulk of gaze redirection is accomplished via warping the input image (Figure 4.3). The task of the network is therefore the prediction of the warping field. This field is constructed in two stages in a coarse-to-fine manner, where the decisions at the fine scale are being informed by the result of the coarse stage. Beyond coarse-to-fine warping, the photorealism of the result is improved by performing pixel-wise brightness correction where the amount of correction is again predicted by the network. All operations outlined above are implemented in a single feed-forward architecture and are trained jointly end-to-end.

We now provide more details on each of the stages outlined above, starting with a description of the data used to train the architecture.

4.3.1. Data Preparation

At training time, our dataset allows us to mine pairs of images containing eyes of the same person looking in two different directions separated by a known angle α. The head pose, the lighting, and all other nuisance parameters are (approximately) the same between the two images in the pair. Following Kononenko and Lempitsky (2015) (with some modifications), we extract the image parts around each of the eye and resize them to a characteristic scale.

For simplicity of explanation, let us assume that we need to handle left eyes only (the right eyes can be handled at training and at test times via mirroring).

To perform the extraction, we employ an external face alignment library (Xiong and Torre, 2013) producing, among other things, $N = 7$ feature points $\{(x_i^{\text{anchor}}, y_i^{\text{anchor}}) \,|\, i = 1, \ldots, N\}$ for the eye (six points along the edge and also the pupil center). Next, we compute a tight axis-aligned bounding box $\mathcal{B}'$ of the points in the *input* image. We enlarge $\mathcal{B}'$ to the final bounding-box $\mathcal{B}$ using a characteristic radius R that equals the distance between the corners of an eye. The size of $\mathcal{B}$ is set to $0.8R \times 1.0R$. We then cut out the interior of the estimated box from the input image, and also from the output image of the pair (using exactly the same bounding box coordinates). Both images are then rescaled to a fixed size ($W \times H = 51 \times 41$ in our experiments). The resulting image pair serves as a training example for the learning procedure (Figure 4.5-Right).

4.3.2. Warping Modules

Each of the two warping modules receives the following inputs: the image, the position of the feature points, and the redirection angle. All inputs are expressed as maps as discussed below, and the architecture of the warping modules is thus "fully-convolutional", including several convolutional layers interleaved with Batch Normalization layers (Ioffe and Szegedy, 2015) and ReLU non-linearities (the actual configuration is shown in Figure 4.10). To preserve the resolution of the input image, we use '`SAME`'-mode convolutions (with zero padding), set all strides to one, and avoid using max-pooling.

Coarse warping. The last convolutional layer of the first (half-scale) warping module produces a pixel-flow field (a two-channel map), which is then upsampled (giving $\mathbf{D}_{\text{coarse}}(I, \alpha)$) and used to warp the input image by means of a bilinear sampler $\mathbf{S}$ (Jaderberg et al., 2015; Oquab, 2015) that finds the *coarse estimate*:

$$O_{\text{coarse}} = \mathbf{S}\left(I, \mathbf{D}_{\text{coarse}}(I, \alpha)\right) . \tag{4.3.1}$$

Here, the sampling procedure $\mathbf{S}$ is defined as:

$$O_{\text{coarse}}(x, y, c) = I\{x + \mathbf{D}_{\text{coarse}}(I, \alpha)(x, y, 1), y + \mathbf{D}_{\text{coarse}}(I, \alpha)(x, y, 2), c\} , \tag{4.3.2}$$

where c corresponds to a color channel (R,G, or B), and the curly brackets correspond to bilinear interpolation of $I(\cdot, \cdot, c)$ at a real-valued position. The operation (4.3.1) is piecewise differentiable (Jaderberg et al., 2015).

Fine warping. In the fine warping module, the rough image estimate O_{coarse} and the upsampled low-resolution flow $\mathbf{D}_{\text{coarse}}(I, \alpha)$ are concatenated with the input data (the image,

the angle encoding, and the feature point encoding) at the original scale and sent to the 1×-scale network which predicts another two-channel flow $\mathbf{D}_{\text{res}}$ that amends the half-scale pixel-flow (additively (He et al., 2015a)):

$$\mathbf{D}(I, \alpha) = \mathbf{D}_{\text{coarse}}(I, \alpha) + \mathbf{D}_{\text{res}}(I, \alpha, O_{\text{coarse}}, \mathbf{D}_{\text{coarse}}(I, \alpha)) \,, \tag{4.3.3}$$

The amended flow is then used to obtain the final output (again, via bilinear sampler):

$$O = \mathbf{S}\left(I, \mathbf{D}(I, \alpha)\right) \,. \tag{4.3.4}$$

The purpose of the coarse-to-fine processing is two-fold. The half-scale (coarse) module effectively increases the receptive field of the model resulting in a flow that moves larger structures in a more coherent way. Secondly, the coarse module gives a rough estimate of how a redirected eye would look like. This is useful for locating problematic regions which can only be fixed by a neural network operating at a finer scale.

4.3.3. Input Encoding

As discussed above, alongside the raw input image, the warping modules also receive the information about the desired redirection angle and feature points also encoded as image-sized feature maps.

Embedding the angle. Similarly to Ghodrati et al. (2015), we treat the correction angle as an attribute and embed it into a higher dimensional space using a multi-layer perceptron $\mathbf{F}_{\text{angle}}(\alpha)$ with ReLU non-linearities. The precise architecture is `FC(16)` → `ReLU` → `FC(16)` → `ReLU`. Unlike Ghodrati et al. (2015), we do not output separate features for each spatial location but rather opt for a single position-independent 16-dimensional vector. The vector is then expressed as 16 constant maps that are concatenated into the input map stack. During learning, the embedding of the angle parameter is also updated by backpropagation.

Embedding the feature points. Although in theory a convolutional neural network of an appropriate architecture should be able to extract necessary features from the raw input pixels, we found it beneficial to further augment 3 color channels with additional 14 feature maps containing information about the eye anchor points.

In order to get the anchor maps, for each previously obtained feature point located at $(x_i^{\text{anchor}}, y_i^{\text{anchor}})$, we compute a pair of maps:

$$\begin{aligned} \Delta_x^i[x, y] &= x - x_i^{\text{anchor}} \,, \\ \Delta_y^i[x, y] &= y - y_i^{\text{anchor}} \,, \end{aligned} \quad \forall (x, y) \in \{0, \ldots, W\} \times \{0, \ldots, H\} \,, \tag{4.3.5}$$

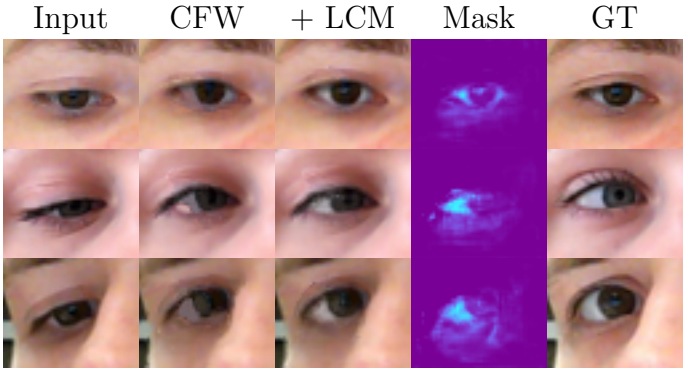


FIGURE 4.4. Visualization of three challenging redirection cases where **Lightness Correction Module** helps considerably compared to the system based solely on coarse-to-fine warping (CFW) which is having difficulties with expanding the area to the left of the iris. The 'Mask' column shows the soft mask corresponding to parts where lightness is increased. Lightness correction fixes problems with inpainting disoccluded eye-white, and what is more emphasizes the specular highlight increasing the perceived realism of the result.

where W, H are width and height of the input image respectively. The embedding give the network "local" access to similar features as used by decision trees in (Kononenko and Lempitsky, 2015).

Ultimately, the input map stack consists of 33 maps (RGB + 16 angle embedding maps + 14 feature point embedding maps).

4.3.4. Lightness Correction Module

While the bulk of appearance changes associated with gaze redirection can be modeled using warping, some subtle but important transformations are more photometric than geometric in nature and require a more general transformation. In addition, the warping approach can struggle to fill in disoccluded areas in some cases.

To increase the generality of the transformation that can be handled by our architecture, we add the final lightness adjustment module (see Figure 4.3). The module receives the features computed within the coarse warping and fine warping modules (specifically, the activations of the third convolutional layer), as well as the image produced by the fine warping module. The output of the module is a single map M of the same size as the output image that is used to modify the brightness of the output O using a simple element-wise transform:

$$O_{\text{final}}(x, y, c) = O(x, y, c) \cdot (1 - M(x, y)) + M(x, y)\,, \tag{4.3.6}$$

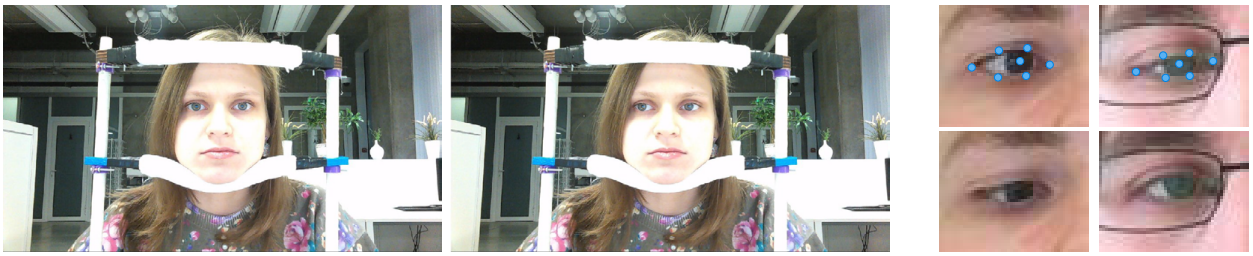


FIGURE 4.5. Left – dataset collection process. Right – examples of two training pairs (input image with superimposed feature points on top, output image in the bottom).

assuming that the brightness in each channel lies in the $[0, 1]$ range. The resulting pixel colors can thus be regarded as blends between the colors of the warped pixels and the white color. The actual architecture for the lightness correction module in our experiments is shown in Figure 4.11.

This idea can be, of course, generalized further to a larger number of colors for admixing, while these colors can be defined either manually or made dataset-dependent or even image-dependent. Our initial experiments along these directions, however, have not brought consistent improvement in photorealism in the case of the gaze redirection task.

4.4. EXPERIMENTS

4.4.1. Dataset

Unfortunately, we are not aware of any publicly available datasets suitable for the gaze correction task with continuously varying redirection angle. We therefore collect our own dataset (Figure 4.5). In order to minimize head movement, a person was asked to place her head in a special stand and track a moving point on the screen in front of the stand. While the point was moving, a special software was repeatedly taking a headshots with eyes gazing in different (but known) directions using a webcam mounted in the middle of the screen. For each person we collected 2-10 sequences each containing about 200 photos. Different sequences featured various head poses and light conditions. We manually excluded bad shots, where a person was blinking or where she was not changing gaze direction smoothly and monotonically. Most of the experiments were done on the dataset of 33 persons containg 98 sequences in total. Unless noted otherwise, we train the model for vertical gaze redirection in the range between -30 and 30.

4.4.2. Training Procedure

The model was trained end-to-end on 128-sized batches using Adam optimizer (Kingma and Ba, 2014). We used regular ℓ^2-distance between the synthesized output O_{output} and the ground-truth O_{gt} as the objective function. We tried to improve over this simple baseline in several ways. First, we tried to put emphasis on the actual eye region (not the rectangular bounding-box) by adding more weight to the corresponding pixels but were not able to get any significant improvements. Our earlier experiments with adversarial loss (Goodfellow et al., 2014) were also inconclusive. As the residual flow predicted by the $1\times$-scale module tends to be quite noisy, we attempted to smoothen the flow-field by imposing a total variation penalty. Unfortunately, this resulted in a slightly worse ℓ^2-loss on the test set.

Sampling training pairs. We found that biasing the selection process for more difficult and unusual head poses and bigger redirection angles improved the results. For this reason, we used the following sampling scheme aimed at reducing the dataset imbalance. We split all possible correction angles (that is, the range between -30 and 30) into 15 bins. A set of samples falling into a bin is further divided into "easy" and "hard" subsets depending on the input's *tilt* angle (an angle between the segment connecting two most distant eye feature points and the horizontal baseline). A sample is considered to be "hard" if its tilt is $\geqslant 8$. This subdivision helps to identify training pairs corresponding to the rare head poses. We form a training batch by picking 4 correction angle bins uniformly at random and sampling 24 "easy" and 8 "hard" examples for each of the chosen bins.

4.4.3. Quantitative Evaluation

We evaluate the proposed approach on our in-house dataset. We randomly split the initial set of subjects into a development (26 persons) and a test (7 persons) sets. Several methods were compared using the mean square error (MSE) between the synthesized and the ground-truth images extracted using the procedure described in Section 4.3.1.

Models. We consider 6 different models:

1. A system based on Structured Random Forests (*RF*) proposed in (Kononenko and Lempitsky, 2015). We train it for 15 redirection only using the reference implementation.
2. A single-scale (*SS* (15 only)) version of our method with a single warping module operating on the original image scale that is trained for 15 redirection only.
3. A single-scale (*SS*) version of our method with a single warping module operating at the original image scale.

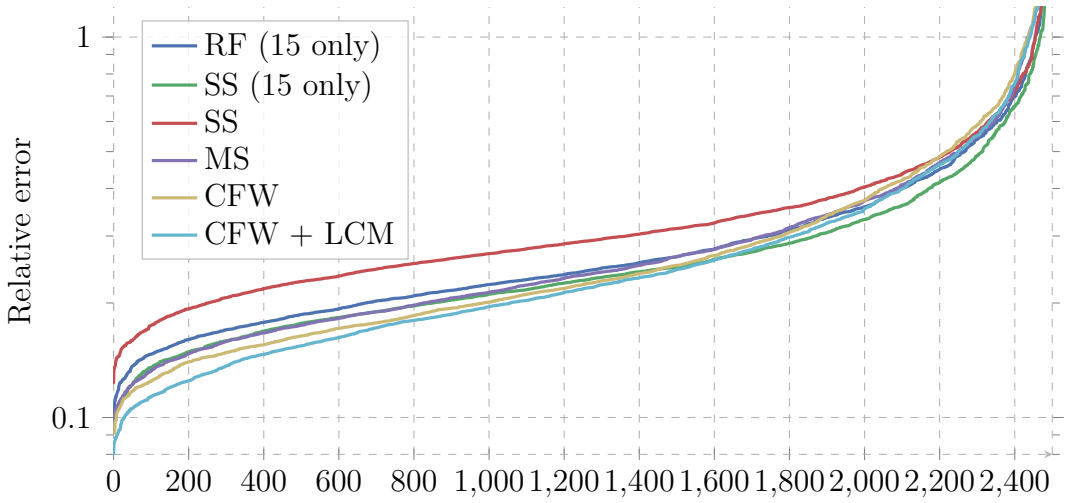


FIGURE 4.6. Ordered errors for 15 redirection. Our multi-scale models (MS, CFW, CFW + LCM) show results that are comparable or superior the Random Forests (RF) (Kononenko and Lempitsky, 2015).

4. A multi-scale (*MS*) network without coarse warping. It processes inputs at two scales and uses features from both scales to predict the final warping transformation.
5. A coarse-to-fine warping-based system described in Section 4.3 (*CFW*).
6. A coarse-to-fine warping-based system with a lightness correction module (*CFW + LCM*).

The latter four models are trained for the task of vertical gaze redirection in the range. We call such models *unified* (as opposed to single angle correction systems).

15 **correction.** In order to have the common ground with the existing systems, we first restrict ourselves to the case of 15 gaze correction. Following Kononenko and Lempitsky (2015), we present a graph of sorted normalized errors (Figure 4.6), where all errors are divided by the MSE obtained by an input image and sorted in the increasing order.

It can be seen that the unified multi-scale models are, in general, comparable or superior to the RF-based approach in (Kononenko and Lempitsky, 2015). Interestingly, the lightness adjustment extension (Section 4.3.4) is able to show quite significant improvements for the samples with low MSE. Those are are mostly cases similar to shown in Figure 4.4. It is also worth noting that the single-scale model trained for this specific correction angle consistently outperforms (Kononenko and Lempitsky, 2015), demonstrating the power of the proposed architecture. However, we note that results of the methods can be improved using additional registration procedure, one example of which is described in Section 4.4.6.

Arbitrary vertical redirection. We also compare different variants of the unified networks

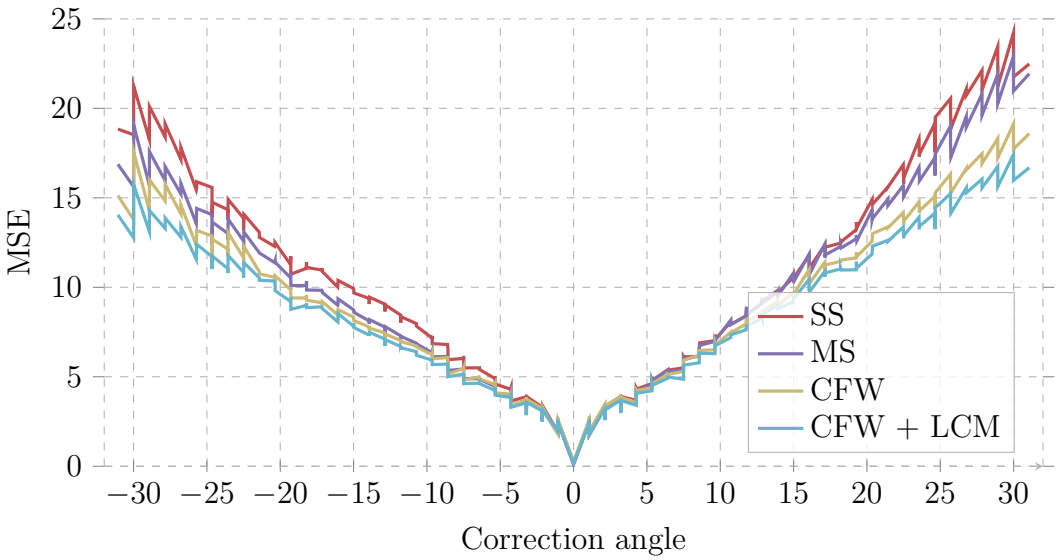


FIGURE 4.7. Distribution of errors over different correction angles.

and plot the error distribution over different redirection angles (Figure 4.7). For small angles, all the methods demonstrate roughly the same performance, but as we increase the amount of correction, the task becomes much harder (which is reflected by the growing error) revealing the difference between the models. Again, the best results are achieved by the LCM model, which is followed by the multi-scale networks making use of the coarse warping.

4.4.4. Perceptual Quality

We demonstrate the results of redirection on 15 degrees upwards in Figure 4.8. CFW-based systems produce the results visually closer to the ground truth than RF. The effect of the lightness correction is pronounced: on the input image with the lack of white Random Forest and CFW fail to get output with sufficient eye-white and copy-paste red pixels instead, whereas CFW+LCM achieve good correspondence with the ground-truth. However, the downside effect of the LCM could be blurring/lower contrast because of the multiplication procedure (4.3.6).

User study. To confirm the improvement corresponding to different aspects of the proposed models, which may not be adequately reflected by the ℓ^2-measure, we performed an informal user study enrolling 16 subjects unrelated to computer vision and comparing four methods (RF, SS, CFW, CFW+LCM).

A user was shown 160 quadruplets of images, each containing one image obtained with one of the methods, and three unprocessed real images. Each of the methods in the comparison

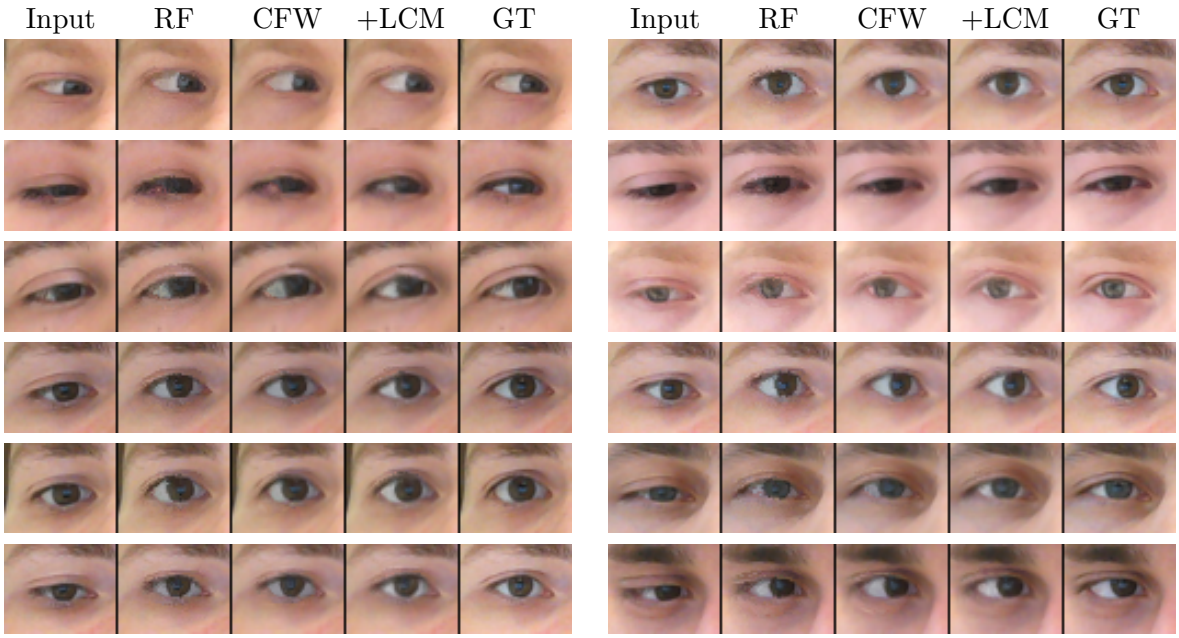


FIGURE 4.8. Sample results on a hold-out. The full version of our model (CFW+LCM) outperforms other methods.

was represented by 40 randomly sampled outputs. The task of the subject was to click on the artificial (resynthesized) image as quickly as possible.

