Université de Montréal

Application de méthodes d'apprentissage profond pour images avec structure additionnelle à différents contextes

par

Laurent Alsène-Racicot

Département de mathématiques et de statistique
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en mathématiques

Orientation mathématiques appliquées

2 mai 2023

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Application de méthodes d'apprentissage profond pour images avec structure additionnelle à différents contextes

présenté par

Laurent Alsène-Racicot

a été évalué par un jury composé des personnes suivantes :

Morgan Craig
(président-rapporteur)

Guy Wolf
(directeur de recherche)

Guillaume Lajoie
(membre du jury)

Résumé

Les méthodes d'apprentissage profond connaissent une croissance fulgurante. Une explication de ce phénomène est l'essor de la puissance de calcul combiné à l'accessibilité de données en grande quantité. Néanmoins, plusieurs applications de la vie réelle présentent des difficultés: la disponibilité et la qualité des données peuvent être faibles, l'étiquetage des données peut être ardu, etc. Dans ce mémoire, nous examinons deux contextes : celui des données limitées et celui du modèle économique CATS. Pour pallier les difficultés rencontrées dans ces contextes, nous utilisons des modèles d'apprentissage profond pour images avec structure additionnelle. Dans un premier temps, nous examinons les réseaux de *scattering* et étudions leur version paramétrée sur des petits jeux de données. Dans un second temps, nous adaptons les modèles de diffusion afin de proposer une alternative aux modèles à base d'agents qui sont complexes à construire et à optimiser. Nous vérifions empiriquement la faisabilité de cette démarche en modélisant le marché de l'emploi du modèle CATS.

Nous constatons tout d'abord que les réseaux de *scattering* paramétrés sont performants sur des jeux de données de classification pour des petits échantillons de données. Nous démontrons que les réseaux de *scattering* paramétrés performent mieux que ceux non paramétrés, c'est-à-dire les réseaux de *scattering* traditionnels. En effet, nous constatons que des banques de filtres adaptés aux jeux de données permettent d'améliorer l'apprentissage. En outre, nous observons que les filtres appris se différencient selon les jeux de données. Nous vérifions également la propriété de robustesse aux petites déformations lisses expérimentalement.

Ensuite, nous confirmons que les modèles de diffusion peuvent être adaptés pour modéliser le marché de l'emploi du modèle CATS dans une approche d'apprentissage profond. Nous vérifions ce fait pour deux architectures de réseau de neurones différentes. De plus, nous constatons que les performances sont maintenues pour différents scénarios impliquant l'apprentissage avec une ou plusieurs séries temporelles issues de CATS, lesquelles peuvent être tirées à partir d'hyperparamètres standards ou de perturbations de ceux-ci.

Mots clés : Apprentissage profond, Données étiquetées limitées, Transformée de *scattering*, Modèle CATS, Modèle de diffusion

Abstract

Deep learning methods are booming. An explanation of this phenomenon is the rise of computing power combined with the accessibility of large data quantity. Nevertheless, several real-life applications present difficulties: the availability and quality of data can be low, data labeling can be tricky, etc. In this thesis, we examine two contexts: that of limited data and that of the CATS economic model. To overcome the difficulties encountered in these contexts, we use deep learning models for images with additional structure. First, we examine scattering networks and study their parameterized version on small datasets. In a second step, we adapt diffusion models in order to propose an alternative to agent-based models which are complex to build and to optimize. We empirically verify the feasibility of this approach by modeling the labor market of the CATS model.

We first observe that the parameterized scattering networks perform well on classification datasets for small samples of data. We demonstrate that parameterized scattering networks perform better than those not parametrized, i.e. traditional scattering networks. Indeed, we find that filterbanks adapted to the datasets make it possible to improve learning. Moreover, we observe that the learned filters differ according to the datasets. We also verify the property of robustness to small smooth deformations experimentally..

Then, we confirm that diffusion models can be adapted to model the labor market of the CATS model in a deep learning approach. We verify this fact for two different neural network architectures. Moreover, we find that performance is maintained for different scenarios involving training with one or more time series from CATS, which can be derived from standard hyperparameters or perturbations thereof.

Keywords : Deep Learning, Limited Labeled Data, Scattering Transform, CATS model, Diffusion model

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles et des abréviations

MLP	Perceptron multicouche, de l'anglais *Multilayer Perceptron*
LL	Couche linéaire, de l'anglais *Linear Layer*
CE	Entropie croisée, de l'anglais *Cross-Entropy*
SGD	Descente de gradient stochastique, de l'anglais *Stochastic Gradient Descent*
EMA	Moyenne mobile exponentielle, de l'anglais *Exponential Moving Average*
CNN	Réseau de neurones convolutifs, de l'anglais *Convolutionnal Neural Network*
ABM	Modèle à base d'agents, de l'anglais *Agent-Based Model*
WRN	Réseau résiduel large, de l'anglais *Wide-Residual Network*
TF	Repère ajusté, de l'anglais *Tight-Frame*

MSE	Erreur quadratique moyenne, de l'anglais *Mean Square Error*
GRN	Réseau résiduel fermé, de l'anglais *Gated Residual Network*
VSN	Réseau de sélection de variables, de l'anglais *Variable Selection Network*
DDIM	Modèle implicite de diffusion de débruitage , de l'anglais *Denoising Diffusion Implicit Model*

Remerciements

Je suis heureux de présenter mon mémoire de maîtrise, l'aboutissement de deux ans de travail qui n'aurait pas été possible sans l'aide et le soutien de nombreuses personnes. Je tiens ici à leur exprimer ma gratitude.

Je tiens tout d'abord à remercier mon directeur de recherche Guy Wolf pour ses précieux conseils tout au long de mon mémoire de maîtrise. Je suis également très reconnaissant envers mes collègues - Shanel, Ben, Muawiz et Frederik - qui m'ont aidé à faire le pas des mathématiques pures vers l'apprentissage profond. Leurs idées et leurs conseils ont grandement contribué à la qualité de mon travail.

Merci à mes amis qui m'ont aidé à faire de ces années des moments inoubliables en m'entraînant dans une multitude d'événements incroyables. Merci à ma merveilleuse amoureuse Flore pour toujours avoir su me faire sourire. Enfin, merci à ma famille pour son soutien indéfectible.

Une fois de plus, je tiens à exprimer ma profonde gratitude à tous ceux qui ont contribué de près ou de loin à la réalisation de ce mémoire.

Introduction

Les modèles d'apprentissage profond permettent aujourd'hui d'effectuer une grande variété de tâches de façon impressionnante. Les percées récentes des modèles de langage, de génération d'images, ainsi que beaucoup d'autres types confirment les prouesses des intelligences artificielles actuelles. Il est désormais possible, en plus de classifier ou d'effectuer des régressions, de générer des sorties d'une qualité comparable à ce que l'on retrouve dans le monde réel, que ce soit des images, de la musique, des textes, etc. Néanmoins, ces percées sont fondées notamment sur deux facteurs qui peuvent prêter problème : l'utilisation de modèles toujours plus complexes ainsi que leur entraînement sur des jeux de données de plus en plus gros.

Différents contextes ne permettent pas de profiter de ces facteurs. Tous n'ont pas accès à la puissance de calcul des plus gros joueurs et la capacité d'entraîner les modèles sur plusieurs mois. Les données ne sont pas toujours disponibles en grand nombre, surtout si elles doivent être étiquetées et de bonne qualité [**5, 8, 70**]. Pensons au domaine médical où la confidentialité des patients est un enjeu ou encore à des séries temporelles qui sont limitées par le temps passé à récolter les données.

Une des solutions pour faire face à ce défi est d'ajouter de la structure aux modèles. Cela permet de diminuer la complexité du modèle tout en conservant un maximum d'expressivité. Les réseaux de *scattering* sont un exemple de réseau très structuré utilisé pour les tâches de classification d'images dans le cadre des données limitées [**10, 20, 47, 58**]. Les modèles hybrides permettent de réintroduire plus d'expressivité en combinant les réseaux de *scattering* à des réseaux convolutifs traditionnels. Ils atteignent des résultats proches de l'état de l'art [**47**]. Les modèles de diffusion sont un exemple de modèle génératif incorporant beaucoup de contraintes, sous la forme d'une chaîne de diffusion. Les résultats des modèles de diffusion constituent l'état de l'art. Mentionnons que ces deux types de modèles ont été originalement pensés pour traiter des images. Puisque les images sont à la fois des données complexes (de très grande dimension) et organisées (les pixels proches l'un de l'autre ont une continuité), les modèles utilisés pour les traiter sont souvent plus structurés que pour d'autres types de données.

La transformée de *scattering*, proposée dans [**39**], consiste en une cascade d'ondelettes et de non-linéarités de module complexe. Elle peut être vue comme un réseau de neurones convolutifs avec des filtres fixes et prédéterminés (les ondelettes). Cette construction possède des propriétés mathématiques importantes sous certaines hypothèses : invariance translationnelle, invariance sous certains difféomorphismes, non-expansivité, etc [**10, 39**]. Les réseaux de *scattering* peuvent être utlisés comme modèle mathématique pour étudier les réseaux de neurones convolutifs [**10, 41**].

Dans les précédents travaux, les filtres sont construits de façon standard, c'est-à-dire qu'une ondelette de Morlet complexe est dilatée, tournée, et autres pour générer une banque de filtres. Cette dernière constitue un repère ajusté (de l'anglais *tight-frame*) couvrant le plan fréquentiel et possédant des propriétés de préservation d'énergie bien établies [**10, 38**]. Nous étudions dans ce mémoire si cette construction est optimale, ou s'il est possible d'apprendre les paramètres servant à générer les filtres (en conservant la forme de Morlet) afin qu'ils soient adaptés au jeu de données. Nous appelons cette approche la transformée de *scattering* paramétrée.

Les modèles de diffusion sont des modèles génératifs fondés sur l'idée de débruiter graduellement un bruit gaussien en un échantillon qui respecte la distribution des données d'entraînements [**60, 62, 28**]. En contraste avec les réseaux antagonistes génératifs qui étaient la principale méthode générative avant l'essor des modèles de diffusion et qui produisent l'échantillon à l'aide d'une unique application d'un réseau de neurones, les modèles de diffusion produisent l'échantillon petit à petit en appliquant à répétition un même réseau de neurones. Ce faisant, la tâche est simplifiée et le modèle a la possibilité de s'auto-corriger. La chaîne de diffusion est ainsi un exemple de structure additionnelle qui permet de produire des échantillons de grande qualité, même avec des données limitées [**55**].

Dans ce mémoire, nous adaptons les modèles de diffusion pour reproduire les résultats d'une variante du modèle économique CATS [**1, 2, 21**] (ou plus précisément d'une sous-partie de cette dernière) à l'aide de l'apprentissage profond [**13**]. Le modèle CATS est un modèle à base d'agents complexe qui simule une économie en produisant des séries temporelles. La sous-partie étudiée - le marché du travail - implémente une politique de recherche et d'appariement. Le caractère aléatoire du processus nous incite à utiliser un modèle génératif, et le fait que l'objet à l'étude soit des séries temporelles rend l'utilisation d'un modèle de diffusion appropriée. La popularisation de cette approche permettrait d'envisager des simulations d'interactions de populations de grande envergure. En effet, la paramétrisation par réseaux de neurones de la relation entre les différentes populations augmenterait la vitesse de simulation et simplifierait l'ajustement des paramètres.

Article adapté

Les sections 1.7 et 3.1 sont des adaptations du contenu de l'article *Parametric Scattering Networks* [**22**], dont je suis l'un des co-auteurs. J'ai contribué aux expérimentions (voir la sec. 3.1.4), à la construction et à la mise en application de l'étude de la robustesse (voir la sec. 3.1.3) ainsi qu'à la rédaction de l'article. Le code relatif à l'article est disponible au lien suivant : https://github.com/bentherien/parametricScatteringNetworks.

Chapitre 1

Cadre d'étude

1.1. Cadre général

Un réseau de neurones artificiel, dans le cadre de l'apprentissage profond, est un modèle mathématique qui apprend des relations entre une ou des entrées et une ou des sorties. Nous utilisons le terme « apprendre » car le réseau de neurones est paramétré, et que ses paramètres vont être optimisés lors d'une phase d'apprentissage. Nous allons nous restreindre au cas d'apprentissage dit « supervisé », c'est-à-dire que les données d'entraînements sont étiquetées. Par exemple, nous pourrions vouloir entraîner un réseau de neurones à distinguer des images de chats de celles de chiens, et ce en l'optimisant à l'aide d'images de chats et de chiens accompagnée de leur étiquette correspondante. Ainsi, une image de chien serait accompagnée de la mention « chien », et similairement pour une image de chat.

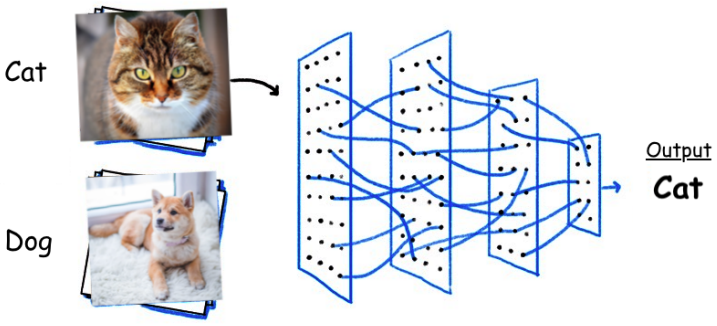


Fig. 1.1. Schéma conceptuel d'un réseau de neurones pour la classification d'images de chat ou de chien. Image tirée de [**57**].

Pour parvenir à faire cette optimisation, une fonction de perte est utilisée. La fonction de perte mesure l'écart entre la sortie produite par le réseau de neurones appliqué à une donnée d'entraînement et l'étiquette associée à la donnée d'entraînement. À partir de cette

fonction de perte, nous pouvons calculer le gradient de la fonction de perte respectivement aux paramètres du réseau de neurones en utilisant la technique de propagation arrière du gradient [**51**], et ainsi minimiser la fonction de perte à l'aide de techniques d'optimisation [**33, 50**]. Cette façon de procéder est fondée sur le principe de minimisation du risque empirique [**67**]. En répétant cette opération d'optimisation sur un grand nombre d'exemples, le réseau devrait s'améliorer progressivement à réaliser la tâche, ainsi qu'à généraliser à de nouveaux exemples. Différents problèmes peuvent survenir limitant l'efficacité du réseau de neurones à effectuer la tâche. Un des principaux aspects du domaine de l'apprentissage profond est précisément de déterminer quelles techniques peuvent être employées dans quelles situations, afin de remplir cet objectif.

1.2. Perceptron multicouche

Concrétisons la discussion précédente à l'aide de l'exemple du perceptron multicouche (MLP, de l'anglais *multilayer perceptron* et illustré à la fig. 1.2) [**31**], un des premiers réseaux de neurones employé et partie centrale d'un grand nombre d'architectures de réseau plus compliquées.

Intéressons-nous à une tâche de classification, c'est-à-dire qu'étant donné une entrée $x \in \mathbb{R}^n$, nous cherchons à produire un vecteur de probabilité $\hat{\boldsymbol{y}} = f(\mathbf{x}; \theta)$, où la i-ième composante y_i correspond à la probabilité estimée par le réseau de neurones f que $\mathbf{x}$ appartienne à la classe i. Le but de la phase d'entraînement est d'optimiser les paramètres θ afin d'approximer une fonction f^* qui envoie l'entrée $\mathbf{x}$ sur l'encodage *one-hot* de l'étiquette $\mathbf{y}$.

Le perceptron multicouche performe les opérations suivantes. Tout d'abord, une couche linéaire (LL, de l'anglais *Linear Layer*) est appliquée :

$$\mathbf{a}^{(1)}(\mathbf{x}) = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \tag{1.2.1}$$

à l'aide d'une matrice de poids $\mathbf{W}^{(1)} \in \mathbb{R}^{n \times m_1}$ et d'un vecteur de biais $\mathbf{b}^{(1)} \in \mathbb{R}^{m_1}$.

Puis une fonction d'activation non-linéaire ϕ est appliquée composante par composante :

$$\mathbf{h}^{(1)}(\mathbf{x}) = \phi(\mathbf{a}^{(1)}(\mathbf{x})). \tag{1.2.2}$$

Un choix commun de fonction d'activation est la fonction ReLU [**43**] définie par $\mathrm{ReLU}(x) = \max(x,0)$. Cette procédure est répétée un certain nombre L de fois, correspondant au nombre de couches cachées du réseau de neurones. On obtient pour $0 < k \leq L + 1$

$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{W}^{(k)} h^{(k-1)}(\mathbf{x}) + \mathbf{b}^{(k)} \tag{1.2.3}$$

$$\mathbf{h}^{(k)}(\mathbf{x}) = \phi(\mathbf{a}^{(k)}(\mathbf{x})). \tag{1.2.4}$$

Pour obtenir les probabilités finales, la fonction SOFTMAX est utilisée. Elle est définie par composante comme suit :

$$\text{SOFTMAX}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \quad \text{pour } i = 1,2,\ldots,K. \tag{1.2.5}$$

Ainsi, la valeur finale du perceptron multicouche est

$$\hat{\mathbf{y}} = f(\mathbf{x};\theta) = \text{SOFTMAX}(\mathbf{a}^{(L+1)}(\mathbf{x})). \tag{1.2.6}$$

La classe prédite correspond à l'indice associé à la plus grande valeur. Nous pouvons également identifier les paramètres du réseau $\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$.

Pour les optimiser, une fonction de perte commune est la fonction d'entropie croisée (CE, de l'anglais *Cross-Entropy*) définie par

$$l(\hat{\mathbf{y}},\mathbf{y}) = -\sum_i y_i \log(\hat{y}_i). \tag{1.2.7}$$

L'optimisation se fait généralement à l'aide de l'algorithme de descente de gradient stochastique (SGD, de l'anglais *Stochastic Gradient Descent*) [**50**] ou ADAM [**33**]. Un régularisateur peut être utilisé afin de contrôler la taille des poids [**44**]. Ce dernier est une fonction qui dépend des poids qui s'ajoute à la fonction de perte. La technique de moyenne mobile exponentielle (EMA, de l'anglais *Exponential Moving Average*) peut aider à atteindre le minimum local de la fonction de perte en faisant une moyenne pondérée des paramètres obtenus aux différents pas d'entraînement (la pondération décroît exponentiellement plus on remonte vers le début de l'entraînement)[**64**].