We present various collected statistics in the Table 4. I. Poor algorithms are quite easy to detect and therefore the number of correct answers will be close to 40. Better systems force the user to resort to random guessing resulting in about 10 correct clicks on average. We also recorded the time that the subject spent deciding on the answer — one would expect higher quality outputs to decrease the click rate.

Notably, in the conducted user study, the gap between methods is much wider then it might seem from the MSE-based comparisons, with CFW+LCM method outperforming others very considerably, especially when taking into account the timings.

4.4.5. Horizontal Redirection

While most of our experiments were about vertical gaze redirection, the same models can be trained to redirect the gaze horizontally (and, with trivial generalization, by a 2D family of angles). In Figure 4.9, we provide qualitative results of CFW+LCM for horizontal redirection. Some examples showing the limitations of our method are given. The limitations are concerned with cases with severe disocclusions, where large areas have to be filled by the network.

TABLE 4. I. Results of the **user study**. Each of the 16 test subjects was presented with 160 quadruplets containing 3 real images and one synthesized image. The task was to click on the latter as quickly as possible. The first three sections of the table contain the numbers of correct guesses (the smaller the better). The last row shows the mean time the participants spent to make a guess (the larger the better). Our full system (coarse-to-fine warping with lightness correction) outperforms the baselines. See Section 4.4.4 and (Ganin et al., 2016b) for details.

	RF	SS	CFW	+LCM
CORRECTLY GUESSED (OUT OF 40)				
MEAN	36.1	33.8	28.8	**25.3**
MEDIAN	37	35	29	**25**
MAX	40	39	38	**34**
MIN	26	22	20	**16**
CORRECTLY GUESSED WITHIN 2 SECONDS (OUT OF 40)				
MEAN	26.4	21.1	11.7	**8.0**
MEDIAN	28.5	20.5	10	**8**
MAX	35	33	23	**17**
MIN	13	11	3	**0**
CORRECTLY GUESSED WITHIN 1 SECOND (OUT OF 40)				
MEAN	8.1	4.4	1.6	**1.1**
MEDIAN	6	3	**1**	**1**
MAX	20	15	7	**5**
MIN	**0**	**0**	**0**	**0**
MEAN TIME TO MAKE A GUESS				
MEAN TIME, SEC	1.89	2.30	3.60	**3.96**

We provide more qualitative results on the project webpage (Ganin et al., 2016b).

4.4.6. Incorporating registration.

We found that results can be further perceptually improved (see (Ganin et al., 2016b)) if the objective is slightly modified to take into account misalignment between inputs and ground-truth images. To that end, we enlarge the bounding-box $\mathcal{B}$ that we use to extract the output image of a training pair by $k = 3$ pixels in all the directions. With this modification, O_{gt} now has the size of $(H + 2k) \times (W + 2k)$, the new objective is defined as:

$$\mathcal{L}(O_{\text{output}}, O_{\text{gt}}) = \min_{i,j} \text{dist}\left(O_{\text{output}}, O_{\text{gt}}[i : i + H, j : j + W]\right) , \tag{4.4.1}$$

where $\text{dist}(\cdot)$ can be either ℓ^2 or ℓ^1-distance (the latter giving slightly sharper results), and $O_{\text{gt}}[i : i+H, j : j+W]$ corresponds to a $H \times W$ crop of O_{gt} with top left corner at the position

FIGURE 4.9. **Horizontal redirection** with a model trained for both vertical and horizontal gaze redirection. For the first six rows the angle varies from -15 to 15 relative to the central (input) image. The last two rows push the redirection to extreme angles (up to 45) breaking our model down.

(i, j). Being an alternative to the offline registration of input/ground-truth pairs (Kononenko and Lempitsky, 2015) which is computationally prohibitive in large-scale scenarios, this small trick greatly increases robustness of the training procedure against small misalignments in a training set.

4.5. DISCUSSION

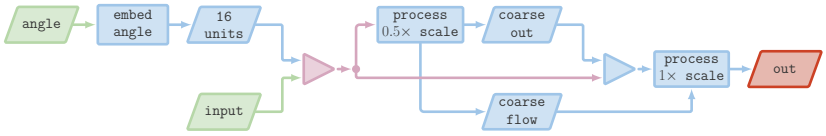
We have suggested a method for realistic gaze redirection, allowing to change the gaze continuously in a certain range. At the core of our approach is the prediction of the warping field using a deep convolutional network. We embed redirection angle and feature points as image-sized maps and suggest "fully-convolutional" coarse-to-fine architecture of warping modules. In addition to warping, photorealism is increased using lightness correction module. Quantitative comparison of MSE-error, qualitative examples and a user study show the advantage of suggested techniques and the benefit of their combination within an end-to-end learnable framework.

Our system is reasonably robust against different head poses (e.g., see Figure 4.4) and deals correctly with the situations where a person wears glasses (see (Ganin et al., 2016b)). Most of the failure modes (*e.g.*, corresponding to extremely tilted head poses or large redirection angles involving disocclusion of the different parts of an eye) are not inherent to the model design and can be addressed by augmenting the training data with appropriate examples.

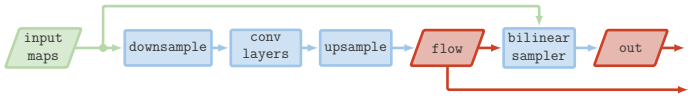
We concentrated on gaze redirection, although our approach might be extended for other similar tasks, *e.g.*, resynthesis of faces. In contrast with the autoencoder-based approach, our architecture does not compress data to a representation with lower explicit or implicit dimension, but directly transforms the input image. Our method thus might be better suited for fine detail preservation, and less prone to the "regression-to-mean" effect.

The computational performance of our method is up to 20 fps on a mid-range consumer GPU (NVIDIA GeForce-750M), which is however slower than the competing method of (Kononenko and Lempitsky, 2015) capable of achieving a similar speed on CPU. Our models are however much more compact than forests from (Kononenko and Lempitsky, 2015) (250 Kb vs 30-60 Mb in our comparisons), while also being universal. We are currently working on the unification of the two approaches.

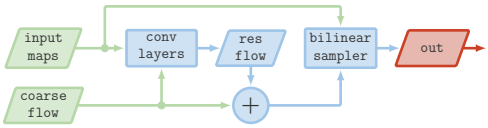
Speed optimization of the proposed system is another topic for future work. Finally, we plan to further investigate non-standard loss functions for our architectures (e.g. the one proposed in Section 4.4.6), as the ℓ^2-loss is not closely enough related to perceptual quality of results (as highlighted by our user study).



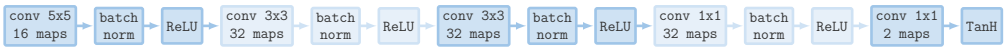
(A) High-level pipeline



(B) 0.5×-scale processing module



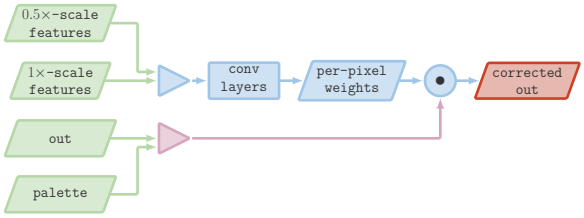
(C) 1×-scale processing module



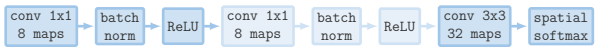
(D) Convolutional layers

(E) Legend

FIGURE 4.10. The **basic warping architecture** Figure 4.10a takes an input eye region augmented with eye feature points information (`input`) as well as a correction `angle` and produces an image of the redirected eye. The model contains three main blocks: angle embedding module (`embed angle`) calculating a vector representation of the correction `angle` and two warping modules (`process 0.5×-scale` Figure 4.10b and `process 1×-scale` Figure 4.10c) predicting and applying pixel-flow to the input image.



(A) Architecture.



(B) Convolutional layers.

FIGURE 4.11. **Lightness Correction Module** increases lightness of selected regions. Figure 4.11a shows the actual architecture of the module. Multi-scale features are processed by the convolutional neural network presented in Figure 4.11b.

PROLOGUE TO THE THIRD ARTICLE

Article Details

Unsupervised Domain Adaptation by Backpropagation. Yaroslav Ganin and Victor Lempitsky. *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015).*

Domain-adversarial Training of Neural Networks. Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. *The Journal of Machine Learning Research (JMLR 2016).*

Personal Contributions. The idea to use two subnetworks following opposite gradient updates in order to obtain domain-invariant features was first brought up by Victor Lempitsky in the fall of 2013. I joined the project in the spring of 2014 after submitting the edge detection paper. At that point, Victor Lempitsky had weak evidence that the method worked (a marginal improvement over a non-adapted baseline on a toy dataset). I became the lead of the project and reimplemented the algorithm for modern deep learning frameworks (`cuda-convnet` and `Caffe`). I was responsible for the choice of network architectures and for running all the experiments that ended up in our CVPR 2015 submission. I wrote the initial draft and Victor Lempitsky helped with the introduction, related work and the conclusion. He also did general editing of the text.

The paper was rejected mainly because the reviewers were not convinced with the experimental evaluation of the method. In order to address the concerns, I ran an additional experiment on the Office dataset (Saenko et al., 2010). It turned out that the way I was approaching other datasets did not work in this case. At that point, I came across a paper by Tzeng et al. (2014) and adopted some of the tricks presented there. That allowed me to obtain the state-of-the-art results beating the competitors by a large margin. An updated version of the manuscript with new experiments and a brief theoretical justification of the

method (which I came up with after reading (Ben-David et al., 2010)) was accepted to ICML 2015.

Later, we also submitted an extended journal article in collaboration with the researchers from Université Laval. Pascal Germain was responsible for merging the source papers, and other co-authors (including me) did general editing. The version of the article presented in this thesis is modified to only include the results that were obtained by me. I also addressed the issue with the theory that was accidentally introduced during the merge process.

Context

We started this project at the time when the subfield of domain adaptation was dominated by non-deep approaches. The only competitive neural network based system that we could identify was DLID (Chopra et al., 2013). The idea we were working on was not based on any existing DA algorithms and seemed conceptually simpler. Incidentally, the core of the proposed method was later rediscovered by Goodfellow et al. in the context of generative models. It is now commonly referred to as *adversarial training*.

As in the case of generative adversarial networks, it took some time for our work to be recognized by the community. For us, the main limiting factor was the quality of the empirical evaluation in the initial version of the paper. That was the reason for rejection from CVPR 2015. Interestingly, when we finally managed to demonstrate major benefits of our approach and had a chance to present the work at ICML 2015, the general audience seemed not to notice the connection between our method and GANs. The latter was not at all wide-spread until LapGAN (Denton et al., 2015) and DCGAN (Radford et al., 2015) models were released. This has changed significantly over time and nowadays the DA literature is gradually moving away from the terminology we proposed in our paper (*e.g.*, recent works prefer to use "discriminator" instead of "domain classifier").

After the paper was accepted to ICML 2015, we were contacted by Hugo Larochelle who pointed at a similar work that was done by his colleagues at Université Laval. Although that work was published later than our project and did not have strong empirical results, we still felt that we needed to share the credit and decided to submit a joint journal article to JMLR.

Contributions

To our knowledge, this work was one of the first (unfortunately, not the first published) to successfully apply adversarial training of neural networks. We devised a simple and effective domain adaptation method that demonstrated significant improvements over the state-of-the-art on standard DA datasets. Although in our experiments we primarily consider visual data, the approach is general and can be extended to a wide range of modalities.

Recent Developments

Our paper paved a way for many recent advances in domain adaptation and beyond. In this section, we are going to mention several interesting variations of the original domain-adversarial approach.

Inspired by the work on distilling deep models (Hinton et al., 2015), Tzeng et al. (2015) tackle the problem of class misalignment by using semi-supervised training with "soft-labels". They also introduce the notion of *domain confusion* thus expanding on the list of adversarial objectives considered in (Ganin and Lempitsky, 2015). Bousmalis et al. (2016) hypothesize that explicitly modeling what is unique to each domain can improve a network's ability to extract domain-invariant representations. Their approach implements this idea by employing a domain-specific residual orthogonal to the features of interest. The subsequent work from the same group of researchers (Bousmalis et al., 2017) explores DATNN in the pixel-space. Tzeng et al. (2017) unify the existing domain-adversarial approaches in a single flexible framework. They confirm the observation of Rozantsev et al. (2016) that using separate feature extractors for each of the domains may significantly improve the performance.

Most recently, Shu et al. (2018) propose a two-stage procedure that exploits the *cluster assumption* (Chapelle and Zien, 2005). The idea here is to push the decision boundary from data-dense regions in the target domain. Despite the simplicity of the approach, the method manages to achieve state-of-the-art results on all the standard datasets.

DATNN has also been successfully applied for tasks other than image classification. In the full journal article (Ganin et al., 2016c), we present additional results on sentiment analysis and person re-identification. We are also aware that since publication, there have been several papers using our idea for medical data processing (Purushotham et al., 2017), speech recognition (Serdyuk et al., 2016), training of deep recurrent networks (Lamb et al., 2016), and sensitive information removal (Beutel et al., 2017; Feutry et al., 2018), to name a few.

Chapter 5

DOMAIN-ADVERSARIAL TRAINING OF NEURAL NETWORKS

Introduction

The cost of generating labeled data for a new prediction task is often an obstacle for applying machine learning methods. In particular, this is a limiting factor for the further progress of deep neural network architectures. Another common problem is that, even when the training sets are big enough for training large-scale deep models, they may that suffer from the *shift* in data distribution from the actual data encountered at "test time". One important example is training an image classifier on synthetic or semi-synthetic images, which may come in abundance and be fully labeled, but which inevitably have a distribution that is different from real images.

Learning a predictor in the presence of a *shift* between training and test distributions is known as *domain adaptation* (DA). The appeal of the domain adaptation approaches is the ability to learn a mapping between domains in the situation when the target domain data are either fully unlabeled (*unsupervised domain annotation*) or have few labeled samples (*semi-supervised domain adaptation*). Below, we focus on the harder unsupervised case, although the proposed approach (*domain-adversarial learning*) can be generalized to the semi-supervised case rather straightforwardly.

Unlike many domain adaptation methods that work with fixed feature representations, we focus on combining domain adaptation and deep feature learning within one training process. Our goal is to embed domain adaptation into the process of learning a representation, so that the final classification decisions are made based on features that are both discriminative and invariant to the change of domains, *i.e.*, have the same or very similar distributions in the source and the target domains. In this way, the obtained feed-forward network can

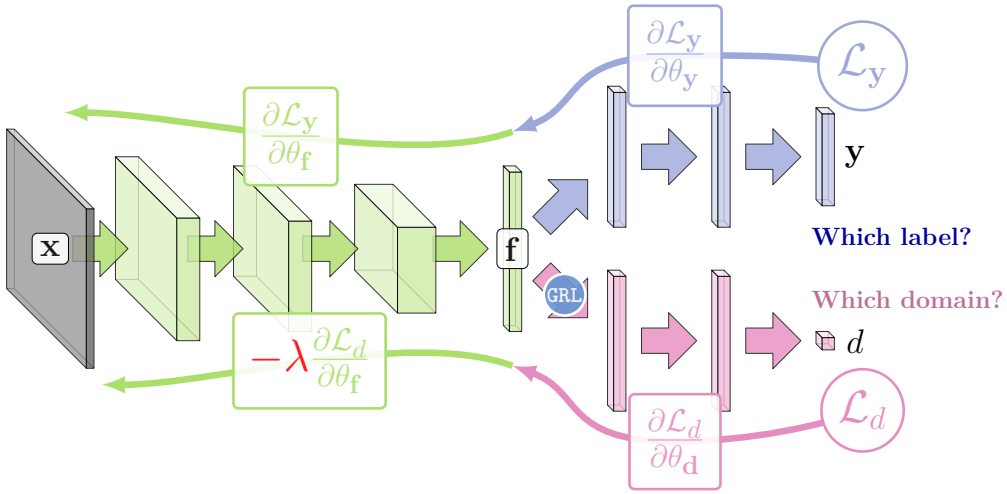


FIGURE 5.1. The **proposed architecture** includes a *feature extractor* (green) and a *label predictor* (blue), which together form a standard feed-forward architecture. Unsupervised domain adaptation is achieved by adding a *domain classifier* (red) connected to the feature extractor via a *gradient reversal layer* (GRL) that multiplies the gradient by a certain negative constant during the backpropagation-based training. The GRL ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier).

be applicable to the target domain without being hindered by the shift between the two domains. Our approach is motivated by the theory on domain adaptation (Ben-David et al., 2006; Blitzer et al., 2008; Ben-David et al., 2010), that suggests that a good representation for cross-domain transfer is one for which an algorithm cannot learn to identify the domain of origin of the input observation.

We thus focus on learning features that combine (i) discriminativeness and (ii) domain-invariance. This is achieved by jointly optimizing the underlying features as well as two discriminative classifiers operating on these features: (i) the *label predictor* that predicts class labels and is used both during training and at test time and (ii) the *domain classifier* that discriminates between the source and the target domains during training. While the parameters of the classifiers are optimized in order to minimize their error on the training set, the parameters of the underlying deep feature mapping are optimized in order to *minimize* the loss of the label classifier and to *maximize* the loss of the domain classifier. The latter update thus works *adversarially* to the domain classifier, and it encourages domain-invariant features to emerge in the course of the optimization.

Crucially, we show that all three training processes can be embedded into an appropriately composed deep feed-forward network, called *domain-adversarial neural network* (DANN) (illustrated by Figure 5.1) that uses standard layers and loss functions, and can be trained using standard backpropagation algorithms based on stochastic gradient descent or its modifications (*e.g.*, SGD with momentum). The approach is generic as a DANN version can be created for almost any existing feed-forward architecture that is trainable by backpropagation. In practice, the only non-standard component of the proposed architecture is a rather trivial *gradient reversal* layer that leaves the input unchanged during forward propagation and reverses the gradient by multiplying it by a negative scalar during the backpropagation.

We provide an experimental evaluation of the proposed idea over a range of image classification tasks, starting with results on traditional deep learning image datasets — such as MNIST (LeCun et al., 1998) and SVHN (Netzer et al., 2011) — and further testing the approach against the OFFICE benchmark (Saenko et al., 2010), where domain-adversarial learning allows obtaining a deep architecture that considerably improves over previous state-of-the-art accuracy.

5.1. Related work

The general approach of achieving domain adaptation has been explored under many facets. Over the years, a large part of the literature has focused mainly on the linear hypothesis (Blitzer et al., 2006; Bruzzone and Marconcini, 2010; Germain et al., 2013; Baktashmotlagh et al., 2013; Cortes and Mohri, 2014). More recently, non-linear representations have become increasingly studied, including neural network representations (Glorot et al., 2011a; Li et al., 2014) and most notably the state-of-the-art mSDA (Chen et al., 2012). That literature has mostly focused on exploiting the principle of robust representations, based on the denoising autoencoder paradigm (Vincent et al., 2008).

Concurrently, multiple methods of matching the feature distributions in the source and the target domains have been proposed for unsupervised domain adaptation. Some approaches perform this by reweighing or selecting samples from the source domain (Borgwardt et al., 2006; Huang et al., 2006; Gong et al., 2013), while others seek an explicit feature space transformation that would map source distribution into the target one (Pan et al., 2011; Gopalan et al., 2011; Baktashmotlagh et al., 2013). An important aspect of the distribution matching approach is the way the (dis)similarity between distributions is measured. Here, one popular choice is matching the distribution means in the kernel-reproducing Hilbert space

(Borgwardt et al., 2006; Huang et al., 2006), whereas Gong et al. (2012) and Fernando et al. (2013) map the principal axes associated with each of the distributions.

Our approach also attempts to match feature space distributions, however this is accomplished by modifying the feature representation itself rather than by reweighing or geometric transformation. Also, our method uses a rather different way to measure the disparity between distributions based on their separability by a deep discriminatively-trained classifier. Note also that several approaches perform transition from the source to the target domain (Gopalan et al., 2011; Gong et al., 2012) by changing gradually the training distribution. Among these methods, (Chopra et al., 2013) does this in a "deep" way by the layerwise training of a sequence of deep autoencoders, while gradually replacing source-domain samples with target-domain samples. This improves over a similar approach of Glorot et al. (2011a) that simply trains a single deep autoencoder for both domains. In both approaches, the actual classifier/predictor is learned in a separate step using the feature representation learned by autoencoder(s). In contrast to Glorot et al. (2011a); Chopra et al. (2013), our approach performs feature learning, domain adaptation and classifier learning jointly, in a unified architecture, and using a single learning algorithm (backpropagation). We therefore argue that our approach is simpler (both conceptually and in terms of implementation). Our method also achieves considerably better results on the popular Office benchmark.

While the above approaches perform unsupervised domain adaptation, there are approaches that perform *supervised* domain adaptation by exploiting labeled data from the target domain. In the context of deep feed-forward architectures, such data can be used to "fine-tune" the network trained on the source domain (Zeiler and Fergus, 2014; Oquab et al., 2014; Babenko et al., 2014). Our approach does not require labeled target-domain data. At the same time, it can easily employ such data when it is available.

An idea related to ours is described in a concurrent work by Goodfellow et al. (2014). While their goal is quite different (building deep generative networks that can synthesize samples), the way they measure and minimize the discrepancy between the distribution of the training data and the distribution of the synthesized data is very similar to the way our architecture measures and minimizes the discrepancy between feature distributions of the two domains. Moreover, Goodfellow et al. (2014) mention the problem of saturating sigmoids which may arise at the early stages of training due to the significant dissimilarity of the domains. The technique they use to circumvent this issue (*i.e.*, the "non-saturating" objective) is directly applicable to our method.

Also, recent and concurrent reports by Tzeng et al. (2014); Long and Wang (2015) focus on domain adaptation in feed-forward networks. Their set of techniques measures and

minimizes the distance between the data distribution means across domains (potentially, after embedding distributions into RKHS). Their approach is thus different from our idea of matching distributions by making them indistinguishable for a discriminative classifier. Below, we compare our approach to Tzeng et al. (2014); Long and Wang (2015) on the OFFICE benchmark. Another approach to deep domain adaptation, which is arguably more different from ours, has been developed in parallel by Chen et al. (2015).

From a theoretical standpoint, our approach is directly derived from the seminal theoretical works by Ben-David et al. (Ben-David et al., 2006, 2010). Indeed, DANN directly optimizes the notion of $\mathcal{H}\Delta\mathcal{H}$-divergence. We do note the work of Huang and Yates (2012), in which HMM representations are learned for word tagging using a posterior regularizer that is also inspired by Ben-David et al. (2010). In addition to the tasks being different — Huang and Yates (2012) focus on word tagging problems — we argue that DANN learning objective more closely optimizes the $\mathcal{H}\Delta\mathcal{H}$-divergence, with Huang and Yates (2012) relying on cruder approximations for efficiency reasons.

5.2. DOMAIN ADAPTATION

We consider classification tasks where $\mathcal{X}$ is the input space and $\mathcal{Y} = \{0, 1, \ldots, L-1\}$ is the set of L possible labels. Moreover, we have two different distributions over $\mathcal{X} \times \mathcal{Y}$, called the *source domain* $\mathcal{D}_{\mathrm{S}}$ and the *target domain* $\mathcal{D}_{\mathrm{T}}$. An unsupervised domain adaptation learning algorithm is then provided with a *labeled source sample* S drawn *i.i.d* from $\mathcal{D}_{\mathrm{S}}$, and an *unlabeled target sample* T drawn *i.i.d* from $\mathcal{D}_{\mathrm{T}}^{\mathcal{X}}$, where $\mathcal{D}_{\mathrm{T}}^{\mathcal{X}}$ is the marginal distribution of $\mathcal{D}_{\mathrm{T}}$ over $\mathcal{X}$:

$$S = \{(\mathbf{x}^s{}_i, y^s{}_i)\}_{i=1}^{n} \sim (\mathcal{D}_{\mathrm{S}})^n\,; \quad T = \{\mathbf{x}^t{}_i\}_{i=n+1}^{N} \sim (\mathcal{D}_{\mathrm{T}}^{\mathcal{X}})^{n'}\,,$$

with $N = n + n'$ being the total number of samples. The goal of the learning algorithm is to build a classifier $\eta : \mathcal{X} \to \mathcal{Y}$ with a low *target risk*

$$R_{\mathcal{D}_{\mathrm{T}}}(\eta) \stackrel{\text{def}}{=} \mathbb{P}_{(\mathbf{x}^t, y^t) \sim \mathcal{D}_{\mathrm{T}}}\Big(\eta(\mathbf{x}^t) \neq y^t\Big)\,,$$

while having no information about the labels of $\mathcal{D}_{\mathrm{T}}$.

5.2.1. Domain Divergence

To tackle the challenging domain adaptation task, many approaches bound the target error by the sum of the source error and a notion of distance between the source and the target distributions. These methods are intuitively justified by a simple assumption: the

source risk is expected to be a good indicator of the target risk when both distributions are similar. Several notions of distance have been proposed for domain adaptation (Ben-David et al., 2006; Blitzer et al., 2008; Ben-David et al., 2010; Mansour et al., 2009a,b; Germain et al., 2013). In this paper, we focus on the $\mathcal{H}\Delta\mathcal{H}$-divergence used by Blitzer et al. (2008); Ben-David et al. (2010), and based on the earlier work of Kifer et al. (2004). Note that following Blitzer et al. (2008); Ben-David et al. (2010) we assume in Definition 5.1 below that the hypothesis class $\mathcal{H}$ is a (discrete or continuous) set of binary classifiers $\eta : \mathcal{X} \to \{0, 1\}$.[1]

Definition 5.1 (Taken from (Blitzer et al., 2008; Ben-David et al., 2010))**.** *Given two domain distributions $\mathcal{D}_{\mathrm{S}}^{x}$ and $\mathcal{D}_{\mathrm{T}}^{x}$ over $\mathcal{X}$, and a hypothesis class $\mathcal{H}$, the* $\mathcal{H}\Delta\mathcal{H}$-divergence *between $\mathcal{D}_{\mathrm{S}}^{x}$ and $\mathcal{D}_{\mathrm{T}}^{x}$ is*

$$d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_{\mathrm{S}}^{x}, \mathcal{D}_{\mathrm{T}}^{x}) \stackrel{\text{def}}{=} 2 \sup_{\eta \in \mathcal{H}\Delta\mathcal{H}} \left| \mathbb{P}_{\mathbf{x}^s \sim \mathcal{D}_{\mathrm{S}}^{x}} \left[\eta(\mathbf{x}^s) = 1 \right] - \mathbb{P}_{\mathbf{x}^t \sim \mathcal{D}_{\mathrm{T}}^{x}} \left[\eta(\mathbf{x}^t) = 1 \right] \right|,$$

where $\mathcal{H}\Delta\mathcal{H} = \{\eta(\mathbf{x}) \oplus \eta'(\mathbf{x}) \,|\, \eta, \eta' \in \mathcal{H}\}$ with $\oplus$ denoting the `XOR` *operator.*

Each hypothesis $\eta \in \mathcal{H}\Delta\mathcal{H}$ labels as positive all points $\mathbf{x}$ on which a given pair of hypothese in $\mathcal{H}$ disagree. That is, the $\mathcal{H}\Delta\mathcal{H}$-divergence relies on the capacity of the hypothesis class $\mathcal{H}\Delta\mathcal{H}$ to distinguish between examples generated by $\mathcal{D}_{\mathrm{S}}^{x}$ from examples generated by $\mathcal{D}_{\mathrm{T}}^{x}$. Blitzer et al. (2008); Ben-David et al. (2010) proved that, for a symmetric hypothesis class $\mathcal{H}$ (*i.e.*, if $\eta \in \mathcal{H}$ then $(1 - \eta) \in \mathcal{H}$), one can compute the *empirical* $\mathcal{H}\Delta\mathcal{H}$*-divergence* between two samples $S \sim (\mathcal{D}_{\mathrm{S}}^{x})^n$ and $T \sim (\mathcal{D}_{\mathrm{T}}^{x})^{n'}$ by using the following formula:

$$\hat{d}_{\mathcal{H}\Delta\mathcal{H}}(S, T) \stackrel{\text{def}}{=} 2 \left(1 - \min_{\eta \in \mathcal{H}\Delta\mathcal{H}} \left[\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[\eta(\mathbf{x}^s{}_i) = 0] + \frac{1}{n'} \sum_{i=n+1}^{N} \mathbb{1}[\eta(\mathbf{x}^t{}_i) = 1] \right] \right), \quad (5.2.1)$$

where $\mathbb{1}[a]$ is the indicator function.

5.2.2. Proxy Distance

Adopting ideas from (Ben-David et al., 2006), Blitzer et al. (2008); Ben-David et al. (2010) suggested that, even if it is generally hard to compute $\hat{d}_{\mathcal{H}\Delta\mathcal{H}}(S, T)$ exactly (*e.g.*, when $\mathcal{H}$ is the space of linear classifiers on $\mathcal{X}$), we can easily approximate it by running a learning algorithm on the problem of discriminating between source and target examples. To do so, we construct a new dataset $U = \{(\mathbf{x}^s{}_i, 0)\}_{i=1}^{n} \cup \{(\mathbf{x}^t{}_i, 1)\}_{i=n+1}^{N}$, where the examples of the source sample are labeled 0 and the examples of the target sample are labeled 1. Then, the risk of the classifier trained on the new dataset U approximates the "min" part of (5.2.1). Given a generalization error ϵ on the problem of discriminating between source and target

[1] As mentioned by Ben-David et al. (2006), the same analysis holds for the multiclass setting.

examples, the $\mathcal{H}\Delta\mathcal{H}$-divergence is then approximated by

$$\zeta(U) \;=\; 2\,(1-2\epsilon)\,. \tag{5.2.2}$$

Analogous to Ben-David et al. (2006), we call the value $\zeta(U)$ the *proxy $\mathcal{H}\Delta\mathcal{H}$-distance*.

5.2.3. Generalization Bound on the Target Risk

The work of Blitzer et al. (2008) and Ben-David et al. (2010) also showed that the $\mathcal{H}\Delta\mathcal{H}$-divergence $d_{\mathcal{H}\Delta\mathcal{H}}(\mathcal{D}_{\mathrm{S}}^{x}, \mathcal{D}_{\mathrm{T}}^{x})$ is upper bounded by its empirical estimate $\hat{d}_{\mathcal{H}\Delta\mathcal{H}}(S,T)$ plus a constant complexity term that depends on the *VC dimension* of $\mathcal{H}$ and the size of samples S and T. By combining this result with a similar bound on the source risk, the following theorem is obtained.

Theorem 5.1 (Blitzer et al., 2008; Ben-David et al., 2010)**.** *Let $\mathcal{H}$ be a hypothesis class of VC dimension d. With probability $1-\delta$ over the choice of samples $S \sim (\mathcal{D}_{\mathrm{S}})^n$ and $T \sim (\mathcal{D}_{\mathrm{T}}^{x})^n$, for every $\eta \in \mathcal{H}$:*

$$\epsilon_T(\eta) \leq \hat{\epsilon}_S + \sqrt{\tfrac{4}{n}\Big(d\,\log\tfrac{2\,e\,n}{d} + \log\tfrac{4}{\delta}\Big)} + \frac{1}{2}\hat{d}_{\mathcal{H}\Delta\mathcal{H}}(S,T) + 4\sqrt{\tfrac{1}{n}\Big(2d\,\log 2n + \log\tfrac{2}{\delta}\Big)} + \beta,$$

with $\beta \geq \inf\limits_{\eta^* \in \mathcal{H}} [\epsilon_S(\eta^*) + \epsilon_T(\eta^*)]$, and $\hat{\epsilon}_S(\eta) = \frac{1}{n}\sum\limits_{(\mathbf{x},y)\in S} \mathbb{1}[\eta(\mathbf{x})\neq y]$ is the empirical source risk.

The previous result tells us that $\epsilon_T(\eta)$ can be low only when the β term is low, *i.e.*, only when there exists a classifier that can achieve a low risk on both distributions. It also tells us that in order to find a classifier with a small $\epsilon_T(\eta)$ in a given class of fixed VC dimension, the learning algorithm should minimize (in that class) a trade-off between the source risk $\hat{\epsilon}_S(\eta)$ and the empirical $\mathcal{H}\Delta\mathcal{H}$-divergence between S andf T. As pointed out by Ben-David et al. (2006), a strategy to control the $\mathcal{H}\Delta\mathcal{H}$-divergence is to find a representation of the examples where both the source and the target domain are as indistinguishable as possible. Under such a representation, a hypothesis with a low source risk will, according to Theorem 5.1, perform well on the target data.

5.3. Domain-Adversarial Neural Networks

An original aspect of our approach is to explicitly implement the idea exhibited by Theorem 5.1 into a neural network classifier. That is, to learn a model that can generalize well from one domain to another, we ensure that the internal representation of the neural network contains no discriminative information about the origin of the input (source or target), while preserving a low risk on the source (labeled) examples.

In this section, we detail the proposed approach for incorporating a "domain adaptation component" to neural networks. We start by developing the idea for the simplest possible case, *i.e.*, a single hidden layer, fully connected neural network. We then describe how to generalize the approach to arbitrary (deep) network architectures.

5.3.1. Shallow Neural Networks

Let us first consider a standard neural network (NN) architecture with a single hidden layer. For simplicity, we suppose that the input space is formed by m-dimensional real vectors. Thus, $\mathcal{X} = \mathbb{R}^m$. The hidden layer G_f learns a function $G_f : \mathcal{X} \to \mathbb{R}^D$ that maps an example into a new D-dimensional representation[2], and is parametrized by a matrix-vector pair $(\mathbf{W}, \mathbf{b}) \in \mathbb{R}^{D\times m} \times \mathbb{R}^D$:

$$G_f(\mathbf{x}; \mathbf{W}, \mathbf{b}) \;=\; \sigma\Big(\mathbf{W}\mathbf{x} + \mathbf{b}\Big), \quad \text{with} \;\; \sigma(\mathbf{a}) \stackrel{\text{def}}{=} \Big[\tfrac{1}{1+\exp(-a_i)}\Big]_{i=1}^{|\mathbf{a}|} . \tag{5.3.1}$$

Similarly, the prediction layer G_y learns a function $G_y : \mathbb{R}^D \to [0,1]^L$ that is parametrized by a pair $(\mathbf{V}, \mathbf{c}) \in \mathbb{R}^{L\times D} \times \mathbb{R}^L$:

$$G_y(G_f(\mathbf{x}); \mathbf{V}, \mathbf{c}) \;=\; \text{softmax}\Big(\mathbf{V}G_f(\mathbf{x}) + \mathbf{c}\Big), \quad \text{with} \;\; \text{softmax}(\mathbf{a}) \stackrel{\text{def}}{=} \left[\frac{\exp(a_i)}{\sum_{j=1}^{|\mathbf{a}|}\exp(a_j)}\right]_{i=1}^{|\mathbf{a}|} .$$

Here we have $L = |\mathcal{Y}|$. By using the softmax function, each component of vector $G_y(G_f(\mathbf{x}))$ denotes the conditional probability that the neural network assigns $\mathbf{x}$ to the class in $\mathcal{Y}$ represented by that component. Given a source example $(\mathbf{x}_i, y_i)$, the natural classification loss to use is the negative log-probability of the correct label: $\mathcal{L}_y\Big(G_y(G_f(\mathbf{x}_i)), y_i\Big) \stackrel{\text{def}}{=} -\log\Big[G_y(G_f(\mathbf{x}))_{y_i}\Big]$. Training the neural network then leads to the following optimization problem on the source domain:

$$\min_{\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}} \left[\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(\mathbf{W}, \mathbf{b}, \mathbf{V}, \mathbf{c}) + \lambda \cdot R(\mathbf{W}, \mathbf{b})\right], \tag{5.3.2}$$

where $\mathcal{L}_y^i(\mathbf{W}, \mathbf{b}, \mathbf{V}, \mathbf{c}) = \mathcal{L}_y\Big(G_y(G_f(\mathbf{x}^s{}_i; \mathbf{W}, \mathbf{b}); \mathbf{V}, \mathbf{c}), y_i\Big)$ is a shorthand notation for the prediction loss on the i-th example, and $R(\mathbf{W}, \mathbf{b})$ is an optional regularizer that is weighted by hyper-parameter λ.

The heart of our approach is to design a *domain regularizer* directly derived from the $\mathcal{H}\Delta\mathcal{H}$-divergence of Definition 5.1. To this end, we view the output of the hidden layer G_f ((5.3.1)) as the internal representation of the neural network. Thus, we denote the source sample representations as $S(G_f) \stackrel{\text{def}}{=} \Big\{G_f(\mathbf{x}^s) \;\Big|\; \mathbf{x} \in S\Big\}$. Similarly, given an unlabeled sample from

[2]For brevity of notation, we will sometimes drop the dependence of G_f on its parameters $(\mathbf{W}, \mathbf{b})$ and shorten $G_f(\mathbf{x}; \mathbf{W}, \mathbf{b})$ to $G_f(\mathbf{x})$.

the target domain we denote the corresponding representations $T(G_f) \stackrel{\text{def}}{=} \Big\{G_f(\mathbf{x}^t) \,\Big|\, \mathbf{x} \in T\Big\}$. Based on (5.2.1), $\hat{d}_{\mathcal{H}\Delta\mathcal{H}}\Big(S(G_f), T(G_f)\Big)$, *i.e.*, the empirical $\mathcal{H}\Delta\mathcal{H}$-divergence of a symmetric hypothesis class $\mathcal{H}$ between samples $S(G_f)$ and $T(G_f)$, is given by

$$2\left(1 - \min_{\eta \in \mathcal{H}\Delta\mathcal{H}} \left[\frac{1}{n}\sum_{i=1}^{n} I\Big[\eta(G_f(\mathbf{x}_i^S))=0\Big] + \frac{1}{n'}\sum_{i=n+1}^{N} I\Big[\eta(G_f(\mathbf{x}^t{}_i))=1\Big]\right]\right). \tag{5.3.3}$$

Let us consider $\mathcal{H}$ as the class of hyperplanes in the representation space. Inspired by the proxy $\mathcal{H}\Delta\mathcal{H}$-distance ((5.2.2)), we suggest estimating the "min" part of (5.3.3) by a *domain classification layer* G_d that learns a logistic regressor $G_d : \mathbb{R}^D \to [0, 1]$, parametrized by a vector-scalar pair $(\mathbf{u}, z) \in \mathbb{R}^D \times \mathbb{R}$, that models the probability that a given input is from the source domain $\mathcal{D}_{\text{S}}^x$ or the target domain $\mathcal{D}_{\text{T}}^x$ (Blitzer et al., 2008; Ben-David et al., 2010).[3] Thus,

$$G_d(G_f(\mathbf{x}); \mathbf{u}, z) \stackrel{\text{def}}{=} \sigma\Big(\mathbf{u}^\top G_f(\mathbf{x}) + z\Big). \tag{5.3.4}$$

Hence, the function $G_d(\cdot)$ is a *domain regressor*. We define its loss by

$$\mathcal{L}_d\Big(G_d(G_f(\mathbf{x}_i)), d_i\Big) \;=\; -d_i \log\Big[\mathbf{d}(\mathbf{f}(\mathbf{x}_i))\Big] - (1-d_i)\log\Big[1-\mathbf{d}(\mathbf{f}(\mathbf{x}_i))\Big].$$

where d_i denotes the binary variable (*domain label*) for the i-th example, which indicates whether $\mathbf{x}_i$ comes from the source distribution ($\mathbf{x}_i \sim \mathcal{D}_{\text{S}}^x$ if $d_i = 0$) or from the target distribution ($\mathbf{x}_i \sim \mathcal{D}_{\text{T}}^x$ if $d_i = 1$).

Recall that for the examples from the source distribution ($d_i = 0$), the corresponding labels $y_i \in Y$ are known at training time. For the examples from the target domains, we do not know the labels at training time, and we want to predict such labels at test time. This enables us to add a domain adaptation term to the objective of (5.3.2), giving the following regularizer:

$$R(\mathbf{W}, \mathbf{b}) \;=\; \max_{\mathbf{u}, z}\left[-\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i(\mathbf{W}, \mathbf{b}, \mathbf{u}, z) - \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i(\mathbf{W}, \mathbf{b}, \mathbf{u}, z)\Big)\right], \tag{5.3.5}$$

where $\mathcal{L}_d^i(\mathbf{W}, \mathbf{b}, \mathbf{u}, z) \;=\; \mathcal{L}_d\Big(G_d(G_f(\mathbf{x}^s{}_i; \mathbf{W}, \mathbf{b}); \mathbf{u}, z), d_i\Big)$. This regularizer seeks to approximate the $\mathcal{H}\Delta\mathcal{H}$-divergence of (5.3.3), as $2(1 - R(\mathbf{W}, \mathbf{b}))$ is a surrogate for $\hat{d}_{\mathcal{H}\Delta\mathcal{H}}\Big(S(G_f), T(G_f)\Big)$.

In line with Theorem 5.1, the optimization problem given by (5.3.2) and (5.3.5) implements a trade-off between the minimization of the source risk $\hat{\epsilon}(\cdot)$ and the divergence $\hat{d}_{\mathcal{H}\Delta\mathcal{H}}(\cdot, \cdot)$. The hyper-parameter λ is then used to tune the trade-off between these two quantities during the learning process.

[3] We note that for linear G_y, $\mathcal{H}\Delta\mathcal{H}$ contains non-linear functions (see Definition 5.1) and therefore linear G_d may be a crude approximation. We consider non-linear domain classifiers further in the text in Section 5.3.2.

For learning, we first note that we can rewrite the complete optimization objective of (5.3.2) as follows:

$$E(\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c},\mathbf{u},z) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}) - \lambda\left(\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i(\mathbf{W},\mathbf{b},\mathbf{u},z) + \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i(\mathbf{W},\mathbf{b},\mathbf{u},z)\right). \tag{5.3.6}$$

We are seeking the parameters $\hat{\mathbf{W}},\hat{\mathbf{V}},\hat{\mathbf{b}},\hat{\mathbf{c}},\hat{\mathbf{u}},\hat{z}$ that deliver a saddle point given by

$$(\hat{\mathbf{W}},\hat{\mathbf{V}},\hat{\mathbf{b}},\hat{\mathbf{c}}) = \arg\min_{\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c}} E(\mathbf{W},\mathbf{V},\mathbf{b},\mathbf{c},\hat{\mathbf{u}},\hat{z})\,,\ \text{with}\ (\hat{\mathbf{u}},\hat{z}) = \arg\max_{\mathbf{u},z} E(\hat{\mathbf{W}},\hat{\mathbf{V}},\hat{\mathbf{b}},\hat{\mathbf{c}},\mathbf{u},z)\,.$$

Thus, the optimization problem involves a minimization with respect to some parameters, as well as a maximization with respect to the others.