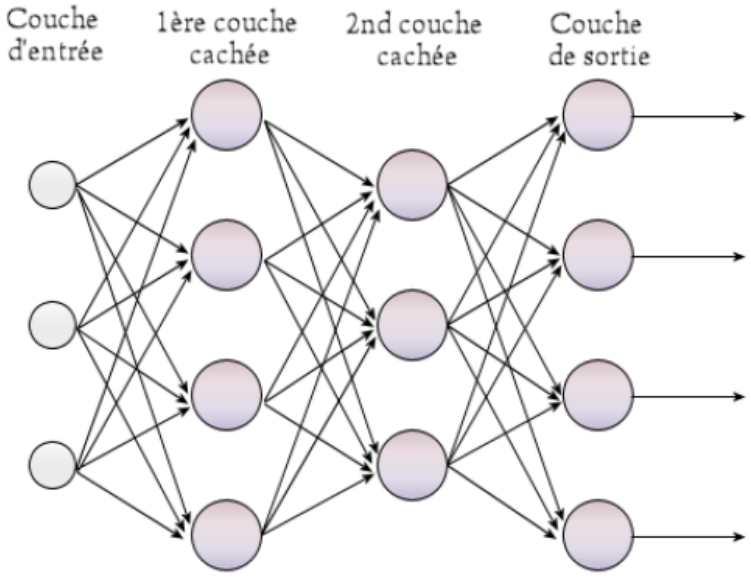


Fig. 1.2. Illustration d'un perceptron multicouche. Image tirée de [**14**].

1.3. Réseaux de neurones convolutifs

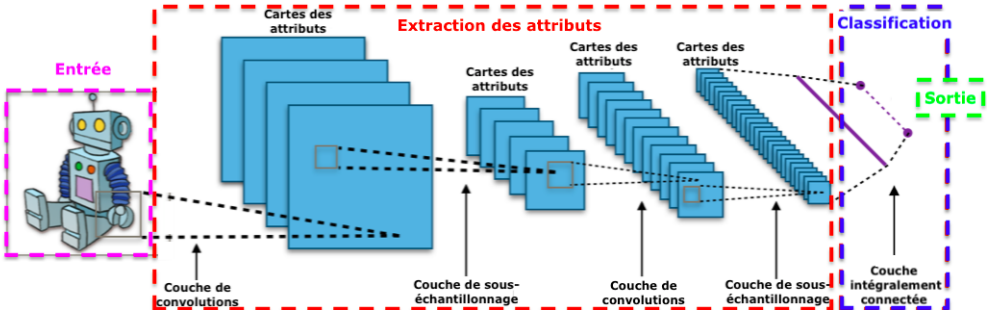


Fig. 1.3. Illustration d'un réseau de neurones convolutifs. Image tirée de [**15**].

Nous explorons une autre structure de réseau de neurones, le réseau de neurones convolutifs (CNN, de l'anglais *Convolutionnal Neural Network* et illustré à la fig 1.3). Ce dernier a été conçu pour traiter des images en entrées. Les images sont généralement beaucoup plus complexes à traiter, considérant que la dimension d'une image peut facilement atteindre des valeurs de l'ordre des millions. En effet, une image peut être représentée par un tenseur de dimension $C \times H \times W$, où C est le nombre de canaux (3 pour une image couleur), H est la hauteur de l'image et W sa largeur. Le phénomène de fléau de la dimension [**7**] implique que sans présupposés supplémentaires, il sera très difficile de traiter des données de si grande dimension. Ainsi, la plupart des modèles exploitent le fait que les pixels sont ordonnés dans un rectangle, avec des pixels voisins ainsi que des pixels éloignés. Cette observation permet de définir un voisinage d'un pixel, sur lequel il est possible de calculer des moyennes pondérées, résumant l'information et luttant contre le fléau de la dimension. Un autre aspect à considérer dans le traitement d'images est le nombre de paramètres des réseaux. Plus le nombre de paramètres est grand, plus l'optimisation est complexe et lente. Il est possible de contrer ceci en utilisant un plus grand nombre de données d'entraînement, encore faut-il que ces données existent et soient accessible. Pour ces raisons, les réseaux de neurones traitant des images utilisent souvent une structure qui permet de limiter au maximum le nombre de paramètres et qui exploite les particularité des images.

Les réseaux de neurones convolutifs utilisent précisément cette approche. Ils sont constitués de trois différents types de couches : les couches convolutionnelles, les couches de sous-échantillonnage (ou de regroupement) et les couches linéaires.

Les couches convolutionnelles sont basées sur l'opération de convolution à partir de filtres (voir la fig. 1.4). Un filtre $\mathbf{K}$ de dimension $m \times n$ (avec m et n impairs) est une matrice de paramètres. Avec $a = \frac{m-1}{2}$ et $b = \frac{n-1}{2}$, une convolution entre un filtre $\mathbf{K}$ et un des canaux $\mathbf{X}$ d'une image est définie par

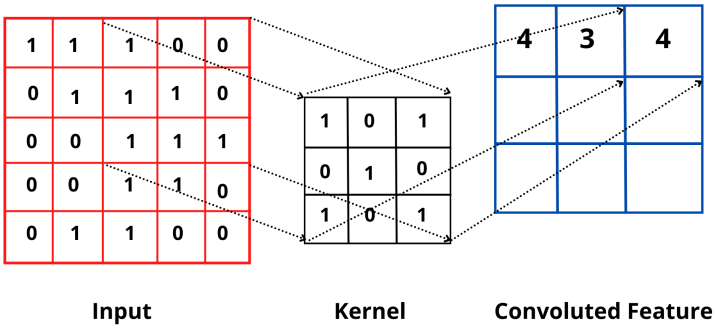


Fig. 1.4. Illustration d'une convolution en 2D. Les caractéristiques extraites sont déterminées à l'aide d'une convolution 2D du filtre sur l'image d'entrée. Image tirée de [**18**].

$$(\mathbf{X} \star \mathbf{K})(x,y) = \sum_{u=-a}^{a} \sum_{v=-b}^{b} \mathbf{X}(x-u, y-v)\mathbf{K}(u,v). \tag{1.3.1}$$

Remarquons que dans l'équation 1.3.1, il n'est pas possible de calculer la convolution pour les pixels se situant sur le bord de l'image, car le filtre « sortirait » de l'image. Pour pallier ce problème, il est possible d'entourer l'image de pixels ayant comme valeur 0 (i.e. des pixels noirs). On appelle cette opération *padding* et on lui associe une valeur correspondant au nombre de bordures de zéros ajoutées. Un bémol est que cette opération introduit des contours plus foncés le long des bords.

Il est également possible de calculer la convolution pour un nombre limité de pixels, par exemple un pixel sur deux. Le nombre de décalages du filtre avant de calculer la convolution pour un nouveau pixel (x',y') depuis (x,y) s'appelle la *stride*.

Étant donné une *stride* S, un *padding* P et un filtre de dimension $m \times n$, la nouvelle largeur et hauteur sont données par

$$H_{out} = \left\lfloor 1 + \frac{H + 2P - m}{S} \right\rfloor, \; W_{out} = \left\lfloor 1 + \frac{W + 2P - n}{S} \right\rfloor. \tag{1.3.2}$$

L'opération de convolution est répétée pour chacun des canaux de l'image, et le résultat final est donné par la somme des résultats avec un biais :

$$\text{out} = \text{bias} + \sum_{c=1}^{C_{in}} \mathbf{X}_c \star \mathbf{K}_c. \tag{1.3.3}$$

Le résultat est de dimension $1 \times H_{out} \times W_{out}$. Toutes les opérations sont alors répétées avec des filtres différents (dont le nombre est laissé à la discrétion du concepteur) et les résultats sont concaténés pour obtenir un résultat de dimension $C_{out} \times H_{out} \times W_{out}$. Une fonction d'activation peut être utilisée sur les sorties.

Les couches de regroupement visent à diminuer la dimension des données, généralement en combinant plusieurs neurones adjacents en un. Il est commun d'utiliser un pavage 2×2 des images, et de remplacer chaque grille du pavage par une unique valeur. Pour cela, on peut calculer la moyenne (en anglais, *average pooling*) ou le maximum (en anglais, *max pooling*). Le pavage pourrait s'entrecouper, ou être d'une dimension autre. La figure 1.5 illustre le fonctionnement.

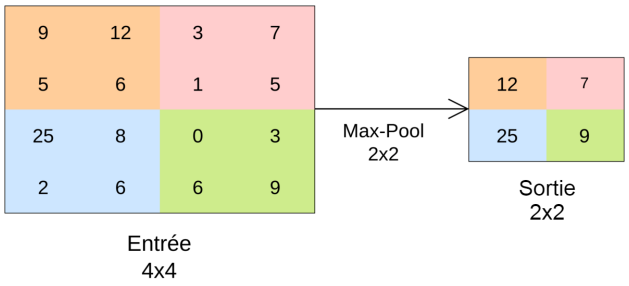


Fig. 1.5. Illustration du *max pooling*. Chaque élément de la sortie est le maximum des éléments de la grille correspondante dans l'entrée. Image tirée de [**16**].

Finalement, la couche linéaire convertit les caractéristiques extraites par le réseau de neurones en un objet qui correspond à la tâche. Par exemple, s'il s'agit d'une tâche de classification, les caractéristiques seront projetées sur un vecteur de dimension le nombre de classes. L'équation utilisée est celle de la transformation linéaire décrite par l'équation 1.2.3.

1.3.1. U-Net

En guise d'exemple d'un réseau de neurones convolutifs, nous présentons le U-Net, tel que décrit dans [**49**]. À la section 2.2.1.1, nous présenterons des modifications qui peuvent être faites au U-Net.

Le U-Net, dont l'architecture est présenté à la figure 1.6, est un réseau de neurones convolutifs créé initialement pour la segmentation d'images médicales. Il présente l'avantage de pouvoir facilement conserver la dimension de l'entrée, ce qui sera pertinent à la section 2.2.1.1.

Il est constitué d'une phase de contraction (côté gauche de la figure 1.6) et d'un phase d'expansion (côté droit). La phase de contraction est constituée d'une répétition de blocs de convolutions et de couches de regroupement. Les blocs de convolutions sont constitués de deux convolutions successives 3×3, chacune suivie d'un ReLU. Le nombre de canaux est doublé à chaque bloc. Une copie est alors enregistrée en mémoire pour usage dans la phase d'expansion. S'ensuit l'application d'un *max-pooling* 2×2, qui vient réduire la taille des cartes des attributs (en anglais, *features maps*).

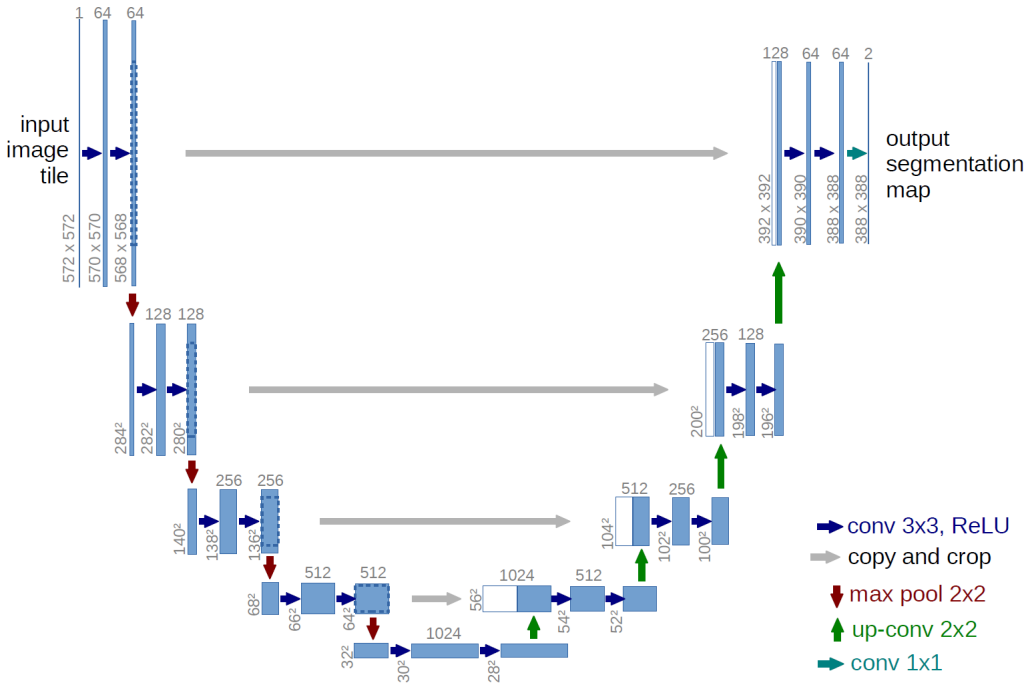


Fig. 1.6. Illustration du U-Net. Image tirée de [**49**]

La phase d'expansion est elle aussi une répétition d'opérations. Chaque répétition débute par l'application d'une *up-convolution*. Cette opération comprend l'opération de *upsampling*, ainsi qu'une convolution qui réduit le nombre de canaux par 2. Différentes variantes de *upsampling* existent, mais une façon simple d'y parvenir est de remplacer chaque pixel par un carré 2×2 de pixels, tous ayant la valeur du pixel initial (voir la fig. 1.7). Le résultat est ensuite concaténé avec la copie de la carte des attributs correspondante (sur le même étage dans la figure 1.6), qui est rognée pour avoir la bonne taille. Le rognage est nécessaire à cause de l'absence de *padding*, ce qui cause une réduction de la taille de l'image à chaque application de convolution. Pour finir chaque répétition, deux convolutions successive 3×3 suivies d'un ReLU sont appliquées comme dans la phase de contraction. Finalement, une dernière couche de convolutions 1×1 est appliquée à la fin du réseau pour obtenir le nombre de canaux voulus (par exemple 1 par classe, dans la tâche de segmentation, ou unique canal, pour une tâche de classification).

1.4. Attention

Le mécanisme d'attention a été proposé en 2017 [**68**] et depuis s'est démocratisé dans une énorme variété de modèle. Il est à la base des populaires *transformers*, mais est également incorporé dans les modèles plus récents de U-Net, par exemple. Nous le décrivons ainsi que sa variante d'attention à plusieurs têtes, puis nous distinguons l'auto-attention de l'attention croisée.

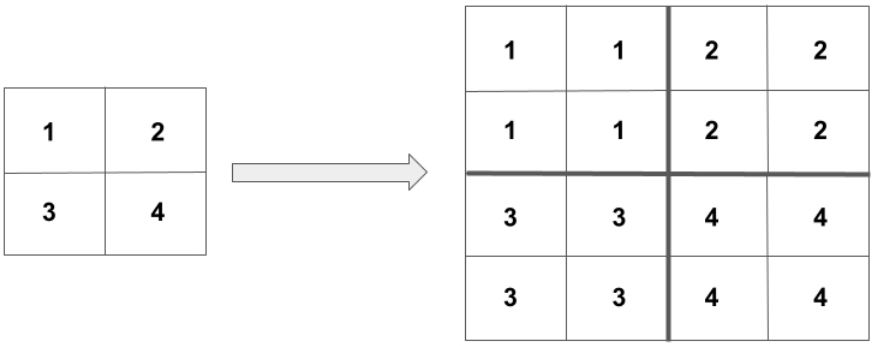


Fig. 1.7. Illustration de l'*upsampling* selon le plus proche voisin. Chaque entrée est remplaçée par une grille d'entrées de la même valeur.

Une couche d'attention compare des requêtes, concaténées dans une matrice $Q \in \mathbb{R}^{n_1 \times d_k}$, à des paires de clés et de valeurs, respectivement concaténés dans les matrices $K \in \mathbb{R}^{n_2 \times d_k}$ et $V \in \mathbb{R}^{n_2 \times d_v}$. Chaque requête produit son propre résultat indépendant, mais toutes les requêtes sont traitées en parallèles. Pour chaque requête, des poids d'attention sont calculés à l'aide de la requête et des clés, et le résultat final est une combinaison linéaire des valeurs dont les coefficients sont ces poids d'attention. La matrice combinant les sorties de toutes les requêtes est donnée par

$$\text{Attention}(Q,K,V) = \text{SOFTMAX}(\frac{QK}{\sqrt{d_k}})V. \tag{1.4.1}$$

L'attention à plusieurs têtes permet de considérer de multiples représentations en effectuant plusieurs mécanismes d'attention en simultané, tout en garder la complexité de calcul comparable à la couche d'attention simple. Supposons que les requêtes, clés et valeurs sont d'une dimension commune d_{model}, et que nous souhaitons utiliser h têtes. Nous pouvons pour chaque tête projeter chaque élément (Q, K et V) dans l'espace de dimension $d_k = d_v = d_{\text{model}}/h$ et calculer leur attention. Ensuite nous pouvons regrouper les sorties en les concaténant, puis de nouveau projeter vers l'espace de dimension d_{model}. Mathématiquement, nous avons

$$\text{MultiHead}(Q,K,V) = \text{Concat}(head_1, \ldots, head_h)W^O, \text{ où} \tag{1.4.2}$$

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \text{ avec} \tag{1.4.3}$$

$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ et $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$. Pour utiliser l'attention à plusieurs têtes défini ici, il faut que le nombre de tête h divise la dimension du modèle d_{model}.

La couche d'auto-attention consiste à utiliser la même matrice X comme requête, clé et valeur et est définie par

$$\text{SelfAttention}(X) = \text{MultiHead}(X,X,X). \tag{1.4.4}$$

Elle est typiquement incluse dans un bloc résiduel défini par

$$\text{out} = \text{LayerNorm}(X + \text{SelfAttention}(X)). \tag{1.4.5}$$

Selon les usages, cette dernière équation peut être référencée directement comme étant une couche d'auto-attention.