5.3.2. Generalization to Arbitrary Architectures

It is straightforward to generalize the single hidden layer model, presented above, to other sophisticated architectures, which might be more appropriate for the data at hand. To do so, let us now use a more general notation for the different components of the model. Namely, let $G_f(\cdot;\theta_f)$ be the D-dimensional neural network feature extractor, with parameters θ_f. Also, let $G_y(\cdot;\theta_y)$ be the part of the model that computes the network's label prediction output layer, with parameters θ_y, while $G_d(\cdot;\theta_d)$ now corresponds to the computation of the domain prediction output of the network, with parameters θ_d. Note that for preserving the theoretical guarantees of Theorem 5.1, the hypothesis class $\mathcal{H}_d$ generated by the domain prediction component G_d should include the hypothesis class $\mathcal{H}_y\Delta\mathcal{H}_y$, where $\mathcal{H}_y$ is induced by the label prediction component G_y.[4]

We denote the prediction loss and the domain loss respectively by

$$\mathcal{L}_y^i(\theta_f,\theta_y) = \mathcal{L}_y\Big(G_y(G_f(\mathbf{x}^s{}_i;\theta_f);\theta_y),y_i\Big)\,;\ \mathcal{L}_d^i(\theta_f,\theta_d) = \mathcal{L}_d\Big(G_d(G_f(\mathbf{x}^s{}_i;\theta_f);\theta_d),d_i\Big)\,.$$

Training DANN then parallels the single layer case and consists in optimizing

$$E(\theta_f,\theta_y,\theta_d) = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i(\theta_f,\theta_y) - \lambda\left(\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d^i(\theta_f,\theta_d) + \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d^i(\theta_f,\theta_d)\right), \tag{5.3.7}$$

by finding the saddle point $\hat{\theta}_f,\hat{\theta}_y,\hat{\theta}_d$ such that

$$(\hat{\theta}_f,\hat{\theta}_y) = \arg\min_{\theta_f,\theta_y} E(\theta_f,\theta_y,\hat{\theta}_d)\,,\quad \text{with}\quad \hat{\theta}_d = \arg\max_{\theta_d} E(\hat{\theta}_f,\hat{\theta}_y,\theta_d)\,. \tag{5.3.8}$$

[4]For example, one can set the architecture of G_d to be a layer-by-layer concatenation of two replicas of G_y followed by a two layer non-linear perceptron aimed to learn the `XOR`-function.

A saddle point defined by (5.3.8) can be found as a stationary point of the following gradient updates:

$$\theta_f \longleftarrow \theta_f - \mu\left(\frac{\partial \mathcal{L}_y^i}{\partial \theta_f} - \lambda \frac{\partial \mathcal{L}_d^i}{\partial \theta_f}\right), \tag{5.3.9}$$

$$\theta_y \longleftarrow \theta_y - \mu\frac{\partial \mathcal{L}_y^i}{\partial \theta_y}, \quad \text{and} \quad \theta_d \longleftarrow \theta_d - \mu\lambda\frac{\partial \mathcal{L}_d^i}{\partial \theta_d}, \tag{5.3.10}$$

where μ is the learning rate. We use stochastic estimates of these gradients, by sampling examples from the dataset.

The updates of Equations (5.3.9-5.3.10) are very similar to that of stochastic gradient descent (SGD) for a feed-forward deep model that is comprised of the feature extractor fed into the label predictor and into the domain classifier (with loss weighted by λ). The only difference is that in (5.3.9), the gradients from the class and domain predictors are subtracted, instead of being summed (the difference is important, as otherwise SGD would try to make features dissimilar across domains in order to minimize the domain classification loss). Since SGD — and its many variants, such as Adam (Kingma and Ba, 2014) or RMSProp (Tieleman and Hinton, 2012) — is the main learning algorithm implemented in most deep learning libraries, it would be convenient to frame an implementation of our stochastic saddle point search as variant of SGD.

Fortunately, such a reduction can be accomplished by introducing a special *gradient reversal layer* (GRL), defined as follows. The gradient reversal layer has no parameters associated with it. During the forward propagation, the GRL acts as an identity transformation. During the backpropagation however, the GRL takes the gradient from the subsequent level and changes its sign, *i.e.*, multiplies it by -1, before passing it to the preceding layer. Implementing such a layer using existing object-oriented packages for deep learning is simple, requiring only to define procedures for the forward propagation (identity transformation), and backpropagation (multiplication by -1). The layer requires no parameter update.

The GRL as defined above is inserted between the feature extractor G_f and the domain classifier G_d, resulting in the architecture depicted in Figure 5.1. As the backpropagation process passes through the GRL, the partial derivatives of the loss that is downstream the GRL (*i.e.*, $\mathcal{L}_d$) *w.r.t.* the layer parameters that are upstream the GRL (*i.e.*, θ_f) get multiplied by -1, *i.e.*, $\frac{\partial \mathcal{L}_d}{\partial \theta_f}$ is effectively replaced with $-\frac{\partial \mathcal{L}_d}{\partial \theta_f}$. Therefore, running SGD in the resulting model implements the updates of Equations (5.3.9-5.3.10) and converges to a saddle point of (5.3.7).

Mathematically, we can treat the gradient reversal layer as a "pseudo-function" $\mathcal{R}(\mathbf{x})$ defined by two (incompatible) equations describing its forward and backpropagation behavior:

$$\mathcal{R}(\mathbf{x}) = \mathbf{x}\,, \quad \text{and} \quad \frac{d\mathcal{R}}{d\mathbf{x}} = -\mathbf{I}\,,$$

where $\mathbf{I}$ is an identity matrix. Then, we define the objective "pseudo-function" of $(\theta_f, \theta_y, \theta_d)$ that is being optimized by the gradient descent within our method:

$$\tilde{E}(\theta_f, \theta_y, \theta_d) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_y\left(G_y(G_f(\mathbf{x}_i;\theta_f);\theta_y), y_i\right) - \tag{5.3.11}$$

$$\lambda\left(\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_d\left(G_d(\mathcal{R}(G_f(\mathbf{x}_i;\theta_f));\theta_d), d_i\right) + \frac{1}{n'}\sum_{i=n+1}^{N}\mathcal{L}_d\left(G_d(\mathcal{R}(G_f(\mathbf{x}_i;\theta_f));\theta_d), d_i\right)\right) \tag{5.3.12}$$

During training, the feature extractor and the domain regressor (classifier) are competing against each other, in an adversarial way, over the objective of (5.3.7). For this reason, we refer to networks trained according to this objective as domain-adversarial neural networks (DANN). DANN will effectively attempt to learn a hidden layer $G_f(\cdot)$ that maps an example (either source or target) into a representation allowing the output layer $G_y(\cdot)$ to accurately classify source samples, but crippling the ability of the domain regressor $G_d(\cdot)$ to detect whether each example belongs to the source or target domains.

5.4. Experiments

We now perform extensive evaluation of a deep version of DANN on a number of popular image datasets and their modifications. These include large-scale datasets of small images popular with deep learning methods, and the Office datasets (Saenko et al., 2010), which are a *de facto* standard for domain adaptation in computer vision, but have much fewer images.

5.4.1. Baselines

The following baselines are evaluated in the experiments of this subsection. The *source-only* model is trained without consideration for target-domain data (no domain classifier branch included into the network). The *train-on-target* model is trained on the target domain with class labels revealed. This model serves as an upper bound on DA methods, assuming that target data are abundant and the shift between the domains is considerable.

FIGURE 5.2. Examples of domain pairs used in the experimental evaluation of DANN.

In addition, we compare our approach against the recently proposed unsupervised DA method based on *subspace alignment* (SA) (Fernando et al., 2013), which is simple to setup and test on new datasets, but has also been shown to perform very well in experimental comparisons with other "shallow" DA methods. To boost the performance of this baseline, we pick its most important free parameter (the number of principal components) from the range $\{2, \ldots, 60\}$, so that the test performance on the target domain is maximized. To apply SA in our setting, we train a source-only model and then consider the activations of the last hidden layer in the label predictor (before the final linear classifier) as descriptors/features, and learn the mapping between the source and the target domains (Fernando et al., 2013). Since the SA baseline requires training a new classifier after adapting the features, and in order to put all the compared settings on an equal footing, we retrain the last layer of the label predictor using a standard linear SVM (Fan et al., 2008) for all four considered methods (including ours; the performance on the target domain remains approximately the same after the retraining).

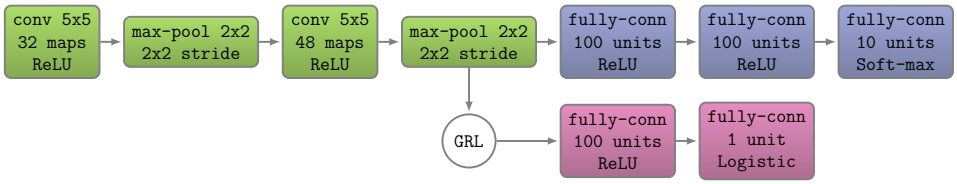
For the OFFICE dataset (Saenko et al., 2010), we directly compare the performance of our full network (feature extractor and label predictor) against recent DA approaches using previously published results.

5.4.2. CNN Architectures and Training Procedure

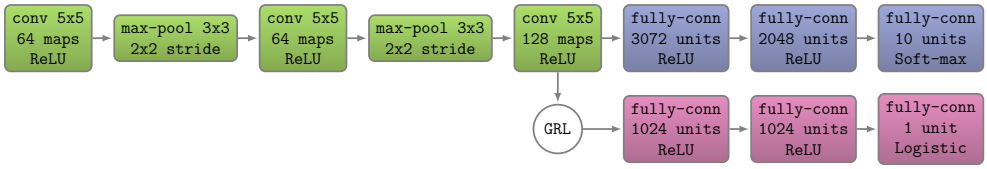
In general, we compose feature extractor from two or three convolutional layers, picking their exact configurations from previous works. More precisely, four different architectures were used in our experiments. The first three are shown in Figure 5.3. For the OFFICE domains, we use pre-trained `AlexNet` from the `Caffe` package (Jia et al., 2014). The adaptation architecture is identical to (Tzeng et al., 2014).[5]

For the domain adaption component, we use three ($x \to 1024 \to 1024 \to 2$) fully connected layers, except for MNIST where we used a simpler ($x \to 100 \to 2$) architecture to speed up

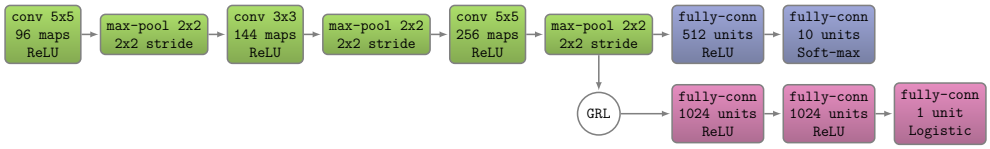
[5]A 2-layer domain classifier ($x \to 1024 \to 1024 \to 2$) is attached to the 256-dimensional bottleneck of `fc7`.



(A) MNIST architecture; inspired by the classical LeNet-5 (LeCun et al., 1998).



(B) SVHN architecture; adopted from Srivastava et al. (2014).

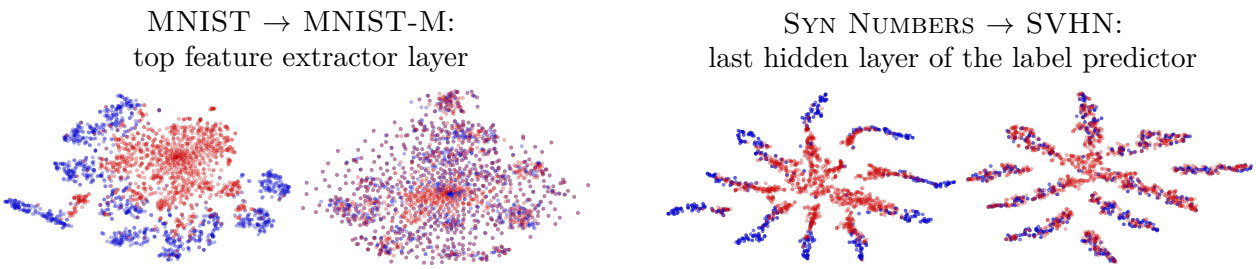


(C) GTSRB architecture; we used the single-CNN baseline from Cireşan et al. (2012) as our starting point.

FIGURE 5.3. CNN architectures used in the experiments. Boxes correspond to transformations applied to the data. Color-coding is the same as in Figure 5.1.

the experiments. Admittedly, these choices for domain classifier are arbitrary, and better adaptation performance might be attained if this part of the architecture is tuned.

For the loss functions, we set $\mathcal{L}_y$ and $\mathcal{L}_d$ to be the logistic regression loss and the binomial cross-entropy respectively. Following Srivastava et al. (2014) we also use dropout and ℓ^2-norm restriction when we train the SVHN architecture. The learning rate is adjusted during the stochastic gradient descent using the formula $\mu_p = \frac{\mu_0}{(1+\alpha \cdot p)^\beta}$, where p is the training progress linearly changing from 0 to 1, $\mu_0 = 0.01$, $\alpha = 10$ and $\beta = 0.75$ (the schedule was optimized to promote convergence and low error on the *source* domain). A momentum term of 0.9 is also used. The domain adaptation parameter λ is initiated at 0 and is gradually changed to 1 using the schedule $\lambda_p = \frac{2}{1+\exp(-\gamma \cdot p)} - 1$, where γ was set to 10 in all experiments (the schedule was not optimized/tweaked). This strategy allows the domain classifier to be less sensitive to noisy signal at the early stages of the training procedure. Note however that these λ_p were used only for updating the *feature extractor* component G_f. For updating the



(A) Non-adapted (B) Adapted (A) Non-adapted (B) Adapted

FIGURE 5.4. The effect of adaptation on the distribution of the extracted features (best viewed in color). The figure shows t-SNE (van der Maaten, 2013) visualizations of the CNN's activations **(a)** when no adaptation was performed and **(b)** when our adaptation procedure was incorporated into training. *Blue* points correspond to the source domain examples, while *red* ones correspond to the target domain.

domain classification component, we used a fixed $\lambda = 1$, to ensure that the latter trains as fast as the *label predictor* G_y.[6]

Finally, note that the model is trained on 128-sized batches (images are preprocessed by the mean subtraction). A half of each batch is populated by the samples from the source domain (with known labels), the rest constitutes the target domain (with labels not revealed to the algorithms except for the train-on-target baseline).

5.4.3. Visualizations

We use t-SNE (van der Maaten, 2013) projection to visualize feature distributions at different points of the network, while color-coding the domains (Figure 5.4). We observe a strong correspondence between the success of the adaptation in terms of the classification accuracy for the target domain, and the overlap between the domain distributions in such visualizations.

5.4.4. Results On Image Datasets

We now discuss the experimental settings and the results. In each case, we train on the source dataset and test on a different target domain dataset, with considerable shifts between domains (see Figure 5.2). The results are summarized in Table 5. I and Table 5. II.

MNIST → MNIST-M. Our first experiment deals with the MNIST dataset (LeCun et al., 1998) (source). In order to obtain the target domain (MNIST-M) we blend digits from the

[6]Equivalently, one can use the same λ_p for both feature extractor and domain classification components, but use a learning rate of μ/λ_p for the latter.

Method / Source / Target	MNIST / MNIST-M	Syn Numbers / SVHN	SVHN / MNIST	Syn Signs / GTSRB
Source only	.5225	.8674	.5490	.7900
SA	.5690 (4.1%)	.8644 (−5.5%)	.5932 (9.9%)	.8165 (12.7%)
DANN	**.7666** (52.9%)	**.9109** (79.7%)	**.7385** (42.6%)	**.8865** (46.4%)
Train on target	.9596	.9220	.9942	.9980

Table 5. I. Digit image classifications accuracies. The first row corresponds to the lower performance bound (no adaptation is performed). The last row corresponds to training on the target data with known labels (upper bound on the DA performance). For each of the two DA methods (ours and by Fernando et al. (2013)) we show how much of the gap between the lower and the upper bounds was covered (in brackets).

Method / Source / Target	Amazon / Webcam	DSLR / Webcam	Webcam / DSLR
GFK(PLS, PCA) (Gong et al., 2012)	.197	.497	.6631
SA* (Fernando et al., 2013)	.450	.648	.699
DLID (Chopra et al., 2013)	.519	.782	.899
DDC (Tzeng et al., 2014)	.618	.950	.985
DAN (Long and Wang, 2015)	.685	.960	.990
Source only	.642	.961	.978
DANN	**.730**	**.964**	**.992**

Table 5. II. Accuracy evaluation of different DA approaches on the standard Office (Saenko et al., 2010) dataset. All methods (except SA) are evaluated in the "fully-transductive" protocol (some results are reproduced from (Long and Wang, 2015)).

original set over patches randomly extracted from color photos from BSDS500 (Arbeláez et al., 2011). This operation is formally defined for two images I^1, I^2 as $I^{out}_{ijk} = |I^1_{ijk} - I^2_{ijk}|$, where i, j are the coordinates of a pixel and k is a channel index. In other words, an output sample is produced by taking a patch from a photo and inverting its pixels at positions corresponding to the pixels of a digit. For a human the classification task becomes only slightly harder compared to the original dataset (the digits are still clearly distinguishable) whereas for a CNN trained on MNIST this domain is quite distinct, as the background and the strokes are no longer constant in intensity. Consequently, the source-only model

performs poorly. Our approach succeeded at aligning feature distributions (Figure 5.4), which led to successful adaptation results (considering that the adaptation is unsupervised). At the same time, the improvement over source-only model achieved by subspace alignment (SA) (Fernando et al., 2013) is quite modest, thus highlighting the difficulty of the adaptation task.

Synthetic numbers → SVHN. To address a common scenario of training on synthetic data and testing on real data, we use Street-View House Number dataset SVHN (Netzer et al., 2011) as the target domain and synthetic digits as the source. The latter (SYN NUMBERS) consists of ≈ 500,000 images generated by ourselves from Windows™ fonts by varying the text (that includes different one-, two-, and three-digit numbers), positioning, orientation, background and stroke colors, and the amount of blur. The degrees of variation were chosen manually to simulate SVHN, however the two datasets are still rather distinct, the biggest difference being the structured clutter in the background of SVHN images.

The proposed backpropagation-based technique works well covering almost 80% of the gap between training with source data only and training on target domain data with known target labels. In contrast, SA (Fernando et al., 2013) results in a slight classification accuracy drop (probably due to the information loss during the dimensionality reduction), indicating that the adaptation task is even more challenging than in the case of the MNIST experiment.

MNIST ↔ SVHN. In this experiment, we further increase the gap between distributions, and test on MNIST and SVHN, which are significantly different in appearance. Training on SVHN even without adaptation is challenging — classification error stays high during the first 150 epochs. In order to avoid ending up in a poor local minimum we, therefore, do not use learning rate annealing here. Obviously, the two directions (MNIST → SVHN and SVHN → MNIST) are not equally difficult. As SVHN is more diverse, a model trained on SVHN is expected to be more generic and to perform reasonably on the MNIST dataset. This, indeed, turns out to be the case and is supported by the appearance of the feature distributions. We observe a quite strong separation between the domains when we feed them into the CNN trained solely on MNIST, whereas for the SVHN-trained network the features are much more intermixed. This difference probably explains why our method succeeded in improving the performance by adaptation in the SVHN → MNIST scenario (see Table 5. I) but not in the opposite direction (SA is not able to perform adaptation in this case either). Unsupervised adaptation from MNIST to SVHN gives a failure example for our approach: it doesn't manage to improve upon the performance of the non-adapted model which achieves ≈ 0.25 accuracy (at the time of publication there were no unsupervised DA methods capable of performing such adaptation).

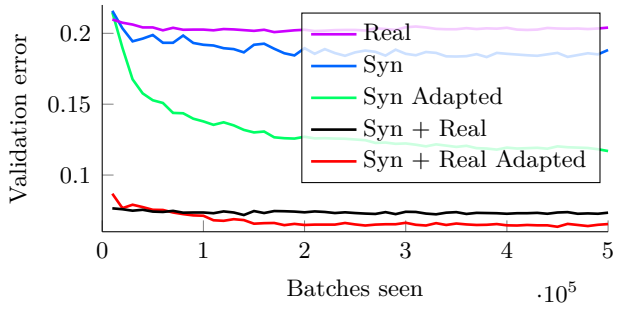


FIGURE 5.5. Results for the traffic signs classification in the semi-supervised setting. *Syn* and *Real* denote available labeled data (100,000 synthetic and 430 real images respectively); *Adapted* means that $\approx$ 31,000 unlabeled target domain images were used for adaptation. The best performance is achieved by employing both the labeled samples and the large unlabeled corpus in the target domain.

Synthetic Signs $\rightarrow$ GTSRB. Overall, this setting is similar to the SYN NUMBERS $\rightarrow$ SVHN experiment, except the distribution of the features is more complex due to the significantly larger number of classes (43 instead of 10). For the source domain we obtained 100,000 synthetic images (which we call SYN SIGNS) simulating various imaging conditions. In the target domain, we use 31,367 random training samples for unsupervised adaptation and the rest for evaluation. Once again, our method achieves a sensible increase in performance proving its suitability for the synthetic-to-real data adaptation.

As an additional experiment, we also evaluate the proposed algorithm for semi-supervised domain adaptation, *i.e.*, when one is additionally provided with a small amount of labeled target data. Here, we reveal 430 labeled examples (10 samples per class) and add them to the training set for the label predictor. Figure 5.5 shows the change of the validation error throughout the training. While the graph clearly suggests that our method can be beneficial in the semi-supervised setting, thorough verification of semi-supervised setting is left for future work.

Office Dataset. We finally evaluate our method on OFFICE dataset, which is a collection of three distinct domains: AMAZON, DSLR, and WEBCAM. Unlike previously discussed datasets, OFFICE is rather small-scale with only 2817 labeled images spread across 31 different categories in the largest domain. The amount of available data is crucial for a successful training of a deep model, hence we opted for the fine-tuning of the CNN pre-trained on the ImageNet (ALEXNET from the `Caffe` package (Jia et al., 2014)) as it is done in some recent DA works (Donahue et al., 2014; Tzeng et al., 2014; Hoffman et al., 2013; Long and Wang, 2015). We make our approach more comparable with Tzeng et al. (2014) by using exactly

the same network architecture replacing domain mean-based regularization with the domain classifier.

Following previous works, we assess the performance of our method across three transfer tasks most commonly used for evaluation. Our training protocol is adopted from (Gong et al., 2013; Chopra et al., 2013; Long and Wang, 2015) as during adaptation we use all available labeled source examples and unlabeled target examples (the premise of our method is the abundance of unlabeled data in the target domain). Also, all source domain data are used for training. Under this "fully-transductive" setting, our method is able to improve previously-reported state-of-the-art accuracy for unsupervised adaptation very considerably (Table 5. II), especially in the most challenging AMAZON → WEBCAM scenario (the two domains with the largest domain shift). Interestingly, in all three experiments we observe a slight over-fitting (performance on the target domain degrades while accuracy on the source continues to improve) as training progresses, however, it doesn't ruin the validation accuracy. Moreover, switching off the domain classifier branch makes this effect far more apparent, from which we conclude that our technique serves as a regularizer.

5.5. CONCLUSION

We have presented a powerful approach to domain adaptation of feed-forward neural networks, which allows large-scale training based on large amount of annotated data in the source domain and large amount of unannotated data in the target domain. Similarly to many previous shallow and deep DA techniques, the adaptation is achieved through aligning the distributions of features across the two domains. However, unlike previous approaches, the alignment is accomplished through standard backpropagation training. Moreover, our approach is motivated and supported by the domain adaptation theory of Ben-David et al. (2006); Blitzer et al. (2008); Ben-David et al. (2010).

The main idea behind DANN is to enjoin a network hidden layer to learn a representation which is predictive of the source example labels, but uninformative about the domain of the input (source or target). We have shown that our approach is flexible and achieves state-of-the-art results on a variety of DA benchmarks.

A convenient aspect of our approach is that the domain adaptation component can be added to almost any neural network architecture that is trainable with backpropagation, allowing simple implementation within virtually any deep learning package through the introduction of a simple gradient reversal layer. The proposed method achieves impressive

results in the image classification setting but can also be applied for a wide range of other tasks both in computer vision and beyond.

PROLOGUE TO THE FOURTH ARTICLE

Article Details

Synthesizing Programs for Images using Reinforced Adversarial Learning. Yaroslav Ganin, Tejas Kulkarni, Igor Babuschkin, S. M. Ali Eslami, and Oriol Vinyals. *ArXiv Preprint: 1804.01118, 2018.*

Personal Contributions. The project started with a discussion with Ali Eslami and Oriol Vinyals in June, 2017. We decided to work on an agent controlling an external painting or CAD environment in order to produce 2D images. I suggested using the GAN framework for that and implemented the model for sampling from the data distribution. I also found a suitable open-source software (`libmypaint` by libmypaint contributors (2018)) that could serve as a base for the painting environment and ported it for use with `Tensorflow`. We then switched our focus to the reconstruction regime and I tried several simple baselines which did not seem to work well. As an alternative, I proposed to adapt the approach from the successful unconditional GAN experiments. That turned out to be a break-through and allowed to reliably train conditional models. I wrote all the corresponding code and ran a series of experiments for both the painting environment and a `MuJoCo`-based environment (which was also implemented by me).

We decided to submit the work to ICML 2018. Tejas Kulkarni and I wrote the initial draft. I focused on the method description and the experimental results, while Tejas Kulkarni worked on the introduction and related work. He also used my code to test several hyperparameter settings for the MuJoCo Scenes dataset. Ali Eslami contributed to the introductory part of the paper and together with Oriol Vinyals and Igor Babuschkin did general editing of the manuscript. I performed a significant amount of additional editing for the `arXiv` preprint and for the camera-ready version of the paper.

Context

I spent some time working on GAN models for visual data during my internship at MILA in the spring of 2016. One observation that I made at the time was that NN-based latent-to-visible decoders often fail to capture global structure wasting their capacity on seemingly unimportant local details. This was one of the reasons I was eager to try combining the adversarial framework with third-party off-the-shelf renderers. Moreover, the idea of training an agent that could control content creation tools normally used by humans (*e.g.*, `GIMP`, `Photoshop` or `3ds Max`) was very appealing.

The ambitious goal of the project (at least, from my perspective) was to build a model capable of creating and reproducing real paintings. We were also interested in venturing into the domain of 3D but since I was more familiar (and fascinated) with plain drawings we mostly focused on that type of data.

When we were planning our first steps we had two options: either start with a reconstruction model (in which case we would not need any GANs) or train an agent that can sample random images from some target distribution. Solely because of my affinity for the adversarial framework, I decided to go with the latter. This choice later turned out to be crucial for the development of the entire project. As I mentioned above, after we verified that our GAN variant works for unconditional generation, we attempted to use simpler objectives for the reconstruction setting. Unfortunately, none of the non-adversarial approaches I tried seemed to work. Since we already had a working model, albeit for a different setting, a natural idea was to modify it so it could be used for inverse graphics. This allowed us to successfully train our reconstruction agent for the first time. Later experiments demonstrated that the conditional GAN was consistently outperforming naive reward signals across the board.

Despite the progress we made, the training procedure remained somewhat flaky (which is a typical phenomenon for RL). One possible remedy to that problem was population-based training (PBT) which improved learning stability for a range of projects at DeepMind. I was the first one to try this technique for training GANs. It took some time to make PBT work for our setting mainly because it was difficult to come up with a reliable fitness score for the instances in the population.

With PBT, adversarial objectives and improved convolutional architectures, our agent managed to achieve impressive results on several distinct datasets some of which were never used in the context of inverse graphics. This success (although we did not quite reach some of the initial goals) motivated us to summarize our findings in an ICML 2018 submission.

Contributions

Two main aspects that diffirentiate our project from existing works combining RL with the adversarial framework are the high dimensionality of the state space and the sparsity of rewards. To the best of our knowledge, SPIRAL is the first model to successfully overcome those challenges while being trained entirely from scratch. Compared to the inverse graphics literature, our approach is more general as it does not rely on any domain knowledge leaving the burden of extracting relevant information to a learned loss driven by a GAN discriminator. This allows us to apply our framework to real-world datasets like CelebA. Moreover, as we empirically demonstrate in the paper, the proposed method can work in absence of access to intermediate states of the simulator which makes it suitable for the full-fledged program synthesis.

As a generative model, SPIRAL is distinct from typical neural decoder-based approaches since it features an interpretable latent space. This opens up a possibility of supplying prior knowledge about the task at hand in a natural way.

We also believe that the general idea of using learned losses in similar settings can lead to major breakthoughs. This goes beyond image-to-image comparison considered in the present work – we give a glimpse of exciting future directions in the last section of the paper.

Chapter 6

SYNTHESIZING PROGRAMS FOR IMAGES USING REINFORCED ADVERSARIAL LEARNING

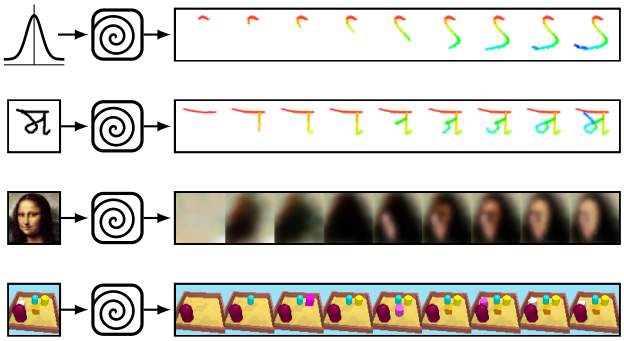


FIGURE 6.1. **SPIRAL** takes as input either random noise or images and iteratively produces plausible samples or reconstructions via graphics program synthesis. The first row depicts an unconditional run given random noise. The second, third and fourth rows depict conditional execution given an image with a handwritten character, the Mona Lisa, and objects arranged in a 3D scene.

6.1. INTRODUCTION

Recovering *structured* representations from raw sensations is an ability that humans readily possess and frequently use. Given a picture of a hand-written character, decomposing it into strokes can make it easier to classify or re-imagine that character, and similarly, knowing the underlying layout of a room can aid with planning, navigation and interaction in that room. Furthermore, this structure can be exploited for generalization, rapid learning, and even communication with other agents. It is commonly believed that humans exploit *simulations*

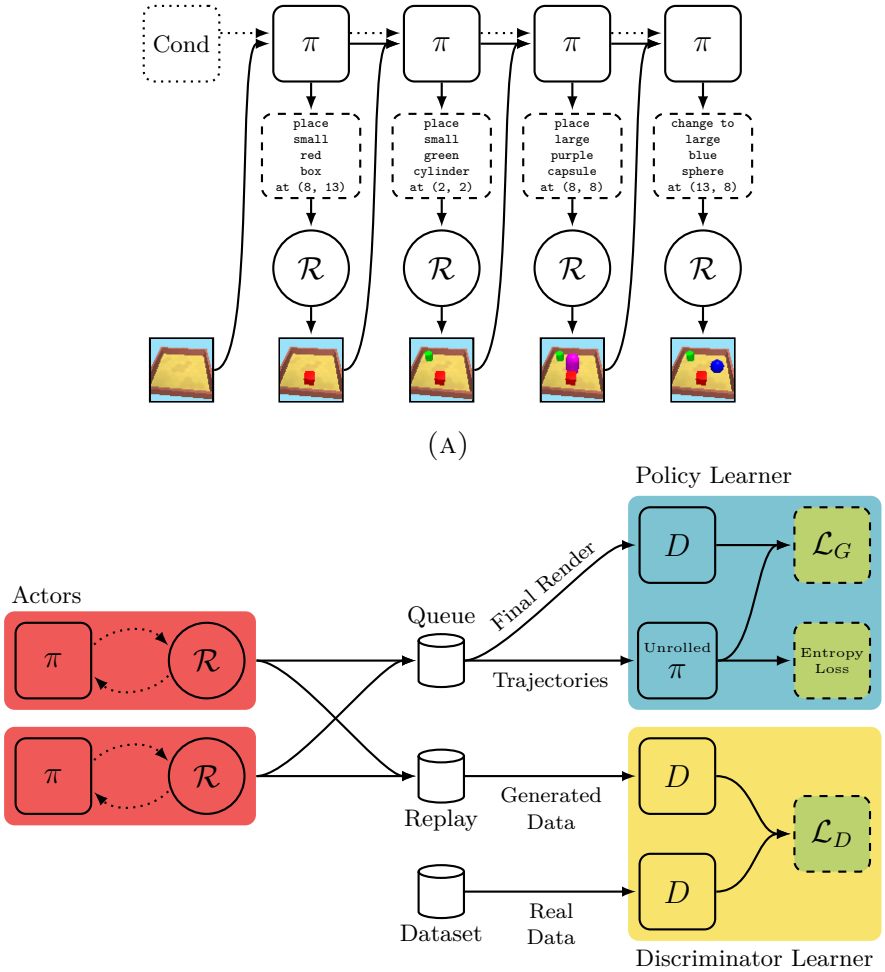


FIGURE 6.2. **The SPIRAL architecture. (A)** An execution trace of the SPIRAL agent. The policy outputs program fragments which are rendered into an image at each step via a graphics engine $\mathcal{R}$. The agent can make use of these intermediate renders to adjust its policy. The agent only receives a reward in the final step of execution. **(B)** Distributed training of SPIRAL. A collection of actors (in our experiments, up to 64), asynchronously and continuously produce execution traces. This data, along with a training dataset of ground-truth renderings, are passed to a Wasserstein discriminator on a separate GPU for adversarial training. The discriminator assesses the similarity of the final renderings of the traces to the ground-truth. A separate off-policy GPU learner receives batches of execution traces and trains the agent's parameters via policy-gradients to maximize the reward assigned to them by the discriminator, *i.e.*, to match the distribution of the ground truth dataset.

to learn this skill (Lake et al., 2017). By experimenting with a pen and a piece of paper we learn how our hand movements lead to written characters, and via imagination we learn how architectural layouts manifest themselves in reality.

In the visual domain, inversion of a renderer for the purposes of scene understanding is typically referred to as inverse graphics (Mansinghka et al., 2013; Kulkarni et al., 2015a). Training vision systems using the inverse graphics approach has remained a challenge. Renderers typically expect as input *programs* that have sequential semantics, are composed of discrete symbols (*e.g.*, keystrokes in a CAD program), and are long (tens or hundreds of symbols). Additionally, matching rendered images to real data poses an optimization problem as black-box graphics simulators are not differentiable in general.

To address these problems, we present a new approach for interpreting and generating images using *deep reinforced adversarial learning*. In this approach, an adversarially trained agent generates visual programs which are in turn executed by a graphics engine to generate images, either conditioned on data or unconditionally. The agent is rewarded by fooling a discriminator network, and is trained with distributed reinforcement learning without any extra supervision. The discriminator network itself is trained to distinguish between rendered and real images.

Our contributions are as follows:

- An adversarially trained reinforcement learning agent that interprets and generates images in the space of visual programs. Crucially, the architecture of our agent is agnostic both to the semantics of the visual program and to the domain.
- Scaling inverse graphics to real world and procedural datasets without the need for labels. In particular, our model discovers pen strokes that give rise to MNIST and Omniglot characters, brush strokes that give rise to celebrity faces, and scene descriptions that, once rendered, reconstruct an image of a 3D scene (Figure 6.1).
- Evidence that utilizing a discriminator's output as the reward signal for reinforcement learning is significantly better at optimizing the pixel error between renderings and data, compared to directly optimizing pixel error.
- A showcase of state-of-the-art deep reinforcement learning techniques, which can provide a scaling path for inverse graphics, and could lead to broader implications for program synthesis in future work.

6.2. Related Work

The idea of inverting simulators to interpret images has been explored extensively in recent years (Nair et al., 2008; Paysan et al., 2009; Mansinghka et al., 2013; Loper and Black, 2014; Kulkarni et al., 2015a; Jampani et al., 2015). Structured object-attribute based 'de-rendering' models have been proposed for interpretation of images (Wu et al., 2017b) and videos (Wu et al., 2017a). Concurrent work has explored the use of *Constructive Solid Geometry* primitives for explaining binary images (Sharma et al., 2017). Loper and Black (2014) proposed the idea of differentiable inverse graphics, which is efficient for optimizing continuous variables but cannot handle discrete variables. Earlier work has also explored using reinforcement learning for automatic generation of single brush strokes (Xie et al., 2013). However, scaling these approaches to larger real-world datasets, particularly at test-time, has remained a challenge.

Inferring motor programs for the reconstruction of MNIST digits was first studied in (Nair and Hinton, 2006). The generative model is parametrized by two pairs of opposing springs whose stiffness is controlled by a motor program. The training procedure involved starting with a prototype program and its corresponding observation. Random noise was then added to this prototype in order to produce new training examples until the generated distribution stretched to cover the manifold of the training digits. In contrast, our model automatically learns the training curriculum via the discriminator and the same agent is suitable for a range of scene understanding problems, including those in 3D.

Visual program induction has recently been studied in the context of hand-written characters on the Omniglot dataset (Lake et al., 2015). This model achieves impressive performance but requires parses from a hand-crafted algorithm to initialize training and was not demonstrated to generalize beyond hand-written characters. Ellis et al. (2017) proposed a visual program induction model to infer LaTeX programs for diagram understanding. More recently, the `sketch-rnn` model (Ha and Eck, 2017) used sequence-to-sequence learning (Sutskever et al., 2014) to produce impressive sketches both unconditionally and conditioned on data. However, similar to the aforementioned works and unlike SPIRAL, the model requires supervision in the form of sketches and corresponding sequences of strokes.

In the neural network community, there have been analogous attempts at inferring and learning feed-forward or recurrent procedures for image generation (LeCun et al., 2015; Hinton and Salakhutdinov, 2006; Goodfellow et al., 2014; Ackley et al., 1987; Kingma and Welling, 2013; Oord et al., 2016; Kulkarni et al., 2015c; Eslami et al., 2016; Reed et al., 2017; Gregor et al., 2015). These models demonstrate impressive image generation capabilities but generally lack the ability to infer structured representations of images.

Our approach employs adversarial training techniques, first used for generative modeling (Goodfellow et al., 2014) and domain adaptation (Ganin and Lempitsky, 2015). Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) were orignally used for image generation but have now been successfully applied to model audio, text and motor behaviors (Ho and Ermon, 2016; Merel et al., 2017). Perhaps the most interesting extension in our context is their use in domain transfer, where images from one domain (*e.g.*, segmentations) were mapped to another (*e.g.*, pixels). Models such as `pix2pix` (Isola et al., 2017), `CycleGAN` (Zhu et al., 2017) and AIGN (Tung et al., 2017) fall in this category.

The SPIRAL agent builds upon this literature, has minimal hand-crafting in its design, requires no supervision in the form of pairs of programs and corresponding images and, as we demonstrate in the following sections, is applicable across a wide range of domains.

6.3. The SPIRAL Agent

6.3.1. Overview

Our goal is to construct a generative model G capable of sampling from some target data distribution p_d. To that end, we propose using an external black-box rendering simulator $\mathcal{R}$ that accepts a sequence of commands $a = (a_1, a_2, \ldots, a_N)$ and transforms them into the domain of interest, *e.g.*, a bitmap. For example, $\mathcal{R}$ could be a CAD program rendering descriptions of primitives into 3D scenes. Thus, our task is equivalent to recovering a distribution p_a such that $p_d \approx \mathcal{R}(p_a)$. We model p_a with a recurrent neural network π which we call the *policy* network (or, somewhat sloppily, the *agent*). The generation process G, which consists of a policy π and a renderer $\mathcal{R}$, is illustrated in Figure 6.2a.

In order to optimize π, we employ the adversarial framework (Goodfellow et al., 2014). In this framework, the *generator* (denoted as G) aims to maximally confuse a *discriminator* network D which is trained to distinguish between the samples drawn from p_d and the samples generated by the model. As a result, the distribution defined by G (denoted as p_g) gradually becomes closer to p_d. Crucially, and unlike previous work, our training procedure does not require any strong supervision in the form of aligned examples from p_d and p_a.

We give the concrete training objectives for G and D below.

6.3.2. Objectives

In our experiments, we found that the original minimax objective from Goodfellow et al. (2014) was hard to optimize in the context of our model, so we opted for using a variant that

employs Wasserstein distance as a measure of divergence between distributions (Gulrajani et al., 2017), as it appears to handle vastly dissimilar p_g and p_d more gracefully. In this case, the discriminator is considered "confused" if it assigns the same scores (in expectation) to the inputs coming both from p_d and p_g. We note that our approach can be used in conjunction with other forms of GAN objectives as well.

Discriminator. Following Gulrajani et al. (2017), we define the objective for D as:

$$\mathcal{L}_D = -\mathbb{E}_{\mathbf{x}\sim p_d}\left[D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x}\sim p_g}\left[D(\mathbf{x})\right] + R, \tag{6.3.1}$$

where R is a regularization term softly constraining D to stay in the set of Lipschitz continuous functions (for some fixed Lipschitz constant). It is worth noting that the solution to (6.3.1) is defined up to an additive constant. Due to the nature of how we use discriminator outputs during training of π, we found it beneficial to resolve this ambiguity by encouraging $D(\mathbf{x})$ to be close to 0 on average for $\mathbf{x} \sim \frac{1}{2}p_g + \frac{1}{2}p_d$.

Generator. We formally define π as a network that at every time step t predicts a distribution over all possible commands $\pi_t = \pi(a_t|s_t;\theta)$, where s_t is the recurrent state of the network, and θ is a set of learnable parameters. Given a sequence of samples $(a_t \,|\, a_t \sim \pi_t, 1 \leq t \leq N)$, a sample from p_g is then computed as $\mathcal{R}(a_1, a_2, \ldots, a_N)$. Since the generator is an arbitrary non-differentiable function, we cannot optimize

$$\mathcal{L}_G = -\mathbb{E}_{\mathbf{x}\sim p_g}\left[D(\mathbf{x})\right] \tag{6.3.2}$$

with naive gradient descent. Therefore we pose this problem as maximization of the expected return which can be solved using standard techniques from reinforcement learning. Specifically, we employ a variant of the REINFORCE (Williams, 1992) algorithm, advantage actor-critic (A2C):

$$\mathcal{L}_G = -\sum_t \log \pi(a_t \,|\, s_t;\theta)\left[R_t - V^\pi(s_t)\right], \tag{6.3.3}$$

where V^π is an approximation to the value function which is considered to be independent of θ, and $R_t = \sum_t^N r_t$ is a 1-sample Monte-Carlo estimate of the return. Optimizing (6.3.3) recovers the solution to (6.3.2) if the rewards are set to:

$$r_t = \begin{cases} 0\,, & t < N\,, \\ D(\mathcal{R}(a_1, a_2, \ldots, a_N))\,, & t = N\,. \end{cases} \tag{6.3.4}$$

One interesting aspect of this new formulation is that we can also bias the search by introducing intermediate rewards which may depend not only on the output of $\mathcal{R}$ but also on

commands used to generate that output. We present several examples of such rewards in Section 6.4.