La couche d'attention croisée est définie par

$$\text{CrossAttention}(X,C) = \text{MultiHead}(X,C,C). \tag{1.4.6}$$

Elle est elle aussi typiquement incluse dans un bloc résiduel :

$$\text{out} = \text{LayerNorm}(X + \text{CrossAttention}(X,C)). \tag{1.4.7}$$

La matrice C peut être construite de plusieurs façons. Dans ce travail, nous définissons $C = \text{Concat}(X,c)$, où c correspond à un conditionnement du réseau de neurones. Nous référons aux sections 2.2.1.1 et 2.2.1.2 pour l'usage précis que nous en ferons.

1.5. Données limitées

1.5.1. Problématique

Une des principales raisons qui explique l'essor fulgurant de l'apprentissage profond est l'amélioration des capacités de calcul des machines. Cela a permis d'augmenter la complexité des modèles, et de les entraîner sur des corpus de plus en plus gros [**32**]. Nous pouvons citer ici un des derniers modèles qui a conquis l'univers médiatique : le modèle de langage GPT-3 derrière l'agent conversationnel ChatGPT à son lancement possédait 175 milliards de paramètres et avait été entraîné sur un corpus de plus de 570GB après filtrage [**9**]. Néanmoins, il reste des applications où la quantité de données est limitée. Cela est d'autant plus vraie dans le cadre de l'apprentissage supervisé, où le processus d'étiquetage peut être fastidieux [**73**].

Un exemple d'une telle application est dans le domaine médical [**23**]. Le développement de diagnostics basés sur l'intelligence artificielle est de plus en plus commun. Néanmoins, la confidentialité des dossiers des patients, le coût d'obtention d'avis d'experts ainsi que le risque d'étiquetage erroné rendent complexe la tâche d'obtenir des données fiables et en grand nombre.

1.5.2. Stratégies

1.5.2.1. Techniques de généralisation.

Les techniques de généralisation cherchent à améliorer la capacité du modèle à généraliser, parfois au coût d'une plus grande erreur d'entraînement. Cela est particulièrement intéressant dans le contexte des données limitées, car la plus grande déficience des modèles est souvent leur capacité à généraliser.

Une première technique est celle de l'abandon (en anglais, *dropout*). Durant la phase d'entraînement, des neurones, voire des unités complètes, sont désactivés avec une certaine probabilité. Cela permet de réduire le surajustement [**27**]. Durant la phase de test, tous les neurones sont utilisés.

Une seconde technique est de normaliser les données à différentes couches du réseau de neurones. La normalisation peut s'effectuer selon le lot (en anglais, *batch normalization*) [**30**] ou selon la couche (en anglais, *layer normalization*) [**3**] et est illustrée à la fig. 1.8. Dans les deux cas, la normalisation s'exprime mathématiquement comme suit :

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}}\gamma + \beta, \tag{1.5.1}$$

où μ est la moyenne, σ^2 est la variance, ε est un petit terme pour éviter les erreurs numériques, γ et β sont des paramètres appris.

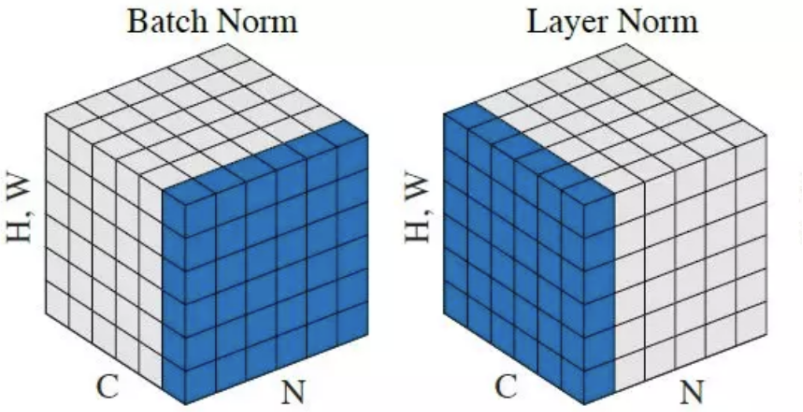


Fig. 1.8. Normalisation selon le lot (à gauche) et selon la couche (à droite) pour N tenseurs de dimensions (C,H,W). Image tirée de [**65**]

Le choix de la fonction de perte peut également être un aspect à considérer. Barz et al. [**6**] propose de prendre comme fonction de perte la distance cosinus

$$L_{cos}(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{f_\theta(\boldsymbol{x}) \cdot \boldsymbol{y}}{\|f_\theta(\boldsymbol{x})\|\|\boldsymbol{y}\|}. \tag{1.5.2}$$

Une explication avancée pour justifier ce choix est qu'il diminue la complexité de la tâche en concentrant toute l'importance sur l'angle en enlevant celle de la norme.

1.5.2.2. Augmentation de données.

Une autre technique est d'élargir artificiellement le nombre de données d'entraînement. Les données ajoutées peuvent être synthétiques ou augmentées. Nous discutons du second cas, utilisé dans ce travail. Pour augmenter des données, il faut appliquer aléatoirement des transformations sur les données existantes (voir la fig. 1.9). En plus d'élargir la quantité de données pour que le réseau de neurones puisse ajuster ses paramètres, cette technique a l'avantage de créer des invariances qui correspondent aux transformations. Usuellement, les transformations sont choisies ou conçues spécifiquement pour le problème traité. Ainsi, par exemple, une translation est appropriée pour un problème de reconnaissance de chat dans des images, car l'emplacement du chat ne devrait (en général) pas avoir une influence sur sa catégorisation. Les transformations peuvent être géométriques, agir sur les couleurs, ajouter du bruit, ou appliquer bien d'autres effets. La procédure d'auto-augmentation [**17**] permet de considérer automatiquement un grand nombre de polices d'augmentation, et d'apprendre les plus efficaces.

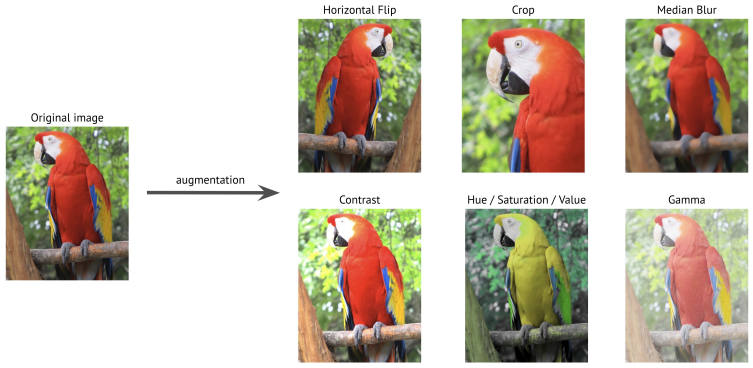


Fig. 1.9. Différentes augmentations sont appliquées sur une image. Image tirée de [**12**].

1.5.2.3. Apprentissage par transfert.

L'apprentissage par transfert consiste à utiliser un réseau de neurones entraîné sur un autre jeu de données, et à faire un réglage fin des paramètres sur le jeu de données principal. Durant le réglage fin, une partie des paramètres du réseau peut être gelée, c'est-à-dire ne pas être optimisée.

Cette approche est basée sur l'observation que l'extraction d'attributs peut être commun pour des tâches distinctes, mais similaires. Ainsi, les premières couches de réseaux de neurones entraînés sur des jeux de données distincts peuvent avoir des paramètres similaires,

ayant la même fonction [**71**]. Une interprétation de l'efficacité de cette technique est donc que le réseau de neurones original permet l'extraction d'attributs intéressants, et que la phase de réglage fin permet l'utilisation de ces attributs pour la tâche actuelle.

C'est une approche très puissante, car la capacité des modèles importés peut être très grande et l'information utilisée pour l'entraînement est décuplée. Néanmoins, plus les données sont différentes des données d'entraînement du modèle original - que ce soit par rapport au format (p. ex. couleur vs noir et blanc), au style (p. ex. dessin vs photo), au contenu (p. ex. objets vs textures), à la tâche (p. ex. reconnaissance d'objets vs diagnostic médical), etc. - moins le transfert sera efficace.

1.5.2.4. Structure additionnelle.

Cette dernière stratégie est très générale. Il a été discuté que le nombre limité d'exemples ne permet pas d'optimiser un grand modèle, lequel pourrait être nécessaire pour réaliser la tâche. Le choix de l'architecture du modèle est un exemple de structure : combien de couches cachées faut-il utiliser? Quels types d'unités faut-il choisir? Nous discutons de deux autres exemples de structure additionnelle.

Premièrement, il est possible de fixer un certain nombre de paramètres. Dans le contexte des réseaux convolutifs, une façon de le faire est d'utiliser des filtres prédéfinis. Un exemple commun de filtre utilisé est le filtre de Morlet (défini à la section 1.7.1). Ce choix est motivé par l'observation que les premiers filtres d'AlexNet sont très similaires aux filtres de Gabor (voir la fig. 1.10), lesquels sont identiques aux filtres de Morlet à une normalisation près. Les réseaux de *scattering* peuvent être vus comme des CNN dont les filtres sont fixes. Il est possible de combiner un réseau de *scattering* avec un CNN, pour obtenir un modèle hybride. Nous détaillerons le fonctionnement de ces réseaux à la section 1.7.

(a) Filtres d'AlexNet. **(b)** Filtres de Gabor.

Fig. 1.10. Les premiers filtres d'AlexNet et les filtres de Gabor sont très semblables, justifiant l'intérêt d'utiliser ces derniers comme filtres prédéfinis. Images tirées de [**35**] et [**56**].

Remarquons la ressemblance de cette approche avec la technique précédente, où une partie des paramètres est également prédéfinie (mais pas nécessairement fixée). Dans un cas,

elle l'est par choix humain motivé par l'expérience, dans l'autre, par un entraînement sur un autre jeu de données.

Deuxièmement, le réseau de neurones peut être intégré à une structure plus large. C'est ce qui est fait par exemple dans les modèles de diffusion. Dans ces derniers, la tâche de générer une image est découpée en un grand nombre d'étapes. À chacune d'entre elle, un réseau de neurones débruite légèrement l'image, jusqu'à l'obtention d'une image claire (voir la fig. 1.11). Nous référons à la section 1.8 pour une description détaillée.

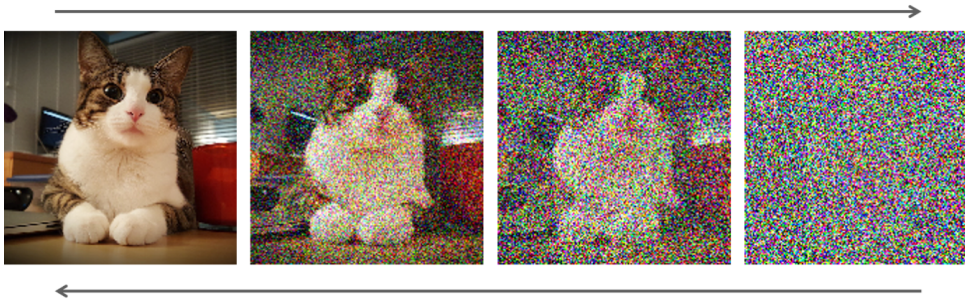


Fig. 1.11. Illustration de la chaîne de diffusion des modèles de diffusion. Le modèle permet de faire une transition entre une image non bruitée et un bruit gaussien pur. Image tirére de [**66**].

1.6. Modèle économique CATS

1.6.1. Modèle à base d'agents

Les modèles à base d'agents (ABM, de l'anglais *Agent-Based Model*) sont des outils utilisés pour étudier l'interaction de populations hétérogènes d'entités appelées agents. De façon générale, la population évolue par pas de temps discrets, où les agents se comportent selon des règles heuristiques ou idéalisées. Dans des ABM plus complexes, les agents peuvent se comporter de façon à optimiser une fonction objectif.

Ces modèles sont populaires dans une variété de domaines. Par exemple, en biologie, un ABM peut être utilisé pour modéliser l'évolution d'une population cellulaire qui présente des hétérogénéités provenant de mutations aléatoires dans des sous-populations. En économie, il est possible d'étudier une population d'individus regroupés en différentes classes, telles que les employeurs et les employés.

L'objectif est généralement d'approximer la dynamique de la population d'agents dans le monde réel en employant un modèle simplifié. Pour ce faire, les modèles ABM se contentent usuellement de reproduire des statistiques au niveau de la population. Les ABM s'appuient

sur un ensemble de paramètres qui régissent le comportement des agents. Un des principaux défis consiste à ajuster ces paramètres afin d'obtenir une bonne approximation.

L'évolution dynamique dans le temps de l'état des agents dépend à la fois du contexte ainsi que des interactions de ces derniers. L'état d'un agent est une propriété de tout agent qui évolue dans le temps. Le contexte d'un agent est une propriété fixe de chaque agent qui influence ses interactions. L'interaction d'un agent est une liste d'autres agents avec lesquels il peut interagir.

Nous nous intéresserons à remplacer les règles heuristiques et la recherche de paramètres par l'utilisation et l'entraînement d'un réseau de neurones.

1.6.2. Fonctionnement

Le modèle CATS [**1, 2**] est un modèle keynésien qui étudie une population d'individus, divisée en propriétaires de firmes (capitalistes) et travailleurs. Les individus suivent des règles comportementales heuristiques. Le modèle comprend un secteur public et bancaire. Le modèle est stationnaire et modélise une économie à l'équilibre, soumise à des fluctuations endogènes.

Les interactions des agents dans le modèle CATS ont lieu sur quatre marchés, à savoir le marché du crédit, le marché du travail, le marché du capital (K) et le marché de la consommation (C), ce qui est illustré à la fig. 1.12. Les interactions entre marchés fonctionnent généralement selon une politique de recherche et d'appariement (en anglais, *search-and-match*). Les participants sont ainsi divisés en fournisseurs et consommateurs. Les consommateurs ont chacun une consommation souhaitée et se voient attribuer une sélection aléatoire de fournisseurs. Ensuite, les clients attendent dans une file d'attente (qui est déterminée au hasard à chaque itération) et interagissent séquentiellement avec leurs fournisseurs assignés, du moins cher au plus cher, jusqu'à ce que la consommation souhaitée soit atteinte ou que l'offre soit épuisée.

Le modèle CATS est un exemple d'ABM, où l'état des agents inclue des attributs tels que la consommation, l'emploi ou l'épargne. Le contexte d'un agent dépend du type d'agent (propriétaire de firme ou travailleur) et l'interaction d'un agent est déterminée au hasard par les politiques de recherche et d'appariement des marchés.

1.6.3. Marché du travail

Le marché du travail est la partie du modèle CATS qui met à jour l'emploi des travailleurs. Considérons W travailleurs, n_F firmes et un temps discret t. Nous pouvons associer à chaque travailleur un entier entre 0 et n_F, correspondant à la firme dans laquelle il travaille, où à l'état de sans-emplois. Le marché du travail met à jour le vecteur regroupant ces entiers.

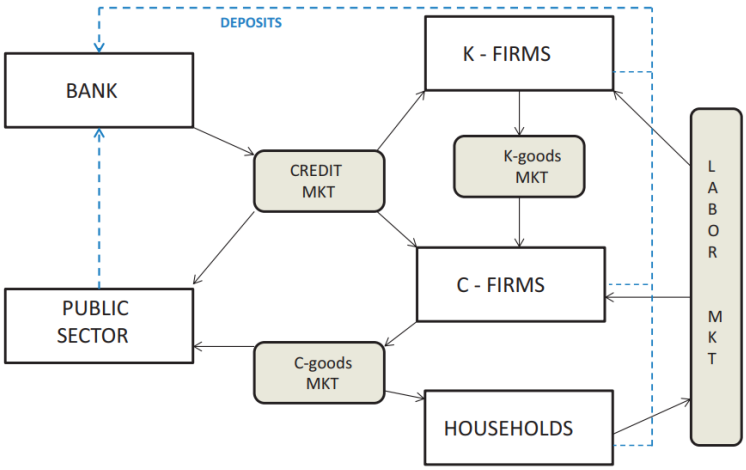


Fig. 1.12. Schéma conceptuel du modèle CATS. Les marchés permettent des interactions entre les différents agents. Image tirée de [**1**].

Pour ce faire, chaque firme détermine si elle a un surplus ou un manque d'employés, à l'aide d'informations $H \in \mathbb{R}^{6\times n_F}$. La matrice d'informations H est la concaténation des six vecteurs suivants : la productivité du travail, la demande prévue, les salaires, les emprunts, la liquidité et le travail efficace. Si une firme est en surplus, elle renvoie aléatoirement des employés jusqu'à en avoir son nombre optimal. Si elle est en manque, elle entre dans le processus de recherche et d'appariement. Chaque travailleur sans emplois applique séquentiellement à k firmes distinctes. Les firmes engagent les travailleurs pour en avoir leur nombre optimal.

À la section 3.2, nous présentons une expérimentation dans laquelle nous cherchons à reproduire les résultats de ce processus à l'aide d'un réseau de neurones plutôt qu'une suite de procédures.

La tâche reste inchangée; elle consiste à mettre à jour le statut d'emploi de chaque travailleur au temps $t+1$, en désignant soit la firme dans laquelle il travaille ou en l'identifiant comme sans emplois, et ce à partir des informations disponibles au temps t.

Dans une perspective macroscopique, et afin de simplifier la tâche, nous nous contenterons de déterminer le nombre d'employés dans chaque firme, ainsi que le nombre de sans-emplois. Comme les travailleurs sont indiscernables les uns des autres au niveau macroscopique, nous pouvons répartir les travailleurs aléatoirement.

Sous cette simplification, la tâche consiste à prédire un vecteur $\boldsymbol{x} \in \mathbb{R}^{n_F+1}$ tel que $\sum_{i=0}^{n_F} x_i = W$, à partir de la répartition au temps précédent $\boldsymbol{x}_{prec}$ et de la matrice d'informations H.

1.7. Réseaux de *scattering*

1.7.1. Ondelettes de Morlet

Les ondelettes de Morlet sont un exemple typique de filtres utilisés dans la transformée de *scattering*. Elles sont définies par

$$\psi_{\sigma,\theta,\xi,\gamma}(u) = e^{-\|D_\gamma R_\theta(u)\|^2/(2\sigma^2)}(e^{i\xi u'} - \beta), \tag{1.7.1}$$

où β est une constante de normalisation pour s'assurer que les ondelettes s'intègrent à 0 sur le domaine spatial, $u' = u_1 \cos\theta + u_2 \sin\theta$, R_θ est la matrice de rotation de l'angle θ et $D_\gamma = \begin{pmatrix} 1 & 0 \\ 0 & \gamma \end{pmatrix}$. Les quatre paramètres peuvent être ajustés et sont présentés dans le tableau 1.1. Remarquons également que les ondelettes de Morlet sont des ondelettes à valeurs complexes. La banque de filtres d'ondelettes canonique est obtenue en dilatant une ondelette $\psi_{\sigma',\theta',\xi',\gamma'}(u)$ par des facteurs 2^j, $0 \leq j < J$, et en la tournant de L angles θ équidistants sur le cercle. On obtient la banque de filtres $\{2^{-2j}\psi_{\sigma',\theta',\xi',\gamma'}(2^{-j}R_\theta(u))\}$, qui est ensuite complétée par le filtre passe-bas ϕ_J. Cela peut être écrit en termes des paramètres présentés dans le tableau 1.1 comme $\psi_{2^j\sigma',\theta'-\theta,2^{-j}\xi',\gamma'}(u) = \psi(2^{-j}R_\theta(u))$. Par léger abus de notations, on dénote les filtres par ψ_λ, $\lambda = (\sigma_j, \theta, \xi_j, \gamma_j)$ pour désigner des ondelettes indexées par θ et j. L'ensemble de filtres résultant est visualisé dans le domaine fréquentiel à la figure 3.2 tandis que l'influence des paramètres est visualisé à la figure 1.13.

Param	Rôle	Param	Rôle
σ	Échelle de la fenêtre gaussienne	θ	Orientation globale
ξ	Échelle de la fréquence	γ	Rapport d'aspect

Tableau 1.1. Paramètres canoniques de l'ondelette de Morlet

1.7.2. Réseaux de *scattering*

Pour simplifier, nous nous concentrons sur les réseaux de *scattering* 2D jusqu'à leur 2ème ordre. Les ordres suivants peuvent être calculés en suivant le même schéma itératif, mais il a été démontré qu'ils produisent une énergie négligeable [**10**]. Étant donné un signal $x(u)$, où u est l'indice de position spatiale, nous calculons les coefficients de *scattering* S^0x, S^1x, S^2x, d'ordre 0, 1 et 2 respectivement. Pour un entier J, correspondant à l'échelle spatiale de la transformée de *scattering*, et en supposant un signal $N \times N$ à un canal, les cartes des attributs résultantes sont de taille $\frac{N}{2^J} \times \frac{N}{2^J}$, avec des tailles de canaux variant selon l'ordre des coefficients de *scattering* (c'est-à-dire, 1 canal à l'ordre 0, JL canaux à l'ordre 1 et $L^2J(J-1)/2$ canaux à l'ordre 2).

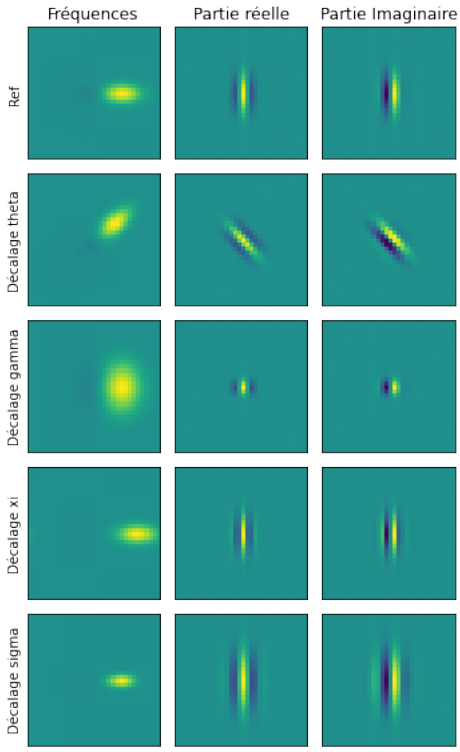


Fig. 1.13. Visualisation d'une ondelette de Morlet et de l'influence de ses paramètres. Une ondelette de référencce est visualisée, puis chaque paramètre est perturbé positivement.

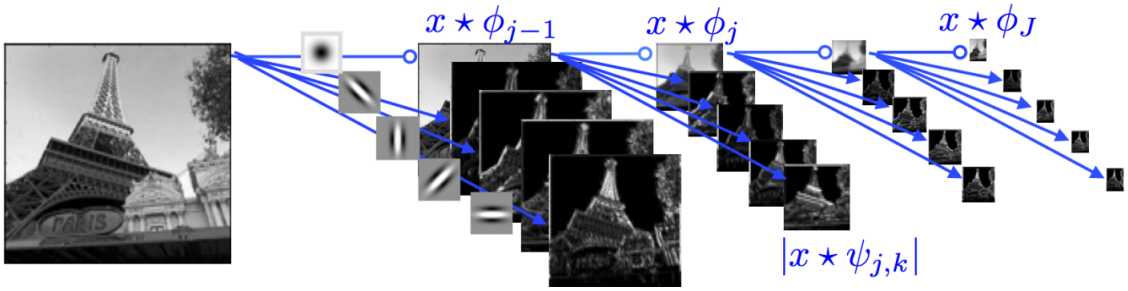


Fig. 1.14. Transformée en ondelettes d'une image $x(u)$, calculée à l'aide d'une cascade de convolutions avec des filtres sur J = 4 échelles et L = 4 orientations. Les filtres passe-bas ainsi que les filtres de bande passante L = 4 sont représentés sur les premières flèches. [**40**]

Pour calculer les coefficients d'ordre 0, on considère un filtre passe-bas ϕ_J avec une fenêtre spatiale d'échelle 2^J, par exemple une fonction de lissage gaussien. Nous convoluons ensuite ce filtre avec le signal et sous-échantillonnons par un facteur de 2^J pour obtenir $S^0x(u) = x * \phi_J(2^J u)$. Du fait du filtrage passe-bas, l'information haute fréquence est écartée et est seulement récupérée dans les coefficients d'ordre supérieur via la banque de filtres de Morlet.

Les coefficients de *scattering* de premier ordre sont calculés en convoluant d'abord le signal d'entrée avec les ondelettes complexes générées (c'est-à-dire indexées par les paramètres du tableau 1.1), puis en sous-échantillonnant le signal filtré résultant par le facteur d'échelle 2^{j_1} de l'ondelette choisie. Ensuite, la valeur absolue complexe est utilisé en chaque point pour ajouter de la non-linéarité, et le signal réel résultant est lissé via un filtre passe-bas. Enfin, une autre étape de sous-échantillonnage est appliquée, cette fois par un facteur de 2^{J-j_1}, pour obtenir une taille de sortie compressée de manière optimale. Mathématiquement, nous avons

$$S^1x(\lambda_1,u) = |x * \psi_{\lambda_1}| * \phi_J(2^J u). \tag{1.7.2}$$

La carte des attributs résultante a $J \cdot L$ canaux, basée sur le nombre d'ondelettes dans la famille générée et le nombre d'orientations. Les coefficients de second ordre sont générés de manière similaire, avec l'ajout d'une autre cascade de transformée en ondelettes et d'opérateur de valeur absolue avant le lissage passe-bas :

$$S^2x(\lambda_1,\lambda_2,u) = ||x * \psi_{\lambda_1}| * \psi_{\lambda_2}| * \phi_J(2^J u). \tag{1.7.3}$$

Du fait de l'interaction entre les bandes passantes et les supports fréquentiels de premier et second ordre, seuls les coefficients avec $j_1 < j_2$ ont une énergie significative. Par conséquent, la sortie de second ordre donne une carte des attributs avec $\frac{1}{2}J(J-1)L^2$ canaux.

Comme mentionné au début de cette section, il est possible d'étendre la définition à des ordres plus grands. Typiquement, et contrairement aux réseaux convolutifs traditionnels, les coefficients de tous les ordres sont utilisés en étant concaténés. La figure 1.14 présente une illustration d'un réseau de *scattering*.

1.7.3. Stabilité aux petites déformations lisses

Dans Mallat [**39**], il est montré que la transformée de *scattering* est stable par rapport à de petites déformations de la forme

$$D_\tau x(u) = x(u - \tau(u)), \tag{1.7.4}$$

où $x(u)$, est un signal et τ est un difféomorphisme. En effet, en supposant que τ est régulier et que x est à support compact, la transformée de *scattering* vérifie [**11**]

$$\|S(D_\tau x) - S(x)\| \leq C m_{\max} \|x\| (2^{-J}\|\tau\|_\infty + \|\nabla\tau\|_\infty), \tag{1.7.5}$$

où $S(\cdot)$ est la transformée de *scattering*, C est une constante, $m_{\max}$ est l'ordre maximal de la transformée de *scattering*, $\|x\| = \int |x(u)|^2\, du$, $\|\tau\|_\infty = \sup_u |\tau(u)|$ et $\|\nabla\tau\|_\infty = \sup_u \|\nabla\tau(u)\| < 1$. En pratique, si $2^J \geq \|\tau\|_\infty / \|\nabla\tau\|_\infty$, l'effet de la déformation est contrôlé uniquement par le terme impliquant $\|\nabla\tau\|_\infty$:

$$\|S(D_\tau x) - S(x)\| \leq C m_{\max} \|x\| \|\nabla\tau\|_\infty. \tag{1.7.6}$$

Cette stabilité est particulièrement importante, notamment dans les tâches de classifications d'images. En effet, si une image présente un léger flou, ou qu'un objet est légèrement décalé, cela ne devrait pas entraîner une mauvaise prédiction du modèle. Nous proposons une étude empirique de cette propriété à la section 3.1.3.

1.7.4. Réseaux hybrides

Les réseaux hybrides sont d'autres tentatives de pallier le problème des données limités. Pour ce faire, les réseaux hybrides combinent l'expressivité des filtres de Morlet avec l'adaptabilité de filtres libres en fusionnant un réseau de *scattering* avec un réseau de neurones convolutifs, tel le réseau résiduel large (WRN, de l'anglais *Wide-Residual Network*) [**47**]. Le réseau de *scattering*, à la base du réseau hybride, vient extraire des cartes des attributs intéressantes et réduit la dimension spatiale, le tout sans apprentissage (dans la version non paramétrée) ou avec peu d'apprentissage (dans la version paramétrée). De son côté, le WRN permet une classification efficace. Le tableau suivant illustre l'architecture. Au lieu d'un WRN, une simple couche linéaire peut être utilisée pour évaluer la séparabilité linéaire du réseau de *scattering*.

Stage	**Description**
scattering	Appris ou non appris
conv1	3×3, Couche Conv $128 \rightarrow 256$
conv2	$\begin{bmatrix} 3 \times 3, \text{Couche Conv } 256 \\ 3 \times 3, \text{Couche Conv } 256 \end{bmatrix} \times 4$
conv3	$\begin{bmatrix} 3 \times 3, \text{Couche Conv } 256 \\ 3 \times 3, \text{Couche Conv } 256 \end{bmatrix} \times 4$
avg-pool	*Avg pooling* à une taille 1x1

Tableau 1.2. Description de l'architecture hybride WRN utilisée pour les expériences de la section 3.1. Chaque couche convolutionnelle représente une convolution 2D suivie d'une normalisation selon le lot et d'une fonction de non-linéarité ReLU.

1.8. Modèles de diffusion

1.8.1. Vue d'ensemble

Les modèles de diffusion sont des modèles génératifs, c'est-à-dire qu'ils génèrent des sorties qui imitent les données d'entraînements. Par exemple, les modèles de diffusion peuvent être utilisés pour créer des images. Si les images d'entraînements sont des images de chats, les images produites par le modèle de diffusion en seront également.

Les modèles plus récents permettent de conditionner la génération de sorties par des entrées. Pour reprendre l'exemple précédent, ces entrées pourraient être un texte qui décrit une action, auquel cas l'image produite (en supposant que l'entraînement ait été efficace) serait une image de chat réalisant cette action. La figure 1.15 présente des exemples d'images générées à partir de conditionnement de texte issues du modèle de diffusion Imagen.

Fig. 1.15. Plusieurs images générées à partir de conditionnement de texte, issues du modèle de diffusion Imagen. Image tirée de [**52**].

Il est possible pour un utilisateur de générer plusieurs sorties distinctes à partir d'un unique modèle, et de choisir celle qui lui convient le plus. De plus, selon la méthode d'échantillonnage, les sorties peuvent être obtenues de façon déterministe ou non.

Pour comprendre le fonctionnement des modèles de diffusion, imaginons prendre un exemple de notre jeu de données et lui ajouter progressivement du bruit jusqu'à ce qu'il soit complètement indiscernable d'un bruit pur (de gauche à droite dans l'image 1.11). Les modèles de diffusion sont basés sur l'idée de reconstruire la sortie en inversant ce processus (de droite à gauche dans l'image 1.11). Pour cela, la tâche complexe de générer une sortie est découpée en un grand nombre d'étapes (de l'ordre de quelques centaines à quelques milliers). À chacune d'entre elles, le modèle cherche à enlever une petite quantité de bruit à la fois, jusqu'à obtenir une sortie qui appartient à la distribution des données d'entraînements. Un réseau de neurones (le même à chaque étape de la chaîne) est entraîné pour répondre à cette tâche.

1.8.2. Fonctionnement

Les modèles de diffusion sont des modèles à variables latentes basés sur une chaîne de Markov qui implémentent un processus de diffusion vers l'avant qui ajoute graduellement du bruit ainsi qu'un processus de diffusion vers l'arrière dans lequel le bruit est transformé progressivement vers une sortie qui pourrait être issue de la distribution des données d'entraînement.

Le processus de diffusion vers l'avant ajoute graduellement du bruit selon une séquence de variances $\beta_1, ..., \beta_T$:

$$q(\boldsymbol{x}_{1:T}|\boldsymbol{x}_0) := \prod_{i=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}), \quad q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) := \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}), \tag{1.8.1}$$

où $\boldsymbol{x}_0$ est l'entrée initiale et $\boldsymbol{x}_1$,...,$\boldsymbol{x}_T$ sont les variables latentes. Le nombre d'étapes T doit être choisi de sorte que le processus converge vers un bruit gaussien isotrope : $q(\boldsymbol{x}_T|\boldsymbol{x}_0) \approx \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Observons qu'en pratique, nous pouvons ajouter du bruit à $\boldsymbol{x}_{t-1}$ au lieu d'échantillonner une nouvelle variable aléatoire :

$$\boldsymbol{x}_t \sim \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}) \iff \boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}, \tag{1.8.2}$$

pour $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I})$.[1] On peut voir que si $\beta_t = 1$, alors il n'y aura que du bruit, tandis que si $\beta_t = 0$, alors seulement la variable $\boldsymbol{x}_{t-1}$ sera conservée. On peut donc interpréter β_t comme relié à la proportion de bruit ajouté.

De plus, il existe un paramétrage en forme close pour $\boldsymbol{x}_t$ depuis $\boldsymbol{x}_0$. En posant $\alpha_t = 1 - \beta_t$ et $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_t$, on obtient que

$$q(\boldsymbol{x}_T|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\boldsymbol{I}).^2 \tag{1.8.3}$$

Les modèles de diffusion implémentent également le processus inverse, paramétré par un réseau de neurones de paramètres θ :

$$p_\theta(\boldsymbol{x}_{0:T}) := p_\theta(\boldsymbol{x}_T) \prod_{i=1}^{T} p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t), \quad p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) := \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \boldsymbol{\Sigma}_\theta(\boldsymbol{x}_t, t)). \tag{1.8.4}$$

Il est typique d'utiliser un U-Net comme réseau de neurones, car il a la particularité de conserver la dimension des données. Le réseau de neurones peut admettre un conditionnement $\boldsymbol{c}$. Un conditionnement est une entrée du réseau de neurones dont on ne souhaite pas apprendre la distribution, contrairement à $\boldsymbol{x}$. Le conditionnement n'est pas bruité et peut être intégré au réseau d'un grand nombre de façons, à la discrétion du concepteur. Nous omettons d'intégrer le conditionnement à la notation afin d'éviter de la surcharger.

[1] L'équivalence est peut-être plus claire sous la forme $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \boldsymbol{I}) \iff \boldsymbol{X} = \boldsymbol{\mu} + \sigma \boldsymbol{Z}$, pour $\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$.
[2] Preuve à l'annexe A.1.

Bien qu'il soit possible de paramétriser la variance $\boldsymbol{\Sigma}_\theta(\boldsymbol{x}_t,t)$ à l'aide d'un réseau de neurones, plusieurs implémentations des modèles de diffusion fixent la variance à $\beta_t\boldsymbol{I}$, de sorte que $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) := \mathcal{N}(\boldsymbol{x}_{t-1};\boldsymbol{\mu}_\theta(\boldsymbol{x}_t,t),\beta_t\boldsymbol{I})$. Ce choix est adopté dans le présent travail. En guise d'exemple, un autre choix est de de considérer, pour une sortie $\boldsymbol{u}$ du réseau de neurones, l'interpolation suivante [**45**] :

$$\boldsymbol{\Sigma}_\theta(\boldsymbol{x}_t,t) = \exp(\boldsymbol{u}\log\beta_t + (1-\boldsymbol{u})\log\tilde{\beta}_t), \tag{1.8.5}$$

où $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$.

En pratique, le réseau de neurones est entraîné de sorte à minimiser une fonction de perte qui est usuellement choisie comme étant l'erreur quadratique moyenne (MSE, de l'anglais *Mean Squared Error*). Une justification théorique de ce choix est présentée à l'annexe A.2. L'algorithme d'apprentissage est présenté ci-dessous pour le paramétrage $\boldsymbol{\epsilon}$ (voir la sec. 1.8.3.1).