6.3.3. Conditional Generation

So far, we have described the case of unconditional generation, but in many situations it is useful to condition the model on auxiliary input (Mirza and Osindero, 2014). For instance, one might be interested in finding a specific program that generates a given image $\mathbf{x}_{\text{target}}$. That could be achieved by supplying $\mathbf{x}_{\text{target}}$ both to the policy and to the discriminator networks. In other words,

$$p_g = \mathcal{R}\left(p_a(a|\mathbf{x}_{\text{target}})\right) , \tag{6.3.5}$$

while p_d becomes a Dirac δ-function centered at $\mathbf{x}_{\text{target}}$. The first two terms in (6.3.1) thus reduce to

$$- D(\mathbf{x}_{\text{target}} \,|\, \mathbf{x}_{\text{target}}) + \mathbb{E}_{\mathbf{x}\sim p_g}\left[D(\mathbf{x}|\mathbf{x}_{\text{target}})\right] . \tag{6.3.6}$$

It can be shown that for this particular setting of p_g and p_d, the ℓ^2-distance is an optimal discriminator. However, in general it is not a unique solution to (6.3.1) and may be a poor candidate to be used as the generator's reward signal (see Section A.1 in the appendix for details). In Section 6.4, we empirically evaluate both ℓ^2 and a dynamically learned D and conclude that those two options are not equivalent in practice (for example, see Figure 6.9a).

6.3.4. Distributed Learning

Our training pipeline is outlined in Figure 6.2b. It is an extension of the recently proposed *IMPALA* architecture (Espeholt et al., 2018). For training, we define three kinds of workers:

- *Actors* are responsible for generating the training trajectories through interaction between the policy network and the rendering simulator. Each trajectory contains a sequence $((\pi_t, a_t) \,|\, 1 \leq t \leq N)$ as well as all intermediate renderings produced by $\mathcal{R}$.
- A *policy learner* receives trajectories from the actors, combines them into a batch and updates π by performing an SGD step on $\mathcal{L}_G$ (6.3.2). Following common practice (Mnih et al., 2016), we augment $\mathcal{L}_G$ with an entropy penalty encouraging exploration.
- In contrast to the base IMPALA setup, we define an additional *discriminator learner*. This worker consumes random examples from p_d, as well as generated data (final renders) coming from the actor workers, and optimizes $\mathcal{L}_D$ (6.3.1).

In the original paper introducing WGAN with gradient penalty (Gulrajani et al., 2017), the authors note that in order to obtain better performance, the discriminator has to be updated more frequently than the generator. In our setting, generation of each model sample is expensive since it involves multiple invocations of an external simulator. We therefore do

not omit any trajectories in the policy learner. Instead, we decouple the D updates from the π updates by introducing a *replay buffer* that serves as a communication layer between the actors and the discriminator learner. That allows the latter to optimize D at a higher rate than the training of the policy network due to the difference in network sizes (π is a multi-step RNN, while D is a plain CNN). We note that even though sampling from a replay buffer inevitably leads to smoothing of p_g[1], we found this setup to work well in practice.

6.4. Experiments

6.4.1. Datasets

We validate our approach on three real-world and one synthetic image dataset. The first, MNIST (LeCun et al., 1998), is regarded as a standard sanity check for newly proposed generative models. It contains 70 000 examples of handwritten digits, of which 10 000 constitute a test set. Each example is a 28×28 grayscale image. Although the dataset is often considered "solved" by neural decoder-based approaches (including GANs and VAEs), these approaches do not focus on recovering interpretable structure from the data. We, therefore, choose not to discard MNIST from the empirical evaluation since it is likely that additional constraints increase the difficulty of the modeling task.

The second dataset, Omniglot (Lake et al., 2015), comprises of 1623 handwritten characters from 50 alphabets. Compared to MNIST, this dataset introduces three additional challenges: higher data variability, higher complexity of symbols (*e.g.*, disjoint subcurves) and fewer (only 20) data points per symbol class.

Since both MNIST and Omniglot represent a restricted line drawing domain, we diversify our set of experiments by testing the proposed method on CelebA (Liu et al., 2015). The dataset contains over 200 000 color headshots of celebrities with large variation in poses, backgrounds and lighting conditions.

Lastly, we are interested in evaluating our approach on the task of unsupervised 3D scene understanding which is a crucial precursor for manipulating and reasoning about objects in the real world. To that end, we created a procedural dataset called MuJoCo Scenes consisting of renders of simple 3D primitives (up to 5 objects) scattered around a square platform (see Figure 6.8). The training set is comprised of 50 000 RGB images generated by means of the `MuJoCo` environment discussed in the next section.

[1]The replay always contains samples from the older versions of the agent and therefore does not exactly follow p_g.

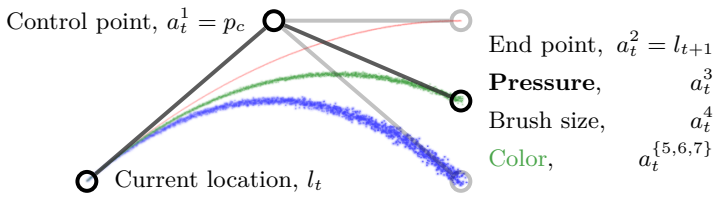


FIGURE 6.3. Illustration of the **agent's action space** in the `libmypaint` environment. We show three different strokes (red, green, blue) that can result from a single instruction from the agent to the renderer. Starting from a position on the canvas, the agent selects the coordinates of the next end point, the coordinates of the intermediate control point, as well as the brush size, pressure and color. See Section 6.4.2 for details.

In each case, we rescale the images to 64×64, which allows us to reuse the same network architectures in all the experiments, demonstrating the generality of our method.

6.4.2. Environments

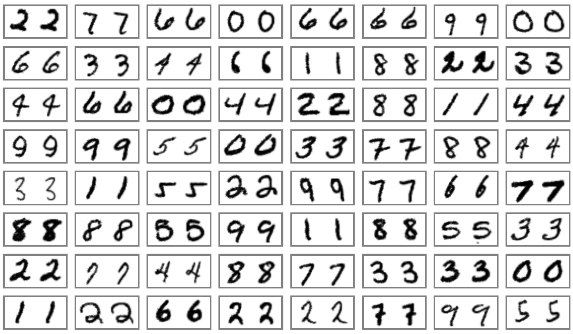
We introduce two new rendering environments. For MNIST, OMNIGLOT and CELEBA generation we use an open-source painting library `libmypaint` (libmypaint contributors, 2018). The agent controls a brush and produces a sequence of (possibly disjoint) strokes on a canvas C. The state of the environment is comprised of the contents of C as well as the current brush location l_t. Each action a_t is a tuple of 8 discrete decisions $(a_t^1, a_t^2, \ldots, a_t^8)$ (see Figure 6.3). The first two components are the control point p_c and the end point l_{t+1} of the stroke, which is specified as a quadratic Bézier curve:

$$p(\tau) = (1-\tau)^2\, l_t + 2(1-\tau)\,\tau\, p_c + \tau^2\, l_{t+1}\,, \tag{6.4.1}$$

where $\tau \in [0, 1]$. In our experiments, we define the valid range of locations as a 32×32 grid imposed on C. We set l_0 to the upper left corner of the canvas. The next 5 components represent the appearance of the stroke: the pressure that the agent applies to the brush (10 levels), the brush size, and the stroke color characterized by mixture of red, green and blue (20 bins for each color component). The last element of a_t is a binary flag specifying the type of action: the agent can choose either to produce a stroke or to jump right to l_{t+1}. For grayscale datasets (MNIST and OMNIGLOT), we omit the color components.

In the MUJOCO SCENES experiment, we render images using a `MuJoCo`-based environment (Todorov et al., 2012). At each time step, the agent has to decide on the object type (4 options), its location on a 16×16 grid, its size (3 options) and the color (3 color components with 4 bins each). The resulting tuple is sent to the environment, which adds an object to

(A) MNIST unconditional generation



(B) MNIST reconstruction

FIGURE 6.4. **MNIST. (A)** A SPIRAL agent is trained to draw MNIST digits via a sequence of strokes in the `libmypaint` environment. As training progresses, the quality of the generations increases. The final samples capture the multi-modality of the dataset, varying brush sizes and digit styles. **(B)** A conditional SPIRAL agent is trained to reconstruct using the same action space. Reconstructions (left) match ground-truth (right) accurately.

the scene according to the specification. Additionally, the agent can decide to skip a move or change the most recently emitted object. All three types of actions are illustrated in Figure 6.2a.

6.4.3. MNIST

For the MNIST dataset, we conduct two sets of experiments. In the first set, we train an unconditional agent to model the data distribution. Along with the reward provided by the discriminator we also use auxiliary penalties expressing our inductive biases for the particular type of data. To encourage the agent to draw a digit in a single continuous motion of the brush, we provide a small negative reward for starting each continuous sequence of strokes. We also found it beneficial to penalize our model for not producing any visible strokes at all. The resulting agent manages to generate samples clearly recognizable as hand-written digits. Examples of such generations are shown in Figure 6.4a.

In the second set of experiments, we train an agent to generate the strokes for a given target digit, and we compare two kinds of rewards discussed in Section 6.3.3: fixed ℓ^2-distance and the discriminator score. The results are summarized in Figure 6.9a (blue curves). We note that the discriminator-based approach significantly speeds up training of the model and achieves lower final ℓ^2 error. When no auxiliary rewards were employed, ℓ^2-based runs failed to learn reasonable reconstructions. Figure 6.4b presents several conditional generations produced by our method.

(A) Omniglot unconditional generation

(B) Omniglot reconstruction

FIGURE 6.5. **Omniglot. (A)** A SPIRAL agent is trained to draw Omniglot characters via a sequence of strokes in the `libmypaint` environment. As training progresses, the quality of the generations increase. The final samples capture the multi-modality of the dataset, varying brush sizes and character styles. **(B)** A conditional SPIRAL agent is trained to reconstruct using the same action space. Reconstructions (left) match ground-truth (right) accurately.

Following Sharma et al. (2017), we also train a "blind" version of the agent, *i.e.*, we do not feed intermediate canvas states as an input to π. That means that the model cannot rely on reactive behaviour since it does not "see" the immediate consequences of its decisions. The training curve for this experiment is shown in Figure 6.9a (dotted blue line). Although the agent does not reach the level of performance of the full model, it can still produce sensible reconstructions which suggests that our approach could be used in the more general setting of program synthesis, where access to intermediate states of the execution pipeline is not assumed.

6.4.4. OMNIGLOT

In the previous section, we showed that our approach works reasonably well for handwritten digits. In this series of experiments, we test our agent in a similar but more challenging setting of handwritten characters. The difficulty of the dataset manifests itself in lower quality of unconditional generations (Figure 6.5a). Note that this task appears to be hard for other neural network based approaches as well: models that do produce good samples, such as (Rezende et al., 2016), do not do so in a manner that mimics actual strokes.

The conditional agent, on the other hand, managed to reach convincing quality of reconstructions (Figure 6.5b). Unfortunately, we could not make the ℓ^2-based model work well in this setting (Figure 6.9a; dashed red line). This suggests not only that discriminator rewards

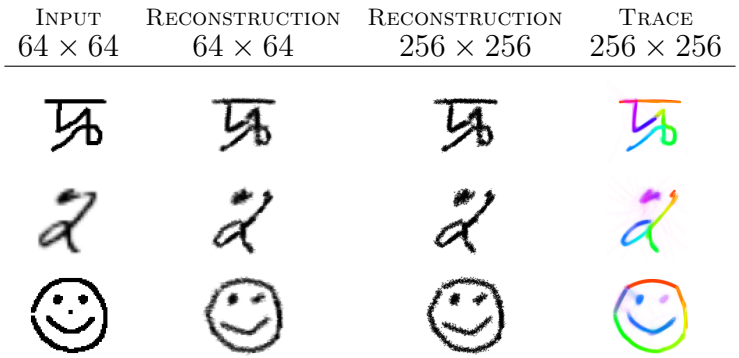


FIGURE 6.6. **Image parsing** using the SPIRAL agent trained on Omniglot. All images from test sets. Given a rastered input (a), the agent produces a reconstruction (b) that closely matches the input. (c) Having access to the underlying strokes, we can render the character at a higher resolution, or in a different stroke style. (d) The agent effectively parses the input image into strokes. Each stroke is depicted in a separate color (we show average across 100 samples).

speed up learning, but also that they allow successful training of agents in cases where naïve rewards like ℓ^2 do not result in sufficient exploration.

Since OMNIGLOT contains a highly diverse set of symbols, over the course of training our model could learn a general notion of image reproduction rather than simply memorizing dataset-specific strokes. In order to test this, we feed a trained agent with previously unseen line drawings. The resulting reconstructions are shown in Figure 6.6. The agent handles out-of-domain images well, although it is slightly better at reconstructing the OMNIGLOT test set.

6.4.5. CELEBA

Since the `libmypaint` environment is in principle, capable of producing complex color paintings, we explore this direction by training a conditional agent on the CELEBA dataset. As in the previous experiments, we use 20-step episodes, and as before, the agent does not receive any intermediate rewards. In addition to the reconstruction reward (either ℓ^2 or discriminator-based), we put a penalty on the earth mover's distance between the color histograms of the model's output and $\mathbf{x}_{\text{target}}$. We found this relatively task-agnostic penalty to slightly improve the performance of the method, but we would like to stress that it is by no means necessary.

Given that we made no effort whatsoever to adapt the action space for this domain, it is not surprising that it takes significantly more time to discover the policy that produces

images resembling the target (Figure 6.7). As in the OMNIGLOT experiment, the ℓ^2-based agent demonstrates some improvement over the random policy but gets stuck and, as a result, fails to learn sensible reconstructions (Figure 6.9b).

Although blurry, the model's reconstruction closely matches the high-level structure of each image. For instance the background color, the position of the face and the color of the person's hair. In some cases, shadows around eyes and the nose are visible. However, we observe that our model tends to generate strokes in the first half of the episode that are fully occluded by strokes in the second half. We hypothesize that this phenomenon is a consequence of credit assignment being quite challenging in this task. One possible remedy is to provide the agent with a mid-episode reward for reproducing a blurred version of the target image. We leave this prospect for future work.

6.4.6. MuJoCo Scenes

For the MuJoCo Scenes dataset, we use our agent to construct simple CAD programs that best explain input images. Here we are only considering the case of conditional generation. Like before, the reward function for the generator can be either the ℓ^2 score or the discriminator output. We did not provide any auxiliary reward signals. The model is unrolled for 20 time steps, so it has the capacity to infer and represent up to 20 objects and their attributes. As we mentioned in Section 6.4.2, the training data consists of scenes with at most 5 objects. The agent does not have this knowledge a priori and needs to learn to place the right number of primitives.

As shown in Figure 6.9b, the agent trained to directly minimize ℓ^2 is unable to solve the task and has significantly higher pixel-wise error. In comparison, the discriminator-based variant solves the task and produces near-perfect reconstructions on a holdout set (Figure 6.8).

We note that our agent has to deal with a high-cardinality action space intractable for a brute-force search. Indeed, the total number of possible execution traces is M^N, where $M = 4 \cdot 16^2 \cdot 3 \cdot 4^3 \cdot 3$ is the total number of attribute settings for a single object (see Section 6.4.2 for details) and $N = 20$ is the length of an episode.[2] In order to demonstrate the computational hardness of the task, we ran a general-purpose *Metropolis-Hastings* inference algorithm on a set of 100 images. The algorithm samples an execution trace defining attributes for a maximum of 20 primitives. These attributes are treated as latent variables. During each time step of inference, a block of attributes (including the presence/absence flag) corresponding to a single object is flipped uniformly within appropriate ranges. The resulting trace is

[2]The actual number of scene configurations is smaller but still intractable.

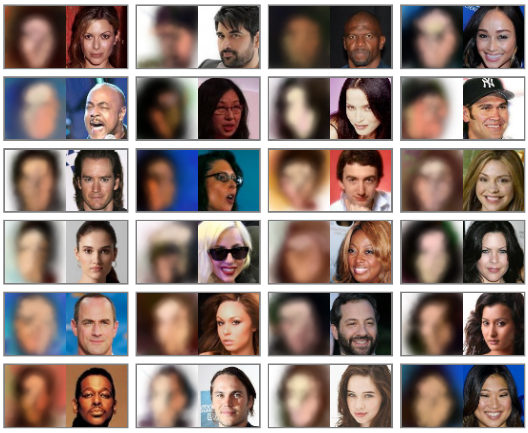


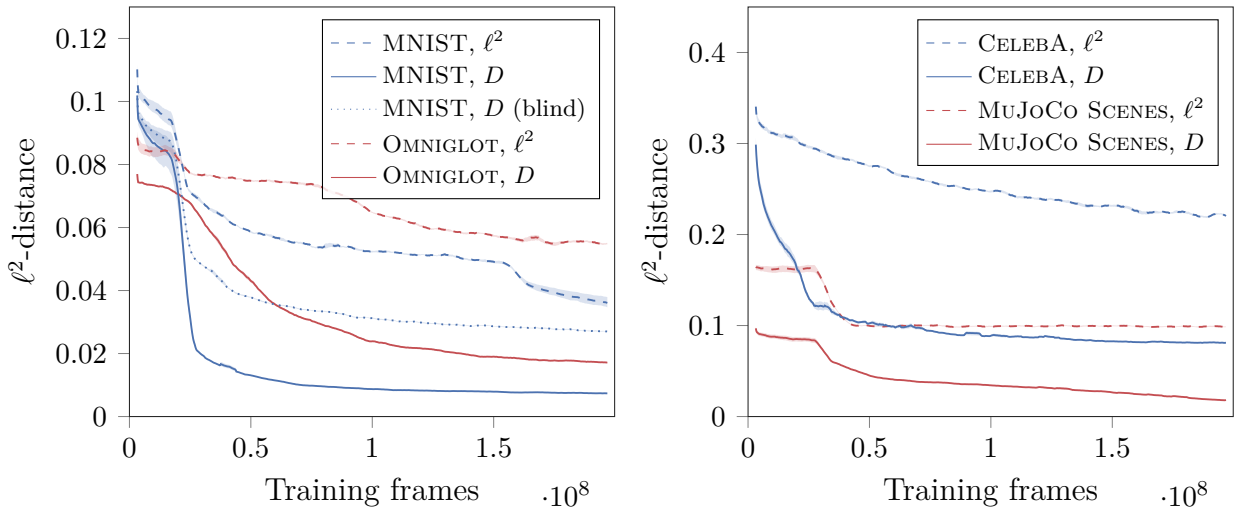
FIGURE 6.7. **CELEBA RECONSTRUCTIONS.** The SPIRAL agent reconstructs human faces in 20 strokes. Although blurry, the reconstructions closely match the high-level structure of each image, for instance the background color, the position of the face and the color of the person's hair. In some cases, shadows around eyes and the nose are visible.

FIGURE 6.8. **3D SCENE RECONSTRUCTIONS.** The SPIRAL agent is trained to reconstruct 3D scenes by emitting sequences of commands for the `MuJoCo` environment. In each pair, the left image corresponds to the model's output while the right one is the target. Our method is capable of acurately inferring the number of objects, their locations, sizes and colors.

rendered by the environement into an output sample which is then accepted or rejected using the Metropolis-Hastings update rule, with a Gaussian likelihood centered around the test image and a fixed diagonal covariance of 0.25. As shown in Figure 6.10, the MCMC search baseline was unable to solve the task even after a large number of evaluations.

6.5. DISCUSSION

Scaling visual program synthesis to real world and combinatorial datasets has been a challenge. We have shown that it is possible to train an adversarial generative agent employing black-box rendering simulators. Our results indicate that using the Wasserstein discriminator's output as a reward function with asynchronous reinforcement learning can provide a scaling path for visual program synthesis. The current exploration strategy used in the agent is entropy-based, but future work should address this limitation by employing sophisticated search algorithms for policy improvement. For instance, Monte Carlo Tree Search can be used, analogous to AlphaGo Zero Silver et al. (2017). General-purpose inference algorithms could also be used for this purpose.



(A) MNIST and Omniglot (B) CelebA and MuJoCo

FIGURE 6.9. **ℓ^2-distance between reconstructions and ground truth** images over the course of training. Across all datasets, we observe that training using a discriminator leads to significantly lower ℓ^2-distances, than when directly minimizing ℓ^2. We also show in (A) that the SPIRAL agent is capable of reconstructing even when it does not have access to the renderer in intermediate steps, however this does lead to a small degradation in performance.

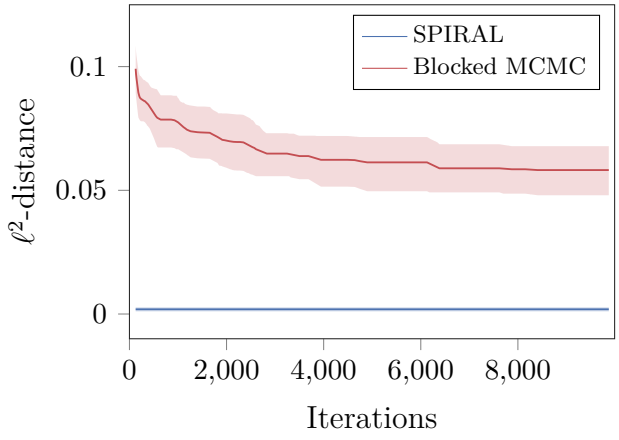


FIGURE 6.10. **Blocked Metropolis-Hastings (MCMC) *vs* SPIRAL.** The MuJoCo Scenes dataset has a large combinatorial search space. We ran a general-purpose MCMC algorithm with object based blocked proposals and SPIRAL on 100 holdout images during inference time. SPIRAL reliably processes every image in a single pass. We ran the MCMC algorithm for thousands of evaluations but it was unable to solve the task.

Future work should explore different parameterizations of action spaces. For instance, the use of two arbitrary control points is perhaps not the best way to represent strokes, as it is hard to deal with straight lines. Actions could also directly parametrize 3D surfaces, planes and learned texture models to invert richer visual scenes. On the reward side, using a joint image-action discriminator similar to BiGAN/ALI (Donahue et al., 2016; Dumoulin et al., 2016) (in this case, the policy can viewed as an encoder, while the renderer becomes a decoder) could result in a more meaningful learning signal, since D will be forced to focus on the semantics of the image.

We hope that this paper provides an avenue to further explore inverse simulation and program synthesis on applications ranging from vision, graphics, speech, music and scientific simulators.

Chapter 7

CONCLUSIONS AND FUTURE WORK

In this thesis, we have proposed several general-purpose methods for solving image processing and synthesis tasks. Although the problems we considered represent different (and sometimes quite distinct) facets of the computer vision field, they are united by the fact that until recently all of them have primarily been tackled by complicated hand-engineered pipelines. We have contributed more evidence that a conceptually simpler alternative, deep learning, can both reduce the amount of labor needed to design a CV system and, even more importantly, significantly improve its performance. Our findings thus verify the superiority of automatically learned distributed representations.

More concretely, in Chapter 3, we showed that several related early-vision tasks such as edge detection and thin object segmentation can be solved within the same data-agnostic framework based on convolutional neural networks. At the heart of the method lies a combination of discriminative descriptor learning and subsequent nearest-neighbor search in the obtained descriptor space. The former eliminates the need for careful design of input features typically required by non-deep systems. The latter serves two purposes: it simultaneously compensates for the effect of underfitting of the CNN and also ensures coherent outputs. The resulting model can be loosely seen as a variant of an existing state-of-the-art edge detector (Dollár and Zitnick, 2013), albeit fully data-driven and therefore much more flexible.

Chapter 4 considered a different image transformation setting – gaze correction. Here, unlike in edge detection, both the input and the desired output come from the same domain, *i.e.*, images of rectangular eye regions. We proposed a system that exploited that fact by making an assumption that the bulk of the transformation can be carried out by warping the input. Thanks to the differentiability of binlinear sampling, the warping procedure can be injected directly into a fully-convolutional network trained on matching pairs of images. Compared to conventional NN-based image-to-image models, one of the advantages of the proposed approach is that it allows the network to copy a lot of visual information directly

from the input and, as a result, the modelling capacity is concentrated on the transformation itself. In a series of experiments, we demonstrated that our end-to-end system achieves an impressive level of photorealism and outperforms existing methods employing more traditional CV techniques.

In Chapter 5, we made a detour and presented a simple yet effective domain adaptation technique applicable not only in the context of computer vision but in virtually any deep learning setting where it is important to address the effect of covariate shift. One example of such a setting would be training a CNN on synthetic data for subsequent deployment in a real environment. We should expect a severe performance degradation due to the mismatch between the training and the test domains. Developed concurrently with and independently of GANs (Goodfellow et al., 2014), our method, DANN, uses an adversarial objective to promote the emergence of domain-invariant features thus making the underlying network robust to the covariate shift. We showed that despite the triviality of implementation (just a few lines of code), DANN achieves excellent results on a number of standard benchmarks beating both deep and non-deep baselines.

Recall that in our gaze correction project, we made an observation that the appearance of the output is very close to the appearance of the input and therefore, it made sense to target the system at modelling what is different. In Chapter 6, we used a similar principle to build a generative model for visual data. Instead of operating at the pixel-level, we proposed to control existing off-the-shelf simulators that already encode a lot of relevant prior knowledge about the image structure. To that end, we developed a GAN-like architecture in which the generator is comprised of a learnable policy network and a fixed external renderer receiving commands from that policy. Our model trained using large-scale RL techniques is free of any domain-specific engineering but nevertheless demonstrates impressive results on the data notoriously hard for conventional inverse graphics systems.

We believe that the techniques we presented in this thesis are interesting, useful, and will serve as a stepping stone for further advances in the field of computer vision and beyond. To conclude, we would like to highlight the key findings[1] of the research we have conducted so far:

- The rapid growth of the amount of available data is gradually rendering hand-engineered features absolete. If one aims to build a robust and flexible computer vision system, they should consider using appropriate representations learned directly from data.

[1]Note that some of the articles included in the present thesis date back to 2013 – 2014. At the time, end-to-end neural networks still were not widely accepted as a go-to backbone for CV systems and, in particular, for edge detection and gaze correction algorithms.

- A preferred way of obtaining data-driven features is through the end-to-end training of deep models. Unlike traditional stacking of disjoint processing blocks, this approach directly optimizes all the components for the ultimate objective we care about and thus results in better performance.
- In some scenarios, however, it is beneficial to step away from generic convolutional models and inject more prior knowledge about the task by incorporating special computational modules (*e.g.*, warping layers) into the differentiable pipeline. The large body of pre-neural CV literature might serve as good source of ideas for such hybrid systems.
- The right choice of the loss function is no less important. In many cases (*e.g.*, for domain adaptation or inverse graphics), this choice is not trivial. Poor design of the objective may lead to a less-than-impressive final system despite its theoretical capabilities. The adversarial framework provides an exciting new way to construct an adaptive loss that automatically focuses on the relevant aspects of the data and the model being trained.

Future Work

Below, we will briefly describe several ideas for the future research efforts that build upon the work we have discussed in the thesis.

The **first** one is related to domain adaptation for semantic segmentation (synthetic $\rightarrow$ real). A common observation in the recent adversarial DA literature is that the performance on the task of interest can be significantly improved if one imposes additional constraints on the structure of the network. Following this idea, instead of adapting some internal representation of a conventional segmentation network, we propose to conduct segmentation in a special domain-invariant space $\mathcal{Z}$. Just like input images, the points from that space have spatial dimensions but potentially more feature channels. We assume that the mapping E from the image to the domain-invariant representation is conservative and preserves the bulk of information about the scene. In order to enforce this property, we could use an additional decoder D to map back to the image space and ensure that reconstructions are good enough. Instead of using two separate decoders for the source and the target data, we could employ a single conditional model (*e.g.*, via conditional batch normalization (Dumoulin et al., 2017)).

Having a conditional D also provides an opportunity to perform domain transfer in the visible space. For transferred images, we need to verify that they get encoded (by E) to the

same domain-invariant representation as their original counterparts. This gives the second set of objectives.

The next objective is simply a segmentation loss computed for some segmentation network S operating on the $\mathcal{Z}$-space.

One of the problems with using the existing image-to-image systems (like `CycleGAN` (Zhu et al., 2017)) in conjunction with segmentation is that unconditional discrimination often results in change of semantics in the transformed image due to the mismatch of marginal distributions of labels in the source and in the target. To mitigate this issue in our model, we propose to adapt $\mathcal{Z}$ by using a discriminator conditioned on the outputs of S. This modification allows us to have different distributions of object types in different domains while maintaining domain-invariance within each type.

The full model is fully-differentiable and can be trained with any variant of stochastic gradient descent.

The **second** idea explores the possibility of using image-to-image translation as a foundation for unconditional generative models. As we discussed in Chapter 4, due to limited modeling power, architectures based on conventional neural decoders tend to discard some amount of fine details present in the data (*i.e.*, samples have simplified appearence and are easily recognizable as artificially synthesized). On the other hand, we saw that in the case of image-to-image translation, one can adopt much of the input information almost for free thus achieving better utilization of the network's capacity. This suggests that posing the image generation task as a direct traversal in the visible space (as opposed to a low-dimensional latent space) might be beneficial.

We therefore propose to represent an image manifold as a collection of tangent planes[2]. In a particular tangent plane placed at some image I, each direction $\mathbf{a}$ corresponds to a transformation of that image. Similarly to DeepWarp (Ganin et al., 2016a), the core of the method is a network T that takes I and $\mathbf{a}$ and performs a "hop" on the manifold landing at I'. The sampling is then conducted as a series of stochastic (in $\mathbf{a}$) "hops" starting at some seed image from the training set.

One can learn T by recovering geodesics between arbitrary pairs of training examples. More concretely, we propose to use an additional encoder network E that estimates the "hop" direction by looking at the current image and the desired target image I_{target}. Optimal E and T minimize the dissimilarity between the final output and I_{target} as well as some notion of distance covered by the model in an attempt to reach I_{target} (*e.g.*, the number of "hops" or a learned cost that depends on the performed transformations). We might also want to

[2]A similar albeit non-"deep" approach is explored in (Bengio and Monperrus, 2005; Bengio et al., 2006).

ensure that each intermediate point produced by T lies on the manifold. This can be achieved with adversarial training. Finally, since we assume that the training set is representative of the manifold, for any image that we encounter as we run a chain of transformations, all directions $\mathbf{a}$ should be (marginally) equally probable. That gives one more term for the training objective.

We realize that the idea we have just described may be too vague and potentially flawed. Regardless, we believe that fusing deep learning techniques with principles of differential geometry is an exciting and promising direction for generative modeling research.

Bibliography

Pattern Recognition and Machine Learning. 2006.

David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. In *Readings in Computer Vision*. Elsevier, 1987.

Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *TPAMI*, 2011.

Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

Artem Babenko, Anton Slesarev, Alexander Chigorin, and Victor S. Lempitsky. Neural codes for image retrieval. In *ECCV*, pages 584–599, 2014.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Mahsa Baktashmotlagh, Mehrtash Tafazzoli Harandi, Brian C. Lovell, and Mathieu Salzmann. Unsupervised domain adaptation by domain invariant projection. In *ICCV*, pages 769–776, 2013.

Aayush Bansal, Yaser Sheikh, and Deva Ramanan. Pixelnn: Example-based image synthesis. *arXiv preprint arXiv:1708.05349*, 2017.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. *ECCV*, 2006.

Carlos J. Becker, Roberto Rigamonti, Vincent Lepetit, and Pascal Fua. Supervised feature learning for curvilinear structure segmentation. In *MICCAI*, volume 8149 of *Lecture Notes in Computer Science*, pages 526–533, 2013.

Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In *NIPS*, pages 137–144, 2006.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *JMLR*, 2010.

Yoshua Bengio and Martin Monperrus. Non-local manifold tangent learning. In *NIPS*, 2005.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 1994.

Yoshua Bengio, Martin Monperrus, and Hugo Larochelle. Nonlocal estimation of manifold structure. *Neural Computation*, 2006.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, 2007.

Gedas Bertasius, Jianbo Shi, and Lorenzo Torresani. DeepEdge: A multi-scale bifurcated deep network for top-down contour detection. In *CVPR*, 2015.

Alex Beutel, Jilin Chen, Zhe Zhao, and Ed H Chi. Data decisions and theoretical implications when adversarially learning fair representations. *arXiv preprint arXiv:1707.00075*, 2017.

John Blitzer, Ryan T. McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Conference on Empirical Methods in Natural Language Processing*, pages 120–128, 2006.

John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman. Learning bounds for domain adaptation. In *Advances in neural information processing systems*, 2008.

Karsten M. Borgwardt, Arthur Gretton, Malte J. Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alexander J. Smola. Integrating structured biological data by kernel maximum mean discrepancy. In *ISMB*, pages 49–57, 2006.

Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. In *NIPS*, 2016.

Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. In *CVPR*, 2017.

Lorenzo Bruzzone and Mattia Marconcini. Domain adaptation problems: A DASVM classification technique and a circular validation strategy. *Transaction Pattern Analysis and Machine Intelligence*, 2010.

Harold Christopher Burger, Christian J Schuler, and Stefan Harmeling. Image denoising: Can plain neural networks compete with bm3d? In *CVPR*, pages 2392–2399, 2012.

Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. Real-time video super-resolution with spatio-temporal networks and motion compensation. In *CVPR*, 2017.

Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *ECCV*, 2010.

John Canny. A computational approach to edge detection. *TPAMI*, 1986.

Olivier Chapelle and Alexander Zien. Semi-supervised classification by low density separation. In *AISTATS*, 2005.

Ken Chatfield, Victor S Lempitsky, Andrea Vedaldi, and Andrew Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *BMVC*, 2011.

Minmin Chen, Zhixiang Eddie Xu, Kilian Q. Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. In *ICML*, pages 767–774, 2012.

Qiang Chen, Junshi Huang, Rogerio Feris, Lisa M. Brown, Jian Dong, and Shuicheng Yan. Deep domain adaptation for describing people based on fine-grained clothing attributes. June 2015.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

S. Chopra, S. Balakrishnan, and R. Gopalan. Dlid: Deep learning for domain adaptation by interpolating between domains. In *ICML Workshop on Challenges in Representation Learning*, 2013.

Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, volume 1, pages 539–546, 2005.

Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CVPR*, 2012.

Dan C. Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In *NIPS*, 2012.

Corinna Cortes and Mehryar Mohri. Domain adaptation and sample bias correction theory and algorithm for regression. *Theor. Comput. Sci.*, 2014.

Antonio Criminisi, Jamie Shotton, Andrew Blake, and Philip HS Torr. Gaze manipulation for one-to-one teleconferencing. In *ICCV*, 2003.

Antonio Criminisi, Patrick Pérez, and Kentaro Toyama. Region filling and object removal by exemplar-based image inpainting. *Image Processing, IEEE Transactions on*, 2004.

Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image restoration by sparse 3d transform-domain collaborative filtering. In *Electronic Imaging 2008*, pages 681207–681207. International Society for Optics and Photonics, 2008.

Andrew Dai, Balaji Lakshminarayanan, Ian Goodfellow, Liam Fedus, Mihaela Rosca, and Shakir Mohamed. Gradient penalties for generative adversarial networks. 2018.

Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, 2015.

Piotr Dollár and C. Lawrence Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, 2014.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. In *ICLR*, 2017.

Michael Egmont-Petersen, Dick de Ridder, and Heinz Handels. Image processing with neural networks —- a review. *Pattern Recognition*, 2002.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B Tenenbaum. Learning to infer graphics programs from hand-drawn images. *arXiv preprint arXiv:1707.09627*, 2017.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 1990.

SM Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E Hinton. Attend, infer, repeat: Fast scene understanding with generative models. In *NIPS*, 2016.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results, 2007.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 2008.

Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *TPAMI*, 2013.

Basura Fernando, Amaury Habrard, Marc Sebban, and Tinne Tuytelaars. Unsupervised visual domain adaptation using subspace alignment. In *ICCV*, 2013.

Clément Feutry, Pablo Piantanida, Yoshua Bengio, and Pierre Duhamel. Learning anonymized representations with adversarial neural networks. *arXiv preprint arXiv:1802.09386*, 2018.

William T. Freeman, Egon C. Pasztor, and Owen T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 2000.

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*. 1982.

Yaroslav Ganin. Deepwarp for facial expression manipulation. https://github.com/ddtm/deep-smile-warp, 2017.

Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *ICML*, 2015.

Yaroslav Ganin, Daniil Kononenko, Diana Sungatullina, and Victor Lempitsky. DeepWarp: Photorealistic image resynthesis for gaze manipulation. In *ECCV*, 2016a.

Yaroslav Ganin, Daniil Kononenko, Diana Sungatullina, and Victor Lempitsky. Deepwarp project page. http://sites.skoltech.ru/compvision/projects/deepwarp/, 2016b.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *JMLR*, 2016c.

Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *NIPS*, 2015.

Pascal Germain, Amaury Habrard, François Laviolette, and Emilie Morvant. A PAC-Bayesian approach for domain adaptation with specialization to linear classifiers. In *ICML*, pages 738–746, 2013.

Amir Ghodrati, Xu Jia, Marco Pedersoli, and Tinne Tuytelaars. Towards automatic image editing: Learning to see another you. *arXiv preprint arXiv:1511.08446*, 2015.

Dominik Giger, Jean-Charles Bazin, Claudia Kuster, Tiberiu Popa, and Markus Gross. Gaze correction with a single webcam. In *IEEE International Conference on Multimedia & Expo*, 2014.

Ross Girshick. Fast R-CNN. In *ICCV*, 2015.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML*, pages 513–520, 2011a.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011b.

Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *CVPR*, pages 2066–2073, 2012.

Boqing Gong, Kristen Grauman, and Fei Sha. Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation. In *ICML*, pages 222–230, 2013.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *ICCV*, pages 999–1006, 2011.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2017.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. In *ICML*, 2015.

Ralph Gross, Iain Matthews, Jeffrey Cohn, Takeo Kanade, and Simon Baker. Multi-pie. *Image and Vision Computing*, 2010.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *NIPS*, 2017.

David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning. COURSERA, 2012a.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, 2012b.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NIPS*, 2016.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Judy Hoffman, Eric Tzeng, Jeff Donahue, Yangqing Jia, Kate Saenko, and Trevor Darrell. One-shot adaptation of supervised deep convolutional models. *CoRR*, 2013.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 1989.

Xiaodi Hou, Alan Yuille, and Christof Koch. Boundary detection benchmarking: Beyond F-measures. In *CVPR*, 2013.

Fei Huang and Alexander Yates. Biased representation learning for domain adaptation. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1313–1323, 2012.

Jiayuan Huang, Alexander J. Smola, Arthur Gretton, Karsten M. Borgwardt, and Bernhard Schölkopf. Correcting sample selection bias by unlabeled data. In *NIPS*, pages 601–608, 2006.

David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 1959.

David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 1962.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H. Adelson. Crisp boundary detection using pointwise mutual information. In *ECCV*, 2014.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, 2017.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Viren Jain and H. Sebastian Seung. Natural image denoising with convolutional networks. In *NIPS*, pages 769–776, 2008.

Viren Jain, Joseph F. Murray, Fabian Roth, Srinivas C. Turaga, Valentin P. Zhigulin, Kevin L. Briggman, Moritz Helmstaedter, Winfried Denk, and H. Sebastian Seung. Supervised learning of image restoration with convolutional networks. In *ICCV*, 2007.

Varun Jampani, Sebastian Nowozin, Matthew Loper, and Peter V Gehler. The informed sampler: A discriminative approach to bayesian inference in generative computer vision models. *CVIU*, 2015.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, 2014.

J. Jiang. A literature survey on domain adaptation of statistical classifiers. Technical report, CS Department at Univ. of Illinois at Urbana-Champaign, 2008.

Andrew Jones, Magnus Lang, Graham Fyffe, Xueming Yu, Jay Busch, Ian McDowall, Mark T. Bolas, and Paul E. Debevec. Achieving eye contact in a one-to-many 3D video teleconferencing system. *ACM Trans. Graph.*, 2009.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, 2015.

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *arXiv preprint arXiv:1507.01526*, 2015.

Yan Ke and Rahul Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *CVPR*. IEEE, 2004.

Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Very Large Data Bases*, pages 180–191, 2004.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Jyri J. Kivinen, Christopher K. I. Williams, and Nicolas Heess. Visual boundary prediction: A deep neural prediction network and quality dissection. In *AISTATS*, pages 512–521, 2014.

Chris L Kleinke. Gaze and eye contact: a research review. *Psychological bulletin*, 1986.

Naveen Kodali, James Hays, Jacob Abernethy, and Zsolt Kira. On convergence and stability of gans. 2018.

Iasonas Kokkinos. Pushing the boundaries of boundary detection using deep learning. In *ICLR*, 2016.

Daniil Kononenko and Victor Lempitsky. Learning to look up: realtime monocular gaze correction using machine learning. In *CVPR*, 2015.

Daniil Kononenko, Yaroslav Ganin, Diana Sungatullina, and Victor Lempitsky. Photorealistic monocular gaze redirection using machine learning. *TPAMI*, 2017.

Anton Konushin. Introduction to computer vision and deep learning. YouTube Lectures, 2017.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012.

Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *CVPR*, 2015a.

Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015b.

Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *NIPS*, 2015c.

Claudia Kuster, Tiberiu Popa, Jean-Charles Bazin, Craig Gotsman, and Markus Gross. Gaze correction for home video conferencing. In *SIGGRAPH Asia*, 2012.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 2015.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 2017.

Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *NIPS*, 2016.

Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In *NIPS*, pages 396–404, 1989a.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, 1989b.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*. 2012.

Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *ICCV*, 2011.

Yujia Li, Kevin Swersky, and Richard Zemel. Unsupervised domain adaptation by domain invariant projection. In *NIPS 2014 Workshop on Transfer and Multitask Learning*, 2014.

libmypaint contributors. libmypaint. https://github.com/mypaint/libmypaint, 2018.

Joseph J Lim, C Lawrence Zitnick, and Piotr Dollár. Sketch tokens: A learned mid-level representation for contour and object detection. In *CVPR*, 2013.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *ICCV*, 2015.

Ziwei Liu, Raymond Yeh, Xiaoou Tang, Yiming Liu, and Aseem Agarwala. Video frame synthesis using deep voxel flow. In *ICCV*, 2017.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

Mingsheng Long and Jianmin Wang. Learning transferable features with deep adaptation networks. *CoRR*, 2015.

Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *ECCV*, 2014.

David G Lowe. Object recognition from local scale-invariant features. In *ICCV*, 1999.

David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *CVPR*, 2015.

Vikash K Mansinghka, Tejas D Kulkarni, Yura N Perov, and Josh Tenenbaum. Approximate bayesian image interpretation using generative probabilistic graphics programs. In *NIPS*, 2013.

Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation: Learning bounds and algorithms. In *COLT*, 2009a.

Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Multiple source adaptation and the rényi divergence. In *UAI*, pages 367–374, 2009b.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 1943.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

Marvin Minsky and Seymour Papert. Perceptrons. 1969.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

Tom M Mitchell et al. Machine learning, 1997.

Volodymyr Mnih and Geoffrey E Hinton. Learning to detect roads in high-resolution aerial images. In *ECCV*, pages 210–223. Springer, 2010.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Vinod Nair and Geoffrey E Hinton. Inferring motor programs from images of handwritten digits. In *NIPS*, 2006.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Vinod Nair, Josh Susskind, and Geoffrey E Hinton. Analysis-by-synthesis by learning to invert generative black boxes. In *ICANN*, 2008.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

Ken-Ichi Okada, Fumihiko Maeda, Yusuke Ichikawaa, and Yutaka Matsushita. Multiparty videoconferencing at virtual social distance: Majic design. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 385–393, 1994.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *ICML*, 2016.