Algorithme 1.8.1. Apprentissage

Répéter

$\boldsymbol{x}_0 \sim q(\boldsymbol{x})$

$t \sim$ Uniforme($[1,2,\ldots,T]$)

$\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$

$\boldsymbol{x}_t = \sqrt{\bar{\alpha}}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}}\boldsymbol{\varepsilon}$

$L_\theta = \|\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\theta(\boldsymbol{x}_t,t)\|^2$

$\theta \leftarrow \theta - \gamma\nabla_\theta L_\theta$

Jusqu'à convergence

Pour générer des données, le processus d'échantillonnage consiste à reverser le processus afin de transformer progressivement un bruit en une donnée. Il est à noter que bien que nous ne pouvons pas échantillonner directement depuis $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, nous pouvons le faire depuis $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)$ en ajoutant l'information sur $\boldsymbol{x}_0$. En effet, nous avons[3]

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_{t-1};\tilde{\boldsymbol{\mu}}_t(\boldsymbol{x}_t,\boldsymbol{x}_0),\tilde{\beta}_t\boldsymbol{I}), \tag{1.8.6}$$

$$\text{où } \tilde{\boldsymbol{\mu}}_t(\boldsymbol{x}_t,\boldsymbol{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t. \tag{1.8.7}$$

Ainsi, en partant de $\boldsymbol{x}_t \sim \mathcal{N}(0,1)$, on peut échantillonner itérativement les données latentes à l'aide de l'équation 1.8.6. Cette façon d'échantillonner s'appelle l'échantillonnage ancestral et est décrit dans l'algorithme ci-dessous, où nous employons une fois de plus le paramétrage $\boldsymbol{\varepsilon}$ en guise d'exemple.

[3]Preuve à l'annexe A.3.

Algorithme 1.8.2. Échantillonnage ancestral

$\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$
Pour $t = T, \ldots, 1$:
  $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$ si $t > 1$, sinon $\boldsymbol{z} = \mathbf{0}$
  $\boldsymbol{x}_{t-1} = \frac{1}{\sqrt{\bar{\alpha}_t}}(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t,t)) + \sqrt{\beta_t}\boldsymbol{z}$
fin
Retourner $\boldsymbol{x}_0$

1.8.3. Techniques et autres considérations

L'étude des modèles de diffusion est un domaine de recherche actif. Ainsi, de nombreuses techniques émergent et sont continuellement raffinées. Cette section présente un aperçu des techniques et considérations utilisées dans ce mémoire, sans pour autant prétendre à une liste exhaustive.

1.8.3.1. Paramétrages.

Il est possible d'entraîner le réseau de neurones à prédire des quantités distinctes pour obtenir des propriétés spécifiques. En particulier, et de façon non-exhaustive, il est possible de :

- prédire la moyenne $\boldsymbol{\mu}$;
- prédire la sortie $\boldsymbol{x}$;
- prédire le bruit $\boldsymbol{\epsilon}$;
- prédire $\boldsymbol{v} := \sqrt{\bar{\alpha}_t}\boldsymbol{\epsilon} - \sqrt{1-\bar{\alpha}_t}\boldsymbol{x}$.

Le premier paramétrage est le plus direct. Il consiste à prédire la moyenne $\boldsymbol{\mu}$ de la gaussienne depuis laquelle on veut échantillonner la prochaine variable latente. Il n'est cependant pas beaucoup utilisé en pratique, car il performe moins bien que d'autres [**28**] en considérant l'objectif d'entraînement simplifié (justifié à l'annexe A.2).

Le paramétrage $\boldsymbol{x}$ est également très naturel. Il consiste à prédire la sortie directement à partir d'une variable latente bruitée : $\boldsymbol{x}_0 \approx \hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t,t)$. L'équation 1.8.6 est ensuite utilisée pour générer $\boldsymbol{x}_{t-1}$. Lui aussi n'est cependant que très peu utilisé en pratique, car la qualité des sorties est diminuée [**28**].

Le troisième paramétrage consiste à prédire le bruit $\boldsymbol{\epsilon}$ qui a été ajouté à l'entrée selon l'équation 1.8.3. La convention étant que $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$, nous pouvons calculer $\boldsymbol{\mu}_\theta(\boldsymbol{x}_t,t) = \frac{1}{\sqrt{\bar{\alpha}_t}}(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\hat{\boldsymbol{\epsilon}}_\theta(\boldsymbol{x}_t,t))$, où $\hat{\boldsymbol{\epsilon}}_\theta(\boldsymbol{x}_t,t)$ est une estimation de $\boldsymbol{\epsilon}_t$ paramétrée par le réseau de neurones. En effet, nous avons d'abord que

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_t; \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0, (1-\bar{\alpha}_t)\boldsymbol{I}) \tag{1.8.8}$$

$$\iff \boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\boldsymbol{\epsilon} \tag{1.8.9}$$

$$\iff \boldsymbol{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\boldsymbol{x}_t - \sqrt{(1-\bar{\alpha}_t)}\boldsymbol{\epsilon}). \tag{1.8.10}$$

Puis en combinant ceci avec 1.8.6, on trouve bien que

$$\tilde{\boldsymbol{\mu}}_t = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}(\boldsymbol{x}_t - \sqrt{(1-\bar{\alpha}_t)}\boldsymbol{\epsilon}) + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t \tag{1.8.11}$$

$$= \frac{1}{\sqrt{\bar{\alpha}_t}}(\boldsymbol{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t). \tag{1.8.12}$$

Ce paramétrage est couramment utilisé dans les modèles de diffusion. Avec ce paramétrage, plus l'image est bruitée, plus la fonction de perte va implicitement être diminuée. Cela conduit à ce que le modèle se calibre plus légèrement à partir d'exemples haut en bruit, ce qui est une propriété qui peut être très utile compte tenu que c'est ce type d'exemple qui risque de présenter les erreurs les plus grandes. Nous référons à la section 1.8.4.1 pour une discussion plus précise.

Un dernier paramétrage a été introduit dans [**54**] afin d'être utilisé dans les procédés de distillation. Le paramétrage $\boldsymbol{v}$ peut néanmoins être utilisé dans d'autres applications et présente des résultats équivalents à différents paramétrages, dont le paramétrage $\boldsymbol{\varepsilon}$. Grâce à $\boldsymbol{v}$, on peut déterminer $\boldsymbol{x_0} \approx \sqrt{\bar{\alpha}_t}\boldsymbol{x_t} - \sqrt{1-\bar{\alpha}_t}\boldsymbol{v}_\theta(\boldsymbol{x_t})$. Ce paramétrage présente le même comportement que le paramétrage $\boldsymbol{\varepsilon}$ en diminuant le poids des images davantage bruitées dans la fonction de perte, mais le fait de façon moins importante.

1.8.3.2. Choix de la variance.

Tel que discuté précédemment, la variance peut être apprise ou fixée. Deux choix sont couramment utilisés lorsqu'elle est fixée. Dans [**28**], β_t décroît linéairement de $\beta_1 = 10^{-4}$ à $\beta_T = 0.02$. Dhariwal et Nichol [**45**] propose plutôt de fixer β_t selon à partir d'une séquence de cosinus :

$$\beta_t = \text{clip}(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.999) \quad \bar{\alpha}_t = \frac{f(t)}{f(0)} \quad \text{où } f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2, \tag{1.8.13}$$

où s est fixé à 0.008 de sorte que le bruit proche de $t = 0$ soit non trivial et qu'il puisse ainsi être prédit. Une comparaison entre les deux choix est présentée à la figure 1.16.

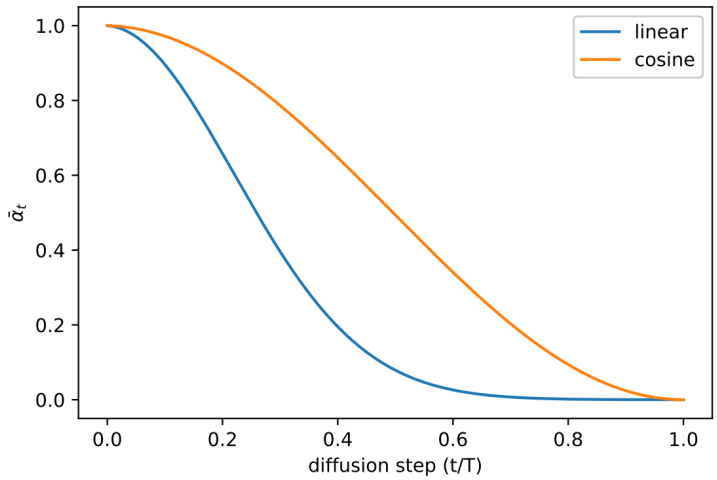


Fig. 1.16. Valeur de $\bar{\alpha}_t$ aux différentes étapes du modèle de diffusion. La séquence linéaire introduit le bruit plus agressivement au milieu de la chaîne que la séquence cosinus. Image tirée de [**45**].

1.8.3.3. Guidage sans classificateur.

Dhariwal et Nichol [**19**] ont proposé de renforcer le conditionnement en modifiant la prédiction à l'aide d'un second réseau de neurones. Cette approche de guidage avec classificateur permet d'obtenir des résultats plus fidèles au conditionnement, au coût d'entraîner un réseau supplémentaire.

La méthode de guidage sans classificateur [**29**] poursuit le même but, sans entraîner un réseau de neurones supplémentaire. Pour ce faire, elle introduit un poids de guidage $w \in \mathbb{R}^{\geq 0}$ qui permet de définir l'intensité de la technique. Pour générer un échantillon, il faut d'abord évaluer un modèle de diffusion avec conditionnement $\hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t,t,\boldsymbol{c})$, et le même modèle de diffusion sans conditionnement $\hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t,t)$. Le modèle est préalablement entraîné avec ces deux conditions conjointement en laissant tomber le conditionnement avec une probabilité donnée. Enfin, la nouvelle sortie $\hat{\boldsymbol{x}}^w_\theta(\boldsymbol{x}_t,t,\boldsymbol{c})$ est produite en considérant la droite reliant les deux évaluations, et en extrapolant dans la direction du conditionnement :

$$\hat{\boldsymbol{x}}^w_\theta(\boldsymbol{x}_t,t,\boldsymbol{c}) = (1+w)\hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t,t,\boldsymbol{c}) - w\hat{\boldsymbol{x}}_\theta(\boldsymbol{x}_t,t). \tag{1.8.14}$$

La technique peut être employée peu importe le paramétrage en modifiant en conséquence l'équation 1.8.14. Une propriété importante du guidage sans classificateur est qu'il produit un échange entre la diversité des sorties et leur fidélité par rapport au conditionnement [**52**].

1.8.3.4. Seuils.

L'application d'un seuil à chaque étape de la chaîne de diffusion permet de limiter une haute saturation des données générées. Cela est d'autant plus vrai lorsque le guidage sans classificateur est appliqué, car ce dernier à tendance à sortir les prédictions de leur plage de valeurs (les données sont normalisées dans $[-1,1]$). Ainsi, le seuillage permet d'éviter une erreur de mauvaise concordance entre les données d'entraînement et de test [**52**].

Différents seuils peuvent être appliqués. Par exemple, on peut restreindre le vecteur de données $\boldsymbol{x}$ de sorte que pour $s \geq 1$, x_i soit dans $[-s,s]$ (si une valeur est hors de l'intervalle, elle est modifiée à s ou $-s$ selon si elle est plus grande ou plus petite que l'intervalle), puis diviser chaque composante par s. En choisissant $s = 1$, cela revient à faire une coupure simple. Imagen [**52**] propose de choisir s à un certain centile de la valeur absolue des composantes de $\boldsymbol{x}$, ce qui permet limiter davantage la saturation en poussant les composantes vers l'intérieur de l'intervalle. Nous employons cette stratégie en fixant $s = \max_i(|x_i|)$, ce qui correspond au 100^{e} centile.

1.8.4. Variantes et recherche active

Dans cette section, nous discutons de développements récents des modèles de diffusion. Ces développements n'ont pas été implémentés dans les expérimentations de la section 3.2, mais restent pertinents à être discutés. En effet, ils simplifient la présentation, présentent une nouvelle approche d'échantillonnage nommée par son sigle DDIM (de l'anglais *Denoising Diffusion Implicit Models* pour modèles implicites de diffusion de débruitage) et permettent de grandement accélérer l'échantillonnage à l'aide de la technique de distillation.

1.8.4.1. Reformulation.

Nous présentons une reformulation qui simplifie la présentation et donne des outils pour améliorer les modèles de diffusion [**34, 52, 54**]. Nous considérons un cas de modèle de diffusion simple où nous avons des données $\{\boldsymbol{x}\}$ et nous cherchons à apprendre la distribution marginale $p(\boldsymbol{x})$. En particulier, nous omettons une fois de plus d'inclure le conditionnement, bien qu'il puisse facilement être ajouté.

La première différence avec le modèle décrit à la section 1.8 est que la chaîne de diffusion n'est plus fixée. Ainsi, au lieu de considérer une variable discrète $t \in \{0,1,\ldots,T\}$, nous considérons une variable continue $t \in [0,1]$, où $t = 0$ correspond à aucun bruit et $t = 1$ correspond au maximum de bruit. Nous dénotons les variables latentes bruitées par $\boldsymbol{z}_t$.

La deuxième différence est qu'au lieu de paramétrer les pas de diffusion $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$, nous paramétrons $q(\boldsymbol{z}_t|\boldsymbol{x})$ directement, ce que nous savons être possible par la formulation antérieure. Pour cela, il faut spécifier deux séquences α_t et σ_t (des fonctions positives et

différentiables en t) telles que le logarithme du ratio signal-à-bruit $\lambda_t := \log(\frac{\alpha_t^2}{\sigma_t^2})$ décroisse avec t. La distribution conditionnelle des variables latentes étant donné $\boldsymbol{x}$ est alors donnée par

$$q(\boldsymbol{z}_t|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}_t; \alpha_t \boldsymbol{x}, \sigma_t \boldsymbol{I}). \tag{1.8.15}$$

Nous pouvons interpréter α_t comme la quantité de signal retenu et σ_t comme la force du bruit ajouté. Exiger que λ_t décroisse, c'est demander que $\boldsymbol{z}_t$ soit plus bruitée pour des valeurs de t plus grande. Les modèles standards de diffusion sont choisis de sorte à préserver la variance : $\sigma_t^2 = 1 - \alpha_t^2$. Un choix commun est de prendre $\alpha_t = \cos(0.5\pi t)$ tel que discuté à la section 1.8.3.2. À partir de l'équation 1.8.15 et pour $0 \leq s \leq t \leq 1$, nous pouvons décrire la distribution conditionnelle d'un pas :

$$q(\boldsymbol{z}_t|\boldsymbol{z}_s) = \mathcal{N}(\boldsymbol{z}_t; (\alpha_t/\alpha_s)\boldsymbol{z}_s, \sigma_{t|s}^2 \boldsymbol{I}), \tag{1.8.16}$$

où $\sigma_{t|s}^2 = (1 - e^{\lambda_t - \lambda_s})\sigma_t^2$. Le but du modèle de diffusion est de débruiter $\boldsymbol{z}_t \sim q(\boldsymbol{z}_t|\boldsymbol{x})$ dans une estimation $\hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t, \lambda_t) = \hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t) \approx \boldsymbol{x}$ (la dépendance à λ_t est omise pour simplifier la notation). Remarquons comme troisième différence que c'est maintenant le logarithme du ratio signal-à-bruit qui est passé au modèle, au lieu de l'indice t. Le modèle $\hat{\boldsymbol{x}}_\theta$ est entraîné pour minimiser l'erreur moyenne au carré pondérée

$$\mathbb{E}_{t\sim U[0,1], \boldsymbol{x}\sim p(\boldsymbol{x}), \boldsymbol{z}_t \sim q(\boldsymbol{z}_t|\boldsymbol{x})}[w(\lambda_t)\|\hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t) - \boldsymbol{x}\|_2^2]. \tag{1.8.17}$$

La fonction de poids $w(\lambda_t)$ est fixée et vient généralement donner plus d'importance à de grandes valeurs de λ_t, car il est fort probable qu'une prédiction issue d'une variable $\boldsymbol{z}_t$ très bruitée soit de mauvaise qualité. Dans le cadre des modèles distillés présentés à la section 1.8.4.3, toutefois, on veut attribuer un poids plus uniforme. En guise d'exemple de poids, le paramétrage $\boldsymbol{\varepsilon}$ du modèle vient implicitement définir $w(\lambda_t) = e^{\lambda_t} = \frac{\alpha_t^2}{\sigma_t^2}$, où l'on voit qu'un poids proche de 0 est appliqué aux images peu bruitées. Le paramétrage $\boldsymbol{v}$ définit implicitement de son côté $w(\lambda_t) = 1 + e^{\lambda_t} = 1 + \frac{\alpha_t^2}{\sigma_t^2}$, où l'on voit que les poids restent proche de 1 pour les images peu bruitées.

Le processus peut être renversé pour définir l'échantillonnage ancestral. Nous avons pour $s < t$ que

$$q(\boldsymbol{z}_s|\boldsymbol{z}_t,\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}_s; \tilde{\boldsymbol{\mu}}_{s|t}, \sigma_{s|t}^2 \boldsymbol{I}), \tag{1.8.18}$$

où $\tilde{\boldsymbol{\mu}}_{s|t}$ est une combinaison convexe de $\boldsymbol{z_t}$ et $\boldsymbol{x}$ à intensité s:

$$\tilde{\boldsymbol{\mu}}_{s|t} = e^{\lambda_t - \lambda_s}(\alpha_s/\alpha_t)\boldsymbol{z}_t + (1 - e^{\lambda_t - \lambda_s})\alpha_s \boldsymbol{x}. \tag{1.8.19}$$

En partant de $\boldsymbol{z}_1 = \mathcal{N}(0,1)$, l'échantillonnage ancestral suit la règle modifiée

$$\boldsymbol{z}_s = \tilde{\boldsymbol{\mu}}_{s|t}(\boldsymbol{z}_t, \hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t)) + \sqrt{(\sigma^2_{s|t})^{1-\gamma} + (\sigma^2_{t|s})^\gamma}\boldsymbol{\varepsilon} \tag{1.8.20}$$

$$= e^{\lambda_t - \lambda_s}(\alpha_s/\alpha_t)\boldsymbol{z}_t + (1 - e^{\lambda_t - \lambda_s})\alpha_s \hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t) + \sqrt{(\sigma^2_{s|t})^{1-\gamma} + (\sigma^2_{t|s})^\gamma}\boldsymbol{\varepsilon}, \tag{1.8.21}$$

où γ est un hyperparamètre qui contrôle la quantité de bruit ajouté.

1.8.4.2. DDIM.

Cette sous-section présente une autre méthode d'échantillonnage appelée DDIM [**61**]. Contrairement à l'échantillonnage ancestral, l'échantillonnage DDIM est déterministe. Il est aussi plus intuitif : au lieu de chercher à échantillonner $\boldsymbol{z}_s$ à partir d'une gaussienne, l'échantillonnage DDIM procède comme suit. L'estimation $\hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t)$ est calculée, de même que le bruit correspondant $(\boldsymbol{z}_t - \alpha_t \hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t))/\sigma_t$ isolé à partir de l'équation 1.8.15. Ensuite, $\boldsymbol{z}_s$ est produite en les combinant à une intensité s:

$$\boldsymbol{z}_s = \alpha_s \hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t) + \sigma_s \frac{\boldsymbol{z}_t - \alpha_t \hat{\boldsymbol{x}}_\theta(\boldsymbol{z}_t)}{\sigma_t}. \tag{1.8.22}$$

On voit qu'aucune variable n'est tirée d'une distribution, comme l'était le bruit $\boldsymbol{\varepsilon}$ dans l'échantillonnage ancestral.

1.8.4.3. Distillation.

Un des principaux reproche fait aux modèles de diffusion est leur long temps d'échantillonnage. En effet, produire un échantillon requiert généralement d'évaluer le réseau de neurones près d'un millier de fois. La technique de distillation permet d'accélérer l'échantillonnage des modèles de diffusion, en réduisant considérablement le nombre de pas d'échantillonnage, tout en conservant la même qualité d'image générée [**42, 54**]. En contrepartie, le temps d'entraînement est augmenté, considérant la procédure décrite ci-après. Tout d'abord, un modèle est entraîné, comme décrit à la section précédente. Ensuite, un deuxième modèle est initialisé en étant une copie du précédent modèle. Le premier modèle sera le modèle professeur et le second le modèle élève. Supposons que le modèle professeur obtienne de bons résultats en N pas. Le modèle élève cherchera à répliquer les résultats du modèle professeur en $N/2$ pas. L'objectif du modèle élève est modifié : il cherche à prédire $\tilde{\boldsymbol{x}}$ qui est choisi de sorte qu'une unique application de DDIM de l'élève corresponde à l'application de deux pas de DDIM du professeur (voir la fig. 1.17). Pour des pas successifs t, t', t'' de DDIM, la valeur cible $\tilde{\boldsymbol{x}}$ doit être de [**54**]

$$\tilde{\boldsymbol{x}} = \frac{\boldsymbol{z}_{t''} - \frac{\sigma_{t''}}{\sigma_t}\boldsymbol{z}_t}{\alpha_{t''} - \frac{\sigma_{t''}}{\sigma_t}\alpha_t}. \tag{1.8.23}$$

Une fois le modèle élève entraîné, il peut devenir le modèle professeur, et un autre modèle élève peut être initialisé, et ainsi de suite jusqu'à avoir le nombre de pas désiré.

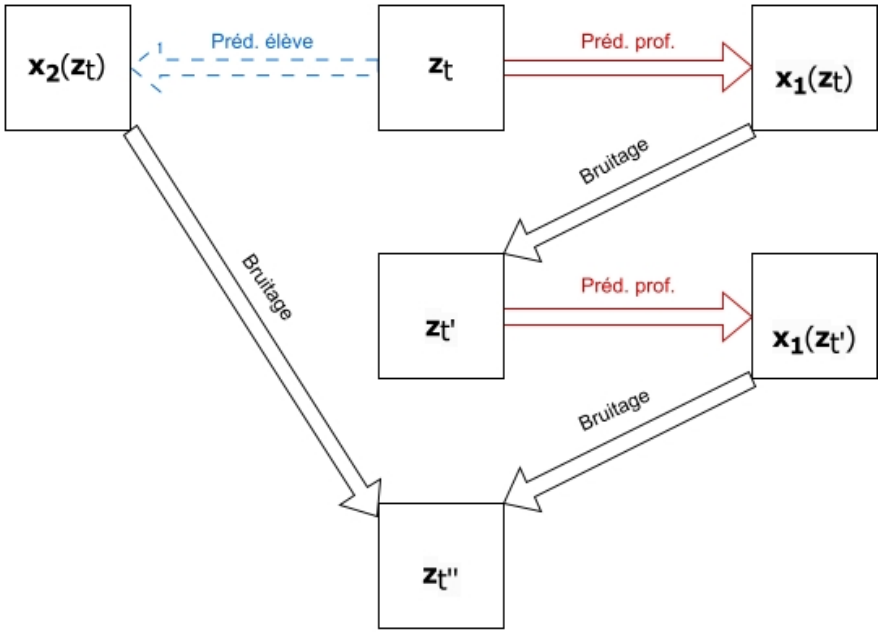


Fig. 1.17. Illustration du processus de distillation. Le modèle élève $\boldsymbol{x}_2$ est entraîné à reproduire deux étapes du modèle professeur $\boldsymbol{x}_1$ en prédisant $\boldsymbol{x}_2(\boldsymbol{z}_t) \approx \tilde{\boldsymbol{x}}$.

Cette technique peut aussi être employée afin d'encapsuler le guidage sans classificateur [**42**]. Similairement à ce qui précède, le modèle élève cherche à répliquer les résultats du modèle professeur, une fois le guidage sans classificateur appliqué. Pour cela, le modèle se voit conditionner par rapport au poids de guidage. Il faut faire cette distillation avant de réduire le nombre de pas du modèle.

Chapitre 2

Adaptations et améliorations des modèles

2.1. Réseaux de *scattering* paramétré

2.1.1. Apprentissage paramétré

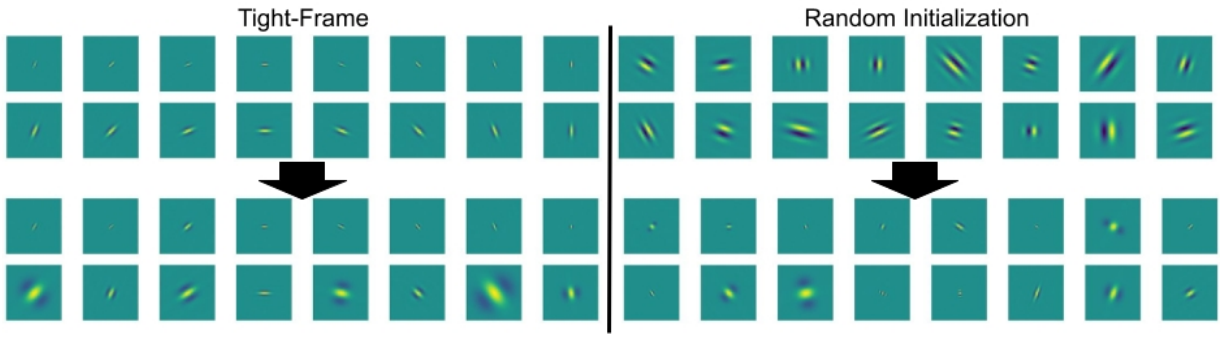


Fig. 2.1. Filtres d'ondelettes avant et après l'entraînement. Partie réelle des filtres d'ondelettes de Morlet initialisés en repère ajusté (à gauche) et aléatoirement (à droite) avant (en haut) et après (en bas) entraînement. Les filtres ont été optimisés sur l'ensemble de l'ensemble d'apprentissage de CIFAR-10. Nous utilisons le paramétrage canonique des ondelettes de Morlet. Pour les filtres initialisés en repère ajusté, nous observons des changements substantiels à la fois dans l'échelle et le rapport d'aspect. D'autre part, tous les filtres aléatoires subissent des changements majeurs d'orientation et d'échelle. Nous référons à la section 3.1.4 pour le protocole exact et les résultats détaillés.

Nous avons vu d'une part que l'apprentissage automatique permet d'effectuer des tâches complexes, mais que d'autre part, le contexte des données limités rend difficile l'apprentissage. Nous proposons un entre deux entre un apprentissage très libre (CNN) et un modèle sans apprentissage (réseau de *scattering*). En effet, nous proposons d'apprendre non pas les filtres du réseau de *scattering* (ce qui serait équivalent à utiliser un CNN), mais uniquement les paramètres (présentés dans le tableau 1.1) qui sont utilisés pour générer les filtres.

2.1.1.1. Rétropropagation.

Nous montrons dans cette section qu'il est possible de rétropropager le gradient à travers cette construction paramétrique. À savoir, nous vérifions la dérivabilité de cette construction en calculant explicitement les dérivées partielles par rapport à ses paramètres. Premièrement, la dérivée $\mathbb{R}$-linéaire du module complexe $f(z) = |z|$ est $f'(z) = \frac{z}{|z|}$. Ensuite, nous montrons la différenciation de la convolution avec des ondelettes par rapport à leurs paramètres. Pour plus de simplicité, nous nous concentrons ici sur la différenciation de la portion de Gabor[1] des filtres (c.-à-d. l'ondelette de Morlet moins le terme normalisant) à partir de l'équation 1.7.1, que l'on peut développer comme suit :

$$\begin{aligned}\varphi(u) = &\exp(-\frac{1}{2\sigma^2}(u_1^2(\cos^2(\theta) + \sin^2(\theta)\gamma^2) + u_2^2(\cos^2(\theta)\gamma^2 + \sin^2(\theta)) \\ &+ 2\cos(\theta)\sin(\theta)u_1u_2(1-\gamma^2)) + i\xi(\cos(\theta)u_1 + \sin(\theta)u_2)). \end{aligned} \tag{4}$$

Ses dérivées partielles sont :

$$\begin{aligned}\frac{\partial\varphi}{\partial\theta}(u) &= \frac{1}{\sigma^2}(u_2\cos\theta - u_1\sin\theta)(i\xi\sigma^2 + u_1(\gamma^2-1)\cos\theta \\ &\quad + u_2(\gamma^2-1)\sin\theta)\varphi(u); \end{aligned} \tag{5}$$

$$\begin{aligned}\frac{\partial\varphi}{\partial\sigma}(u) &= \frac{1}{\sigma^3}(u_1^2(\cos^2\theta + \gamma^2\sin^2\theta) + u_2^2(\gamma^2\cos^2\theta + \sin^2\theta) \\ &\quad + 2u_1u_2\cos\theta\sin\theta(1-\gamma^2))\varphi(u); \end{aligned} \tag{6}$$

$$\frac{\partial\varphi}{\partial\xi}(u) = i(u_1\cos\theta + u_2\sin\theta)\varphi(u);\ \text{et} \tag{7}$$

$$\begin{aligned}\frac{\partial\varphi}{\partial\gamma}(u) &= -\frac{1}{\sigma^2}(u_1^2\gamma\sin^2\theta + u_2^2\gamma\cos^2\theta \\ &\quad - 2u_1u_2\gamma\cos\theta\sin\theta)\varphi(u). \end{aligned} \tag{8}$$

Enfin, la dérivée de la convolution avec de tels filtres est donnée par $\frac{\partial}{\partial\zeta}(f * \varphi)(t) = \int f(t-u)\frac{\partial\varphi}{\partial\zeta}(u)du$ où ζ est l'un des paramètres de filtre du tableau 1.1. Il est facile de vérifier que ces dérivées peuvent être enchaînées pour se propager à travers les cascades de *scattering* définies dans la section 1.7.2. Nous pouvons donc apprendre les paramètres optimaux conjointement avec d'autres paramètres (si voulu) dans une architecture différentiable de bout en bout. En pratique, les modules de différentiation automatique sont utilisés, et la discussion précédente permait d'assurer leur stabilité.

[1]Il n'est pas difficile d'étendre cette dérivation aux ondelettes de Morlet, mais les expressions résultantes sont plutôt lourdes et laissées de côté par souci de brièveté.

2.1.1.2. Initialisation.

Nous considérons deux initialisations pour les filtres : la construction canonique et une initialisation aléatoire, lesquelles sont visualisées avant et après l'apprentissage à la figure 2.1. La construction canonique suit les implémentations courantes de la transformée de *scattering* en définissant $\sigma_{j,\ell} = 0{,}8 \times 2^j$, $\xi_{j,\ell} = \frac{3\pi}{4} 2^{-j}$, et $\gamma_{j,\ell} = \frac{4}{L}$ pour $j = 1, \ldots, J$, $\ell = 1, \ldots, L$, tandis que pour chaque j, nous fixons $\theta_{j,\ell}$ pour qu'il soit également espacé sur $[0,\pi)$. La construction garantit que la banque de filtres résultante forme un repère ajusté (TF, de l'anglais *Tight-Frame*). Nous appelons conséquemment cette construction l'initialisation en repère ajusté.

Dans l'initialisation aléatoire, les paramètres sont échantillonnés comme $\sigma_{j,\ell} \sim \log(U[e, e^5])$, $\xi_{j,\ell} \sim U[0.5,1]$, $\gamma_{j,\ell} \sim U[0.5,1.5]$, et $\theta_{j,\ell} \sim U[0, 2\pi]$. En mots, les orientations sont sélectionnées uniformément au hasard sur le cercle, la largeur du filtre σ est sélectionnée en utilisant une distribution exponentielle sur les échelles disponibles et la fréquence spatiale ξ est choisie pour être dans l'intervalle $[0.5, 1]$, qui se situe au centre de la plage possible entre l'*aliasing* ($> \pi$) et la fréquence fondamentale de la taille du signal ($2\pi/N$ où N est le nombre de pixels). Enfin, nous sélectionnons le rapport d'aspect γ pour qu'il soit centré sur le cas sphérique de 1.0, avec une variation entre une plus forte sélectivité d'orientation (0.5) et une plus petite sélectivité d'orientation (1.5).

2.1.1.3. Paramétrages.

Nous considérons deux paramétrages distincts. Le premier, que nous appelons paramétrage canonique, est celui qui a été implicitement utilisé jusqu'à présent. Dans ce dernier, chaque paramètre est indépendant les uns des autres.

Le second, que nous appelons paramétrage équivariant, crée une interdépendance entre les filtres d'une échelle donnée. Dans ce dernier, pour chaque échelle $j \leq J$, un unique groupe de paramètres λ est appris, et les L filtres partagent les mêmes paramètres, à l'exception de leur orientation qui fixée à $[\theta, \theta + \frac{\pi}{L}, \ldots, \theta + \frac{(L-1)\pi}{L}]$. Le repère ajusté décrit à la section précédente est ainsi équivariant par construction. Imposer un paramétrage équivarient réduit le nombre de paramètres appris d'un facteur L et garantit une couverture plus uniforme des ondelettes.

2.1.2. Distance entre banques de filtres

Nous souhaitons pouvoir comparer quantitativement deux banques de filtres de Morlet. Un tel outil permettrait de répondre plus précisément à des questions comme « À quel point la banque de filtres évolue-t-elle durant l'apprentissage? » ou encore « Est-ce que les filtres appris à partir d'une initialisation aléatoire convergent vers le repère ajusté standard? ». Nous définissons donc une distance entre banques de filtres afin d'effectuer de telles comparaisons.

Nous évaluons la distances entre deux ondelettes de Morlet individuelles comme suit :

$$\Upsilon(M_1,M_2) = \left\|(\sigma_1,\xi_1,\gamma_1)^T - (\sigma_2,\xi_2,\gamma_2)^T\right\|_2 + \text{arcdist}(\theta_1,\theta_2), \tag{2.1.1}$$

où $M_i = (\sigma_i,\xi_i,\gamma_i,\theta_i)^T$ désigne le paramétrage d'une ondelette de Morlet. Nous utilisons la distance de l'arc sur le cercle unitaire pour comparer les valeurs de thêta. Étant donné que l'ensemble de filtres appris n'a pas d'ordre canonique, pour comparer un réseau de *scattering* appris au réseau de *scattering* à repère ajusté, nous utilisons un algorithme d'appariement pour faire correspondre un ensemble de filtres à l'autre. Plus précisément, nous calculons d'abord Υ entre toutes les combinaisons de paires de filtres des deux réseaux, puis utilisons un algorithme d'appariement bipartite à coût minimum [**36**] pour trouver la correspondance de distance minimale entre les deux ensembles de filtres. La distance finale que nous utilisons comme notion de similarité entre deux réseaux de *scattering* est la somme de Υ pour toutes les paires appariées dans le graphe bipartite. Dorénavant, nous appellerons cette distance la *distance de la banque de filtres*.

2.2. Modèle de diffusion

2.2.1. Réseaux de neurones

2.2.1.1. U-Net pour diffusion.

Nous décrivons les principales modifications faites au U-Net (voir la sec. 1.3.1) dans le cadre de leur utilisation dans nos modèles de diffusion. Le premier changement consiste à ajouter une couche de *padding* aux couches de convolution. Le but de ce changement est de retrouver la même taille des données à la fin du U-Net qu'au début.

Le deuxième changement est plus substantiel. Il consiste à ajouter des couches d'attention dans les différents blocs du U-Net [**46**]. Plusieurs implémentations existent; nous décrivons ci-dessous celle que nous avons utilisée dans les expériences de la section 3.2.

Nous utilisons l'attention croisée pour intégrer le conditionnement [**48, 52**]. La couche d'attention est appliquée entre les convolutions et le *pooling*, dans chaque bloc autant descendant que montant. Pour ce faire, nous concaténons le conditionnement $\boldsymbol{c}$ aux attributs entrants dans les paires clés-valeurs.

2.2.1.2. Réseau personnalisé.

Nous décrivons le réseau conçu dans le cadre de la modélisation du marché de l'emploi, dont le problème est décrit à la section 1.6.3.

Nous commençons par introduire deux sous-réseaux : le réseau résiduel fermé (GRN, de l'anglais *Gated Residual Network*) et le réseau de sélection de variables (VSN, de l'anglais *Variable Selection Network*) [**37**].

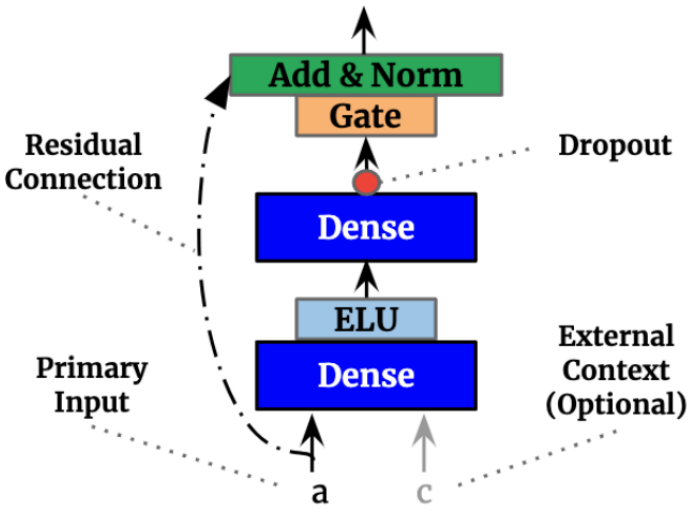


Fig. 2.2. Illustration du réseau résiduel fermé. Image tirée de [**37**].