M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, 2014.

Maxime Oquab. Torch7 modules for spatial transformer networks. https://github.com/qassemoquab/stnbhwd, 2015.

Sinno Jialin Pan, Ivor W. Tsang, James T. Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 2011.

Seymour A Papert. The summer vision project. 1966.

Eunbyung Park, Jimei Yang, Ersin Yumer, Duygu Ceylan, and Alexander C Berg. Transformation-grounded image generation network for novel 3d view synthesis. *arXiv preprint arXiv:1703.02921*, 2017.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3d face model for pose and illumination invariant face recognition. In *AVSS*, 2009.

Florent Perronnin and Christopher Dance. Fisher kernels on visual vocabularies for image categorization. In *CVPR*, 2007.

Lizzie Plaugic. The article on the verge. https://www.theverge.com/2017/3/11/14885986/deepwarp-neural-networks-eye-rolling-keanu-reeves, 2017.

Sanjay Purushotham, Wilka Carvalho, Tanachat Nilanon, and Yan Liu. Variational recurrent adversarial deep domain adaptation. In *ICLR*, 2017.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *ICML*, 2009.

Marc'Aurelio Ranzato and Geoffrey E. Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *CVPR*, pages 2551–2558, 2010.

Scott Reed, Aäron Oord, Nal Kalchbrenner, Sergio Gómez Colmenarejo, Ziyu Wang, Yutian Chen, Dan Belov, and Nando Freitas. Parallel multiscale autoregressive density estimation. In *ICML*, 2017.

Scott E Reed, Yi Zhang, Yuting Zhang, and Honglak Lee. Deep visual analogy-making. In *NIPS*, 2015.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

Danilo Rezende, Ivo Danihelka, Karol Gregor, Daan Wierstra, et al. One-shot generalization in deep generative models. In *ICML*, 2016.

Lawrence Gilman Roberts. *Machine perception of three-dimensional soups*. PhD thesis, Massachusetts Institute of Technology, 1963.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 1958.

Kevin Roth, Aurelien Lucchi, Sebastian Nowozin, and Thomas Hofmann. Stabilizing training of generative adversarial networks through regularization. In *NIPS*, 2017.

Artem Rozantsev, Mathieu Salzmann, and Pascal Fua. Beyond sharing weights for deep domain adaptation. *arXiv preprint arXiv:1603.06432*, 2016.

David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al. A general framework for parallel distributed processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1986.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 1988.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In *ECCV*, pages 213–226. 2010.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 1959.

Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Tim Lillicrap. A simple neural network module for relational reasoning. In *NIPS*, 2017.

Jürgen Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 1992.

John Schulman. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, 2016.

Hannes Schulz and Sven Behnke. Learning object-class segmentation with convolutional neural networks. In *ESANN*, volume 3, 2012.

Dmitriy Serdyuk, Kartik Audhkhasi, Philémon Brakel, Bhuvana Ramabhadran, Samuel Thomas, and Yoshua Bengio. Invariant representations for noisy speech recognition. *arXiv preprint arXiv:1612.01928*, 2016.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, 2013.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. CSGNet: Neural shape parser for constructive solid geometry. *arXiv preprint arXiv:1712.08290*, 2017.

Wei Shen, Xinggang Wang, Yan Wang, Xiang Bai, and Zhijiang Zhang. DeepContour: A deep convolutional feature learned by positive-sharing loss for contour detection. In *CVPR*, 2015.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 2000.

Rui Shu, Hung H Bui, Hirokazu Narui, and Stefano Ermon. A dirt-t approach to unsupervised domain adaptation. *arXiv preprint arXiv:1802.08735*, 2018.

Nathan Silberman and Rob Fergus. Indoor scene segmentation using a structured light sensor. In *ICCV Workshops*, pages 601–608. IEEE, 2011.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of*

Machine Learning Research, 2014.

Joes Staal, Michael D. Abràmoff, Meindert Niemeijer, Max A. Viergever, and Bram van Ginneken. Ridge-based vessel segmentation in color images of the retina. *IEEE Trans. Med. Imaging*, 2004.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. 2016.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 2000.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. In *CVPR*, 2015.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 2012.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

Carlo Tomasi. Early vision. In *Encyclopedia of Cognitive Science*. 2006.

Hsiao-Yu Fish Tung, Adam W Harley, William Seto, and Katerina Fragkiadaki. Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision. In *ICCV*, 2017.

Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *CoRR*, 2014.

Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *ICCV*, 2015.

Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *CVPR*, 2017.

Branislav Ulicny. Alteredqualia's demo. http://alteredqualia.com/xg/examples/eyes_gaze3.html, 2017.

Joost van Amersfoort, Wenzhe Shi, Alejandro Acosta, Francisco Massa, Johannes Totz, Zehan Wang, and Jose Caballero. Frame interpolation with multi-scale deep loss functions and generative adversarial networks. *arXiv preprint arXiv:1711.06045*, 2017.

Laurens van der Maaten. Barnes-Hut-SNE. *CoRR*, 2013.

Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of Complexity*, pages 11–30. 2015.

A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *TPAMI*, 2012.

Cédric Villani. *Optimal transport: old and new*. Springer Science & Business Media, 2008.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, pages 1096–1103, 2008.

Lisa J Wallis, Friederike Range, Corsin A Müller, Samuel Serisier, Ludwig Huber, and Zsófia Virányi. Training for eye contact modulates gaze following in dogs. *Animal behaviour*, 2015.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1988.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*. Springer, 1992.

Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 1990.

Lior Wolf, Ziv Freund, and Shai Avidan. An eye for an eye: A single camera gaze-replacement method. In *CVPR*, 2010.

Erroll Wood, Tadas Baltrusaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. Gazedirector: Fully articulated eye gaze redirection in video. *arXiv preprint arXiv:1704.08763*, 2017.

Jiajun Wu, Erika Lu, Pushmeet Kohli, Bill Freeman, and Josh Tenenbaum. Learning to see physics via visual de-animation. In *NIPS*, 2017a.

Jiajun Wu, Joshua B Tenenbaum, and Pushmeet Kohli. Neural scene de-rendering. In *CVPR*, 2017b.

Ren Xiaofeng and Liefeng Bo. Discriminatively trained sparse code gradients for contour detection. In *NIPS*. 2012.

Ning Xie, Hirotaka Hachiya, and Masashi Sugiyama. Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. *IEICE Transactions on Information and Systems*, 2013.

Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *CVPR*, 2015.

Xuehan Xiong and Fernando Torre. Supervised descent method and its applications to face alignment. In *CVPR*, 2013.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

Ruigang Yang and Zhengyou Zhang. Eye gaze correction with stereovision for video-teleconferencing. In *ECCV*, 2002.

Raymond Yeh, Ziwei Liu, Dan B Goldman, and Aseem Agarwala. Semantic facial expression editing using autoencoded flow. *arXiv preprint arXiv:1611.09961*, 2016.

B Yip and J. S. Jin. Face re-orientation using ellipsoid model in video conference. In *Proc. 7th IASTED International Conference on Internet and Multimedia Systems and Applications*, pages 245–250, 2003.

Jason J Yu, Adam W Harley, and Konstantinos G Derpanis. Back to basics: Unsupervised learning of optical flow via brightness constancy and motion smoothness. In *ECCV*, 2016.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Jure Zbontar and Yann LeCun. Stereo matching by training a convolutional neural network to compare image patches. *JMLR*, 2016.

Matthew D. Zeiler. ADADELTA: an adaptive learning rate method, 2012.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*. 2014.

Tinghui Zhou, Shubham Tulsiani, Weilun Sun, Jitendra Malik, and Alexei A Efros. View synthesis by appearance flow. In *ECCV*, 2016.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *CVPR*, 2017.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.

Appendix A

ADDITIONAL DETAILS TO CHAPTER 6

A.1. Optimal D for Conditional Generation

It turns out that in the case of conditional generation (*i.e.*, p_d is a Dirac δ-function), we can derive an explicit form of the optimal (non-parametric) discriminator D. Indeed, (6.3.1) corresponds to the dual representation of the Wasserstein-1 metric (Villani, 2008). The primal form of that metric is defined as

$$W_1(p_g, p_d) = \inf_{\gamma \in \Gamma(p_g, p_d)} \int \|\mathbf{x} - \mathbf{y}\|_2 \, d\gamma(\mathbf{x}, \mathbf{y}) \,, \tag{A.1.1}$$

where $\Gamma(p_g, p_d)$ is a set of all couplings between p_g and p_d. Taking into account that the data distribution is a point mass, we can simplify (A.1.1):

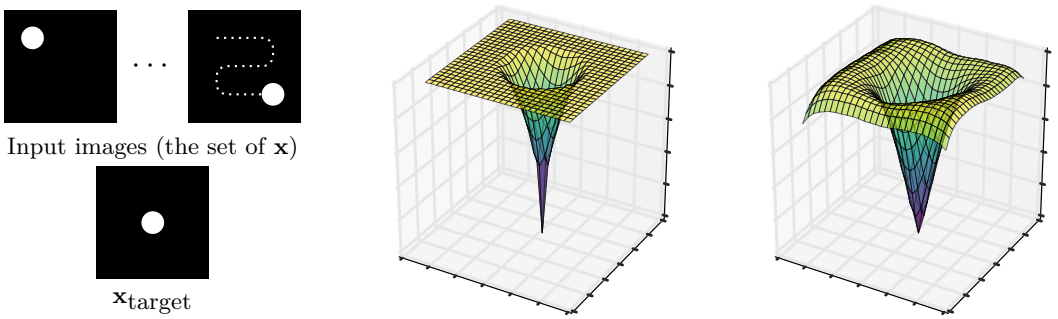
$$W_1(p_g, p_d) = \mathbb{E}_{\mathbf{x} \sim p_g} \|\mathbf{x} - \mathbf{x}_{\text{target}}\|_2 \,. \tag{A.1.2}$$

The expression above gives the optimal value for $\mathcal{L}_D$ in (6.3.1). Therefore $D(\mathbf{x}) = \|\mathbf{x} - \mathbf{x}_{\text{target}}\|_2$ is a solution of (6.3.1):

$$\begin{aligned} \mathcal{L}_D(\|\mathbf{x} - \mathbf{x}_{\text{target}}\|_2) = &- \|\mathbf{x}_{\text{target}} - \mathbf{x}_{\text{target}}\|_2 \\ &+ \mathbb{E}_{\mathbf{x} \sim p_g} \|\mathbf{x} - \mathbf{x}_{\text{target}}\|_2 \\ &+ 0 \,, \end{aligned} \tag{A.1.3}$$

where the last term (R) is zero since the Euclidean distance belongs to the set of 1-Lipschitz functions.

This result suggests that for inverse graphics, in (6.3.4), one may use a fixed image distance (like the Euclidean distance ℓ^2) instead of a parametric function optimized via the WGAN objective. Note, however, that ℓ^2 is not a unique solution to (6.3.1). Consider, for example, the case where the model distribution is also a Dirac delta centered at $\mathbf{x}_{\text{g}}$. The



(A) Data. (B) ℓ^2-distance to $\mathbf{x}_{\text{target}}$. (C) $D(\mathbf{x}, \mathbf{x}_{\text{target}})$.

FIGURE A.1. A toy experiment illustrating the **difference between ℓ^2 and discriminator** training in practice. **(A)** We collect a dataset of images with a single solid circle in all possible locations (top) and pick one of them as a target image (bottom). **(B)** The ℓ_2-distance (in the pixel space) from the input images to the target as a function of the circle location; the surface is flat around the borders since the circles do not overlap. **(C)** We train a discriminative model D that takes a pair of images and tells whether they match or not; just like ℓ^2, the resulting function has a pit centered at the target location, but unlike (b), the surface now has better behaved gradients.

Wasserstein distance is equal to $d = \|\mathbf{x}_{\text{g}} - \mathbf{x}_{\text{target}}\|_2$. In order to achieve that value in (6.3.1), we could take any D such that $\forall\, \alpha \in [0, 1]$

$$D\left(\alpha\, \mathbf{x}_{\text{g}} + (1 - \alpha)\, \mathbf{x}_{\text{target}}\right) = \alpha \cdot d\,, \tag{A.1.4}$$

and $\text{Lip}(D) \leq 1$. One example of such function would be a hyperplane H containing the segment (A.1.4). Let us now consider a set of points

$$V = \left\{\mathbf{x} \,|\, \|\mathbf{x} - \mathbf{x}_{\text{g}}\|_2 < \varepsilon, \|\mathbf{x} - \mathbf{x}_{\text{target}}\|_2 = d\right\} \tag{A.1.5}$$

in an ε-vicinity of $\mathbf{x}_{\text{g}}$. By definition, $\|\mathbf{x} - \mathbf{x}_{\text{target}}\|_2$ is constant for any $\mathbf{x} \in V$. That means that ℓ^2 expresses no preference over points that are equidistant from $\mathbf{x}_{\text{target}}$ even though some of them may be semantically closer to $\mathbf{x}_{\text{target}}$. This property may significantly slow down learning if we are relying on ℓ^2 (or similar distance) as our training signal. Functions like H, on the other hand, have non-zero slope in V and therefore can potentially shift the search towards more promising subspaces.

One other reason why discriminator training is different from using a fixed image distance is that in practice, we do not optimize the exact dual formulation of the Wasserstein distance and, on top of that, use stochastic gradient descent methods which we do not run until convergence. A toy example illustrating that difference is presented in Figure A.1.

A.2. Network Architectures

The policy network (shown in Figure A.2) takes the observation (*i.e.*, the current state of the canvas C_t) and conditions it on a tuple corresponding to the last performed action a_t. The resulting features are then downsampled to a lower-dimensional spatial resolution by means of strided convolutions and passed through a stack of ResNet blocks (He et al., 2016) followed by a fully-connected layer. This yields an embedding which we feed into an LSTM (Hochreiter and Schmidhuber, 1997). The LSTM produces a hidden vector z_0 serving as a seed for the action sampling procedure described below.

In order to obtain a_{t+1}, we employ an *autoregressive decoder* depicted in Figure A.3. Each component a^i_{t+1} is sampled from a categorical distribution whose parameters are computed as a function of z_i. We use two kinds of functions depending on whether a^i_{t+1} corresponds to a scalar (*e.g.*, brush size) or to a spatial location (*e.g.*, a control point of a Bézier curve). In the scalar case, z_i is transformed by a fully-connected layer, otherwise we process it using several ResNet blocks followed by a series of transpose convolutions and a final convolution. After a^i_{t+1} is sampled, we obtain an updated hidden vector z_{i+1} by embedding a^i_{t+1} into a 16-dimensional code and combining it with z_i. The procedure is repeated until the entire action tuple has been generated.

For the discriminator network, we use a conventional architecture similar to DCGAN (Radford et al., 2015).

A.3. Training Details

Following standard practice in the GAN literature, we optimize the discriminator objective using Adam (Kingma and Ba, 2014) with a learning rate of 10^{-4} and β_1 set to 0.5. For generator training, we employ population-based exploration of hyperparameters (PBT) (Jaderberg et al., 2017) to find values for the entropy loss coefficient and learning rate of the policy learner. A population contains 12 training instances with each instance running 64 CPU actor jobs and 2 GPU jobs (1 for the policy learner and 1 for the discriminator learner). We assume that discriminator scores are compatible across different instances and use them as a measure of fitness in the exploitation phase of PBT.

The batch size is set to 64 on both the policy learner and discriminator learner. The generated data is sampled uniformly from a replay buffer with a capacity of 20 batches.

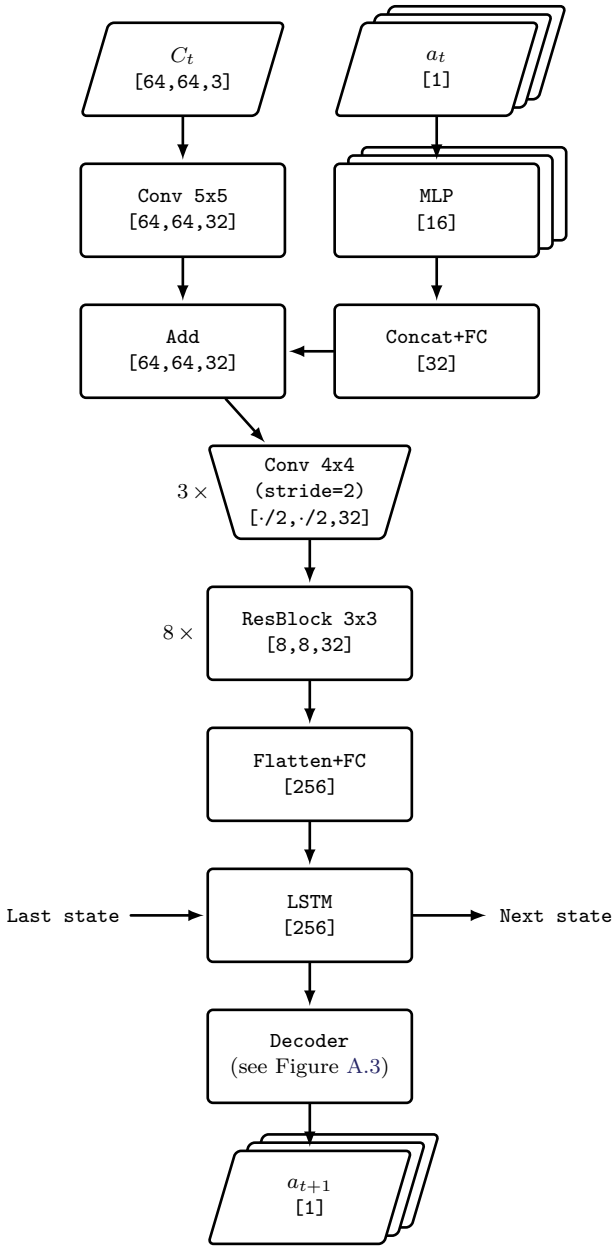


FIGURE A.2. The architecture of the **policy network** for a single step. `FC` refers to a fully-connected layer, `MLP` is a multilayer perceptron, `Conv` is a convolutional layer and `ResBlock` is a residual block. We give the dimensions of the output tensors in the square brackets. ReLU activations between the layers have been omitted for brevity.

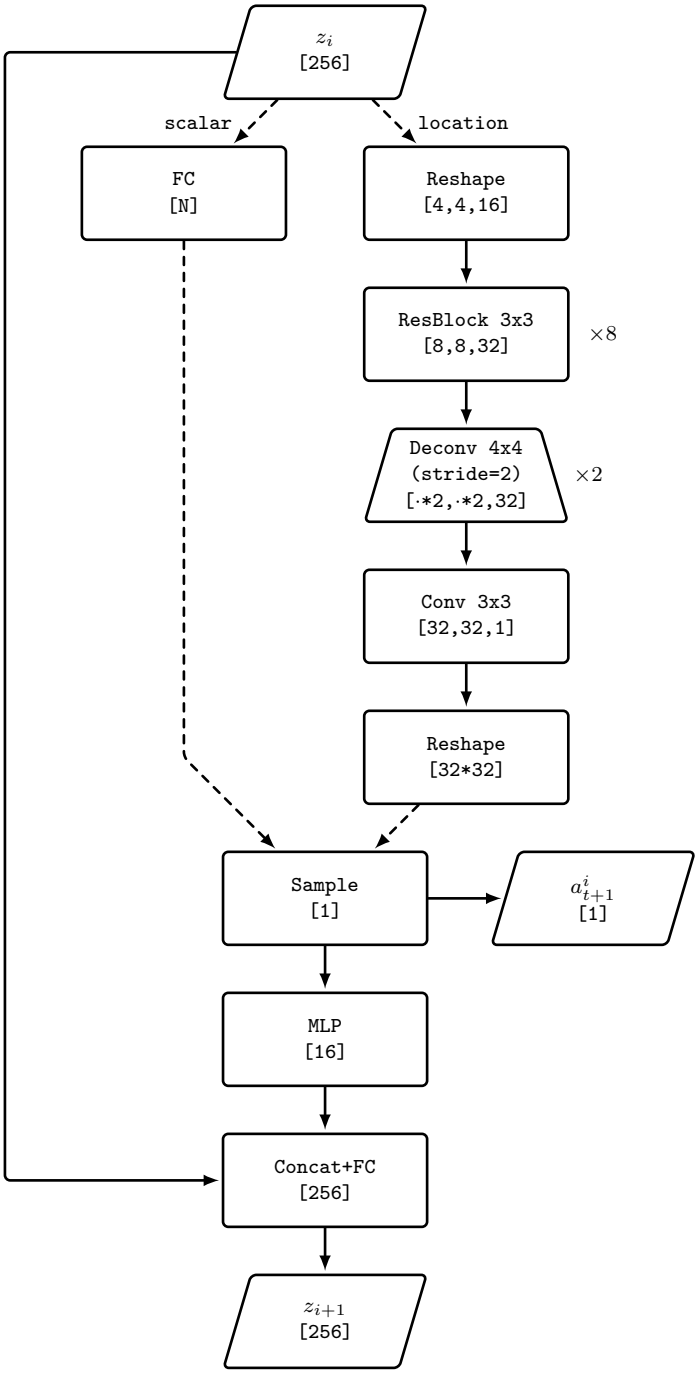


FIGURE A.3. The architecture of the **autoregressive decoder** for sampling an element a^i_{t+1} of the action tuple. The initial hidden vector z_0 is provided by an upstream LSTM. Depending on the type of the subaction to be sampled, we use either the `scalar` or the `location` branch of the diagram.