Le GRN (voir fig. 2.2) est similaire à des couches linéaires standards à deux nuances près. Premièrement, il intègre la possibilité d'un conditionnement $\boldsymbol{c}$. Deuxièmement, il permet de facilement bloquer des parties d'information grâce à son mécanisme de porte. Il est décrit par les équations suivantes :

$$\mathrm{GRN}(\boldsymbol{x}, \boldsymbol{c}) = \mathrm{LayerNorm}(\boldsymbol{x} + \mathrm{GLU}(\boldsymbol{\eta}_1)), \text{ où} \tag{2.2.1}$$

$$\boldsymbol{\eta}_1 = W_1\boldsymbol{\eta}_2 + \boldsymbol{b}_1, \text{ et} \tag{2.2.2}$$

$$\boldsymbol{\eta}_2 = \mathrm{ELU}(W_2\boldsymbol{x} + W_3\boldsymbol{c} + \boldsymbol{b}_2); \tag{2.2.3}$$

avec

$$\mathrm{GLU}(\boldsymbol{\gamma}) = \sigma(W_4\boldsymbol{\gamma} + \boldsymbol{b}_4) \odot (W_5\boldsymbol{\gamma} + \boldsymbol{b}_5). \tag{2.2.4}$$

Le réseau de sélection de variables (voir fig. 2.3) permet de sélectionner et de combiner plusieurs entrées (ou plus précisément, une représentation des entrées). Étant donné des entrées ξ_i, on peut considérer la concaténation aplatie Ξ (c'est-à-dire qu'on considère les entrées mises bout à bout afin de former un grand vecteur). Ensuite, on fait passer chaque entrée ξ_i ainsi que Ξ dans leur propre GRN. On choisit le GRN appliqué à Ξ de sorte que la dimension de la sortie soit égale au nombre d'entrées. On peut alors prendre un SOFTMAX,

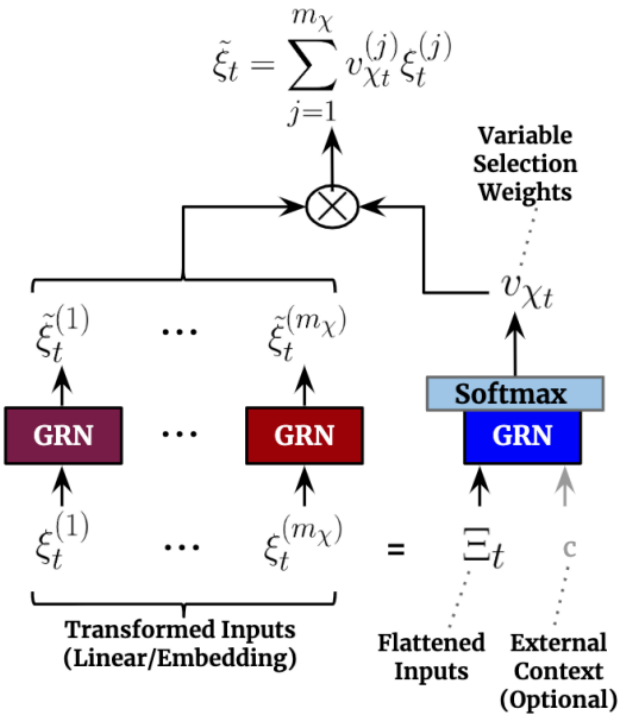


Fig. 2.3. Illustration du réseau de sélection de variables. Image tirée de [**37**].

pour obtenir des poids qui serviront à pondérer les sorties des GRN appliqués aux ξ_i avant de les additionner. Le réseau de sélection de variables est décrit par les équations suivantes :

$$\tilde{\xi} = \sum_{j=1}^{n} v_j \tilde{\xi}_j, \text{ où} \tag{2.2.5}$$

$$\tilde{\xi}_j = \text{GRN}(\xi_i), \text{ et} \tag{2.2.6}$$

$$v = \text{SOFTMAX}(\text{GRN}(\Xi)). \tag{2.2.7}$$

Nous pouvons maintenant définir notre réseau de neurones personnalisé (voir fig. 2.4). Il se structure en deux parties. La première s'occupe d'encoder l'entrée $\boldsymbol{x}_{prec}$, tandis que la seconde débruite la variable latente $\boldsymbol{x}_t$ associée au vecteur $\boldsymbol{x}$.

L'encodeur consiste en une série de blocs. Chaque bloc est constitué d'une couche d'attention croisée, puis d'un GRN. La couche d'attention croisée a comme requête la sortie du bloc précédent, ou, au début du réseau, l'entrée initiale, et comme clé et valeur la concaténation de $\boldsymbol{x}_{prec}$ avec H, que l'on note $\boldsymbol{c}$. Le GRN a comme entrée la sortie de la couche d'attention croisée, et est conditionné par la sortie d'un réseau de sélection de variables appliqué à $\boldsymbol{c}$. Mathématiquement,

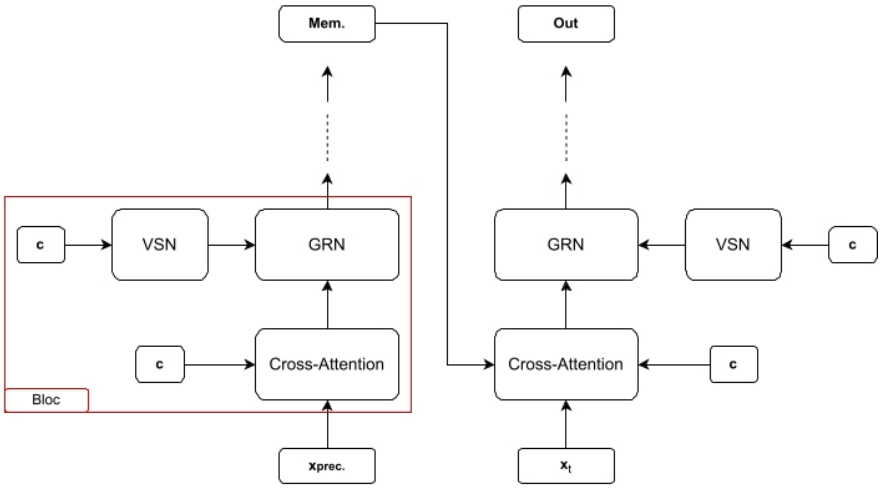


Fig. 2.4. Illustration du modèle personnalisé. Un exemple de bloc est identifié en rouge.

$$\text{Bloc}(\boldsymbol{x},\boldsymbol{c},t) = \text{GRN}(\boldsymbol{y},\boldsymbol{d},t), \text{ où} \tag{2.2.8}$$

$$\boldsymbol{y} = \text{CrossAttention}(\boldsymbol{x},\boldsymbol{c}), \text{ et} \tag{2.2.9}$$

$$\boldsymbol{d} = \text{VariableSelectionNetwork}(\boldsymbol{c}). \tag{2.2.10}$$

Le décodeur cherche à débruiter $\boldsymbol{x}_t$. Il est passé dans une chaine similaire de blocs, chacun encore constitué de Cross-Attention et de GRN. On rajoute toutefois au conditionnement la sortie de l'encodeur par concaténation.

2.2.2. Interpolation

Dans le modèle *Stable Diffusion* [**48**], il est proposé d'effectuer la diffusion dans un espace latent plutôt que de la faire directement sur les données. Pour ce faire, ils utilisent un encodeur et un décodeur de part et d'autre de la chaîne de diffusion. Néanmoins, dans le cadre de la tâche de modélisation du modèle CATS, cette démarche est excessive (il faut générer un vecteur de dimension 121 par comparaison à des images de plusieurs centaines de millier de pixels). Nous proposons plutôt d'interpoler le vecteur d'entrée et de sortie afin de modifier les dimensions à une dimension intermédiaire. Dans le cadre du contexte de série temporelle limitée, cela a l'avantage de modifier la dimension du modèle à notre guise (notamment pour utiliser efficacement l'attention à plusieurs têtes décrite à la section 2.2.1.1), sans pour autant tomber dans le piège d'extraire des mauvaises représentations des données (comme le ferait une simple projection linéaire) ni de complexifier déraisonnablement le modèle et possiblement tomber dans le piège du surajustement (comme le ferait un encodeur-décodeur).

Chapitre 3

Expérimentations

3.1. Réseau de *scattering* paramétré sur des jeux de données limités

3.1.1. Jeux de données

Nos évaluations empiriques pour les réseaux de *scattering* paramétrés sont basées sur trois jeux de données d'images, illustrés à la figure 3.1. CIFAR-10 et KTH-TIPS2 sont respectivement des jeux de données de reconnaissance d'images naturelles et de textures. COVIDx CRX-2 est un jeu de données de radiographies pour le diagnostic de COVID-19; son utilisation démontre la viabilité de notre approche dans la pratique, notamment dans les applications d'imagerie médicale.

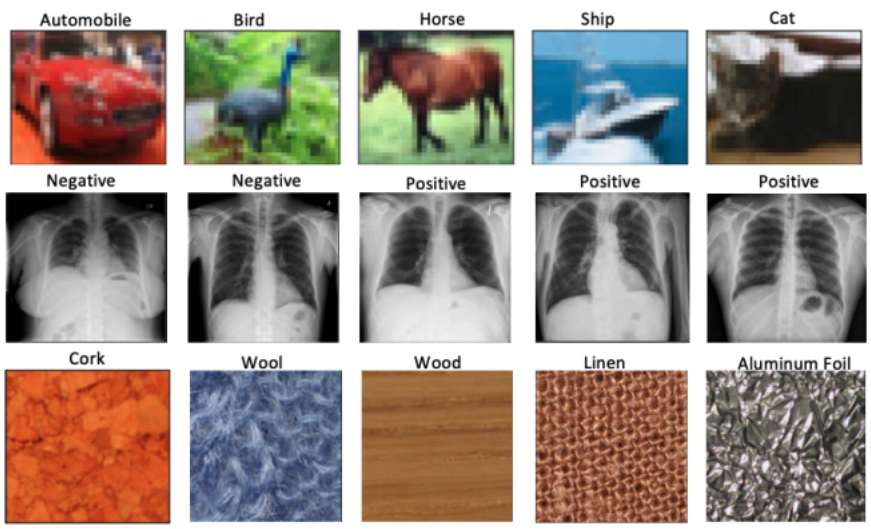


Fig. 3.1. Exemple d'images appartenant aux différents jeux de données. (En haut) CIFAR-10. (Au milieu) COVIDx CRX-2.(En bas) KTH-TIPS2.

CIFAR-10 se compose de 60 000 images de taille $3 \times 32 \times 32$ réparties en dix classes. L'ensemble d'entraînement contient 50 000 échantillons équilibrés en classe, tandis que l'ensemble de test contient les images restantes.

COVIDx CRX-2 est un ensemble de données d'images de radiographie pulmonaire de patients COVID-19 de taille $1 \times 1024 \times 1024$ réparties en deux classes (positives et négatives) [**69**]. L'ensemble d'entraînement contient 15 951 images déséquilibrées en classe, tandis que l'ensemble de test contient 200 images positives et 200 images négatives.

KTH-TIPS2 contient 4 752 images de matériaux réparties en 11 classes. Les images sont des photographies de matériaux prises à différentes distances. Chaque classe (type de matériaux) est divisée en quatre échantillons (108 images chacun) de distances variées. En utilisant le protocole standard, nous entraînons le modèle sur un échantillon (11×108 images), tandis que le reste est utilisé pour le test [**63**]. Au total, chaque ensemble d'entraînement contient 1 188 images.

3.1.2. Exploration de paramétrages spécifiques aux jeux de données

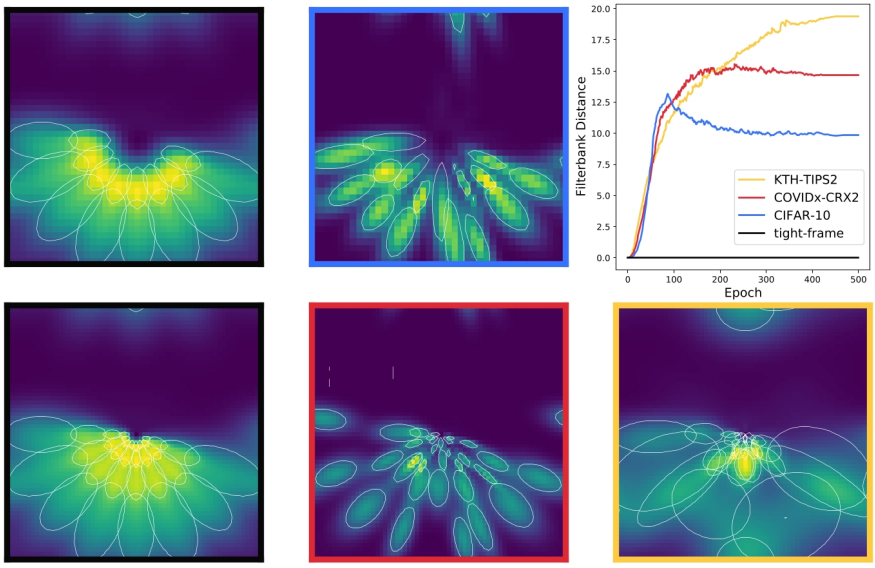


Fig. 3.2. Le réseau de *scattering* paramétré apprend des filtres spécifiques aux jeux de données. Le graphique (en haut à droite) montre l'évoluation de la distance de banque de filtres durant l'entraînement pour les différents jeux de données. Nous visualisons les banques de filtres ayant des paramétrages spécifiques au jeux de données (couleurs de bordure de la légende) dans l'espace de Fourier. Les filtres de *scattreing* optimisés pour des images naturelles (CIFAR-10) et médicales (COVIDx CRX2) deviennent plus sélectifs en orientation, c'est-à-dire plus fins dans le domaine de Fourier. Les filtres optimisés pour la discrimination de texture (KTH-TIPS2) deviennent moins sélectifs en orientation et s'écartent le plus d'une configuration en repère ajusté.

Nous comparons d'abord les paramétrages des filtres entraînés sur les jeux de données et évaluons leurs similitudes avec un repère ajusté. Plus précisément, nous formons nos réseaux de *scattering* paramétrés à l'aide de la formulation canonique des ondelettes de Morlet ainsi qu'une couche de classification linéaire et comparons qualitativement les similitudes de la banque de filtres appris à celle de l'initialisation en repère ajusté. Pour faciliter la comparaison quantitative, nous utilisons la métrique introduite à la section 2.1.2.

Le graphique de la figure 3.2 exploite la distance entre banques de filtres pour montrer l'évolution des réseaux de *scattering* initialisés en repère ajusté et entraînés sur les différents jeux de données. Chaque réseau est entraîné sur 1188 échantillons de son ensemble de données respectif (la taille standard pour KTH-TIPS2). Tous les filtres s'écartent rapidement du repère ajusté, mais ceux de KTH-TIPS2 changent le plus longtemps et finissent par s'écarter le plus. Nous observons également que les filtres initialisés aléatoirement deviennent plus similaires à notre initialisation en repère ajusté au cours de l'entraînement (voir la fig. 3.3).

Sur le côté gauche de la figure 3.2, nous visualisons les paramétrages du réseau de *scattering* spécifiques aux jeux de données dans l'espace de Fourier. Des contours blancs sont dessinés autour de chaque ondelette de Morlet pour plus de clarté. La bordure noire supérieure correspond à l'initialisation en repère ajusté avec J = 2, illustrée pour comparaison avec CIFAR-10 en bleu (également J = 2). La bordure noire inférieure correspond à l'initialisation en repère ajusté avec J = 4, illustrée à des fins de comparaison avec celle en rouge de COVIDX-CRX2 et celle en jaune de KTH-TIPS2 (les deux J = 4).

Les filtres optimisés sur le jeu de données de texture KTH-TIPS2 (jaune) deviennent moins sélectifs en orientation (plus larges dans l'espace de Fourier) que l'initialisation en repère ajusté, les filtres à J = 0 devenant les moins sélectifs en orientation de toute la banque de filtres. Les filtres optimisés sur CIFAR-10 ainsi que ceux optimisés sur COVIDx-CRX2 deviennent plus sélectifs en matière d'orientation que leurs homologues en repère ajusté. Nous soupçonnons que cela est dû à une plus grande dépendance aux bords pour les jeux de données de classification d'objets (par exemple, on peut reconnaître un chat uniquement par son contour), lesquels semblent nécessiter davantage de filtres sélectifs d'orientation. À l'inverse, les ondelettes de Morlet optimisées pour la classification des textures semblent écarter certaines informations de contour au profit de filtres moins spécifiques à l'orientation (il n'y a pas de bord dans des motifs de textures). Chaque paramétrage spécifique à un ensemble de données semble écarter les informations inutiles de l'initialisation en repère ajusté en faveur de l'accentuation des attributs spécifiques au problème. Dans la section 3.1.4, nous démontrons que ces filtres appris sont non seulement interprétables, mais qu'ils améliorent les performances des modèles de *scattering*, suggérant que le repère ajusté n'est pas optimal pour de nombreux problèmes d'intérêt. Néanmoins, un repère ajusté constitue un bon point de départ pour l'apprentissage. En effet, les paramétrages spécifiques aux jeux de données pour COVIDX-CRX2 et KTH-TIPS2 sont visuellement très différents, mais leur distance

de banque de filtres reste similaire par rapport à l'initialisation en repère ajusté (voir la fig. 3.2), laquelle est petite par rapport aux distances observées pour les modèles initialisés et entraînés de manière aléatoire (voir la fig. 3.3).

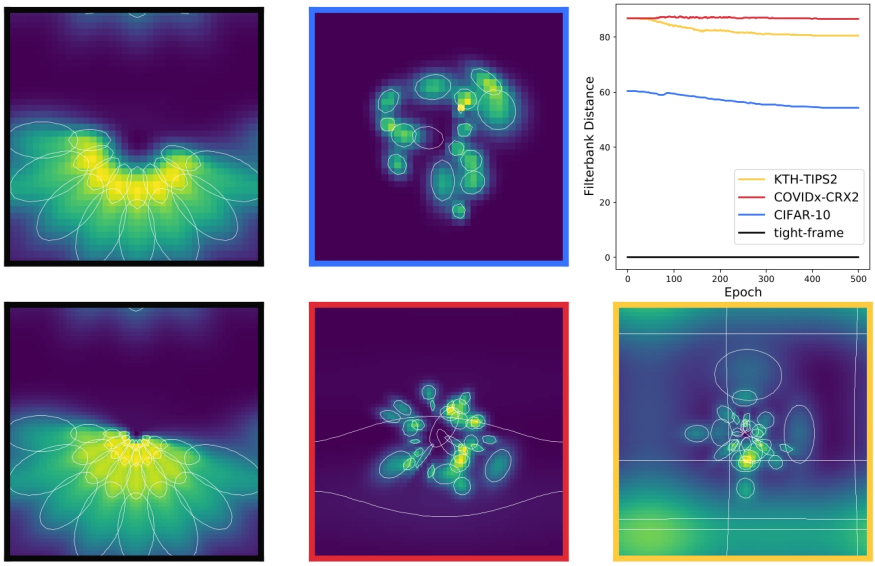
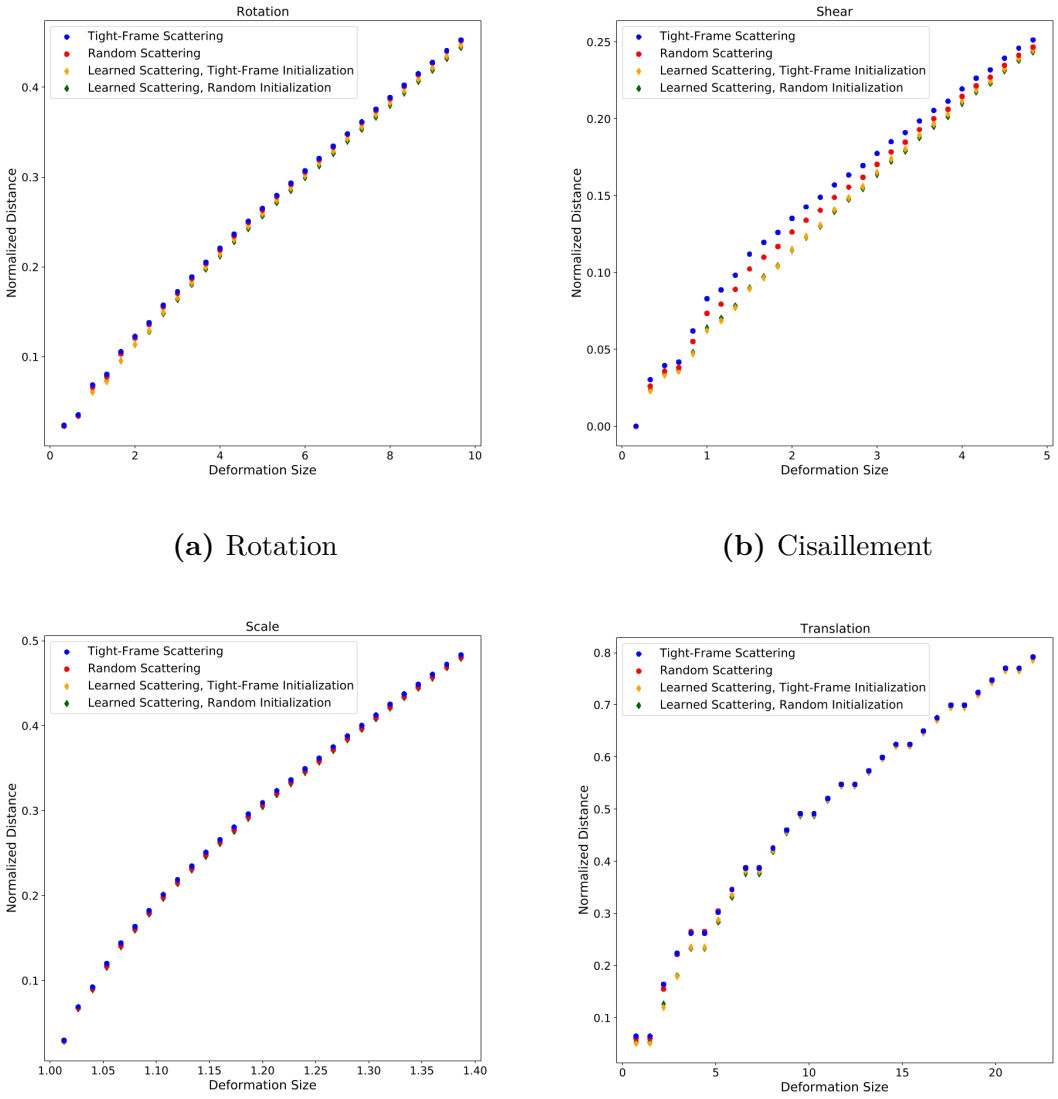


Fig. 3.3. Le graphique montre l'évolution de la distance de banque de filtres pour des filtres initialisés à partir d'une initialisation aléatoire et entraînés sur les différents jeux de données. À gauche, nous visualisons les banques de filtres ayant des paramétrages spécifiques aux jeux de données dans l'espace de Fourier. Le graphique de droite montre que les banques de filtres initialisées de manière aléatoire ressemblent de plus en plus à un repère ajusté pendant l'apprentissage.

3.1.3. Robustesse aux déformations

Nous avons vu à la section 1.7.3 que la transformée de *scattering* est stable aux petites déformations de la forme $x(u-\tau(u))$ où $x(u)$ est un signal et τ un difféomorphisme. Compte tenu des changements substantiels apportés aux filtres dans le processus d'apprentissage, nous nous demandons maintenant si ceux-ci semblent s'écarter de manière significative du résultat de stabilité obtenu à partir de la construction soigneusement artisanale proposée dans [**39**], et largement utilisée dans des travaux antérieurs, par exemple, [**10, 20**]. Pour évaluer la robustesse de notre réseau de *scattering* paramétré à différentes distorsions géométriques, nous appliquons plusieurs déformations à une image radiographique thoracique x avec une force de déformation variable. Plus précisément, nous entraînons nos réseaux de *scattering* paramétrés en utilisant la formulation d'ondelettes canonique de Morlet avec une couche de classification linéaire. L'image transformée est notée $\tilde{x}$. Pour chacune des différentes forces de déformation, nous traçons la distance euclidienne entre la transformée de *scattering* construite à partir de l'image originale $S(x)$ et la transformée de *scattering*



(a) Rotation **(b)** Cisaillement

(c) Étirement **(d)** Translation

Fig. 3.4. Distances normalisées entre les représentations de *scattreing* d'une image et de sa déformation. Notre transformée de *scattering* paramétrée a une stabilité aux déformations similaire à la transformée de *scattering*.

construite à partir de l'image transformée $S(\tilde{x})$. Nous normalisons ensuite la distance obtenue par $S(x)$ pour mesurer l'écart relatif des coefficients de *scattering* (conçus ou appris). La figure 3.4 montre des résultats représentatifs pour une petite rotation, un cisaillement et un étirement sur les images du jeu de données COVIDx. La figure 3.5 explore la translation, ainsi que deux déformations définies par des difféomorphismes (notés Transformation personnalisée 1 et 2, respectivement) selon l'équation 1.7.4. Elles sont définies comme suit :

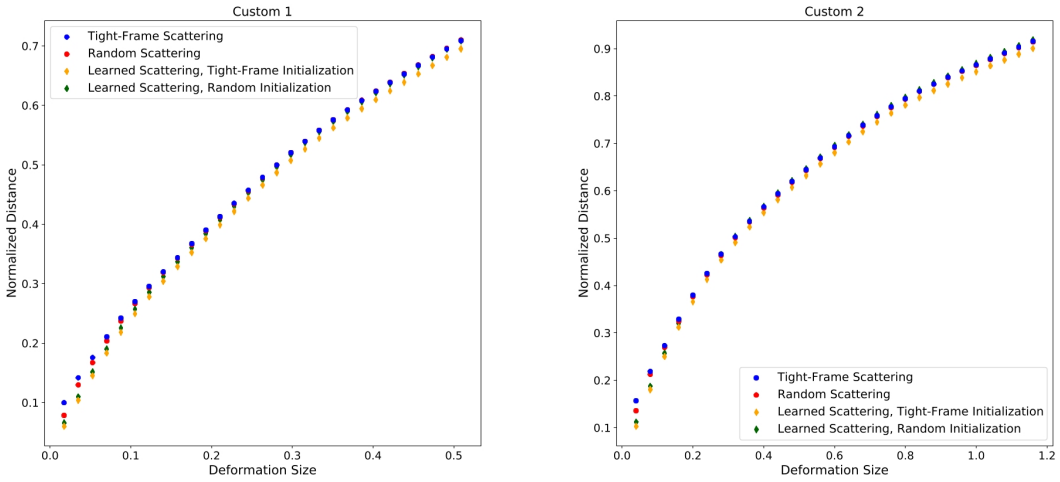
$$\tau_\epsilon^1(u) = \epsilon \begin{bmatrix} 0.3u_1^2 + 0.2u_2^2 \\ 0.2(0.2u_1) \end{bmatrix}, \tau_\epsilon^2(u) = \epsilon \begin{bmatrix} 0.3(u_1^2 + u_2^2) \\ -0.3(2u_1u_2) \end{bmatrix}.$$

Les forces pour l'ensemble des déformations utilisées vont de 0 à la valeur maximale donnée dans le tableau 3.1, sauf pour l'étirement qui va de 1 à sa valeur maximale.

Tableau 3.1. Les déformations et leur valeur maximale

Déformation	Valeur maximale
Personnalisée 1	1
Personnalisée 2	1
Rotation	10
Étirement	1.4
Cisaillement	5
Translation	22

Nous observons que malgré les changements substantiels dans les filtres, les réseaux conservent les propriétés de robustesse pour ces déformations simples, indiquant ainsi que l'utilisation de filtres appris (au lieu de filtres conçus) ne nuit pas nécessairement à la stabilité de la transformée résultante.



(a) Personnalisée 1 **(b)** Personnalisée 2

Fig. 3.5. Distances normalisées entre les représentations de *scattering* d'une image et de sa déformation. Même dans le cas des transformations personalisées, le modèle paramétré présente une stabilité similaire au modèle non-paramétré.

3.1.4. Résultats

Nous évaluons le réseau de *scattering* paramétré dans des contextes de données étiquetées limitées. Suivant le protocole d'évaluation de [**47**], nous sous-échantillonnons chaque jeu

de données à différentes tailles d'échantillons pour mettre en valeur les performances des architectures basées sur la transformée de *scattering* en fonction de la quantité de données d'entraînement.

3.1.4.1. Protocole.

Nous évaluons l'utilisation de la transformée de *scattering* paramétrée avec deux modèles communs. Dans le premier cas, on considère la transformée de *scattering* comme alimentant un modèle linéaire simple (noté LL). Les configurations LL sont utilisées pour évaluer la séparabilité linéaire des représentations de *scattering* obtenues et ont l'avantage supplémentaire de fournir un modèle plus interprétable. Dans le second cas, nous adoptons l'approche de [**47**] et considérons la transformée de *scattering* comme la première étape d'un CNN plus profond, dans notre cas un réseau résiduel large (WRN) [**72**]. L'architecture du modèle hybride avec WRN est décrite à la section 1.7.4.

Pour les deux modèles (LL et WRN), nous comparons les réseaux de *scattering* paramétrés appris (LS) aux réseaux de *scattering* traditionnels fixes (S). Pour la transformée de *scattering* apprise (LS), nous considérons deux approches de paramétrage : le paramétrage canonique et la paramétrage équivariant, décrits à la section 2.1.1.3. Pour montrer l'importance de l'approche paramétrique, nous testons également le paramétrage naïf où tous les pixels des ondelettes sont adaptés, que nous appelons paramétrage par pixels.

Dans cette approche de paramétrage par pixels, nous assouplissons les contraintes et optimisons les pixels des filtres dans l'espace de de Fourier directement, sans la contrainte qu'ils conservent la forme d'ondelettes de Morlet. Cette alternative revient à initialiser les filtres d'un CNN avec des ondelettes de Morlet et à laisser le modèle optimiser les pixels des noyaux sans aucune contrainte en utilisant la rétropropagation.

Pour chaque architecture de *scattering*, nous considérons l'initialisation aléatoire (A) et celle en repère ajusté (TF). Les modèles de *scattering* fixes avec la construction TF sont équivalents aux transformées de *scattering* traditionnelles. Enfin, nous comparons également notre approche à un WRN entièrement appris (sans a priori de *scattering*) et au ResNet-50 [**24**] appliqué directement aux données d'entrée.

Dans toutes les configurations de *scattering*, une couche de normalisation selon le lot est ajoutée après chaque couche de *scattering*. La classification est effectuée via une couche SOFTMAX donnant la sortie finale. Tous les modèles sont entraînés à l'aide de la fonction de perte d'entropie croisée, minimisée par une descente de gradient stochastique avec une impulsion de 0.9. Un régularisateur est appliqué au modèle linéaire et au WRN. Le taux d'apprentissage est planifié selon la politique d'un cycle [**59**].

Dans nos expériences, nous entraînons les modèles sur un petit sous-ensemble aléatoire des données d'entraînement, et testons toujours sur l'ensemble du test comme cela a été

fait dans [**47**]. Nous identifions en gras les meilleurs résultats pour chaque taille d'ensemble d'entraînement dans les tableaux. Pour obtenir des résultats comparables et reproductibles, nous contrôlons le comportement déterministe du GPU et nous nous assurons que chaque modèle est initialisé de la même manière pour la même graine. De plus, nous utilisons le même ensemble de graines pour les modèles évalués sur le même nombre d'échantillons. Par exemple, l'hybride appris TF avec un modèle linéaire serait évalué sur les mêmes dix graines que l'hybride en repère ajusté fixe avec un modèle linéaire lorsqu'il est entraîné sur 100 échantillons de CIFAR-10. Certaines fluctuations sont inévitables lors du sous-échantillonnage des ensembles de données. Par conséquent, tous nos chiffres incluent les moyennes et l'erreur standard calculées sur les différentes graines.

Nous reproduisons certaines des expériences avec des réseaux de *scattering* appris suivis du WRN sur CIFAR-10, COVIDx-CRX2 et KTH-TIPS2 en utilisant la fonction de perte cosinus [**5**]. Les résultats sont rapportés à la section 3.1.4.5. Finalement, nous conduisons une étude sur le nombre optimal de filtres pour la transformée de *scattering* apprise avec couche linéaire sur CIFAR-10 à la section 3.1.4.6.

3.1.4.2. CIFAR-10.

Le tableau 3.2 rapporte l'évaluation de notre approche de *scattering* apprise sur CIFAR-10 avec des tailles d'échantillons d'apprentissage de 100, 500, 1K et 50K (qui est en fait tout l'ensemble d'apprentissage). L'ensemble d'apprentissage est augmenté à l'aide d'un retournement horizontal, d'un recadrage aléatoire et d'une augmentation automatique pré-spécifiée pour CIFAR-10 établie par le processus d'auto-augmentation. Nous utilisons une échelle spatiale de $J = 2$ dans les transformées de *scattering*.

Les modèles linéaires ont été entraînés en utilisant un taux d'apprentissage maximal de 0.06 pour tous les paramètres sur 5K, 1K, 500 et 500 époques pour 100, 500, 1K, 50K échantillons, respectivement. Les modèles WRN hybrides ont été entraînés en utilisant un taux d'apprentissage maximal de 0.1 sur 3K, 2K, 1K et 200 époques pour 100, 500, 1K et 50K échantillons respectivement. Nous utilisons la descente de gradient par lot sauf lorsque les modèles sont entraînés avec des échantillons de 50K où nous utilisons une descente de gradient par mini-lot de taille 1024. Nous entraînons les modèles sur 10 graines différentes.

Comme le montre le tableau 3.2, les réseaux de *scattering* avec des filtres optimisés par pixels fonctionnent le moins bien dans le régime des données limitées. Il n'y a pas assez de données et trop de paramètres pour apprendre efficacement les pixels des ondelettes. Ajouter plus de contraintes (c'est-à-dire contraindre les ondelettes à être Morlet) est bénéfique dans ce cadre. Nous observons également que le paramétrage canonique donne des performances similaires au paramétrage équivariant (c'est-à-dire que la plupart des erreurs standard se chevauchent). Ainsi, ajouter encore plus de contraintes, en réduisant le nombre de paramètres

Tableau 3.2. CIFAR-10 - Précision moyenne et erreur std. sur 10 graines, avec $J = 2$ et plusieurs tailles d'échantillons d'apprentissage. La transformée de *scattering* apprise avec initialisation TF améliore les performances de toutes les architectures, tandis que celle initialisée de manière aléatoire nécessite davantage de données d'entraînement pour atteindre des performances similaires.

Arch.	Init.	Paramétrage	100 exemples	500 exemples	1000 exemples	Tout
LS+LL†	TF	Canonique	37.84 ± 0.57	$\mathbf{52.68} \pm 0.31$	$\mathbf{57.43} \pm 0.17$	**69.57**
LS+LL†	TF	Équivariant	$\mathbf{39.69} \pm 0.56$	51.98 ± 0.25	57.01 ± 0.16	66.65
LS+LL	TF	Par pixel	32.30 ± 0.69	47.14 ± 0.91	51.87 ± 0.34	64.53
S +LL	TF	-	36.01 ± 0.55	48.12 ± 0.25	53.25 ± 0.24	65.58
LS+LL†	A	Canonique	34.81 ± 0.60	49.6 ± 0.39	55.72 ± 0.39	69.39
LS+LL†	A	Équivariant	34.67 ± 0.73	46.59 ± 0.60	52.95 ± 0.36	65.64
LS+LL	A	Par pixel	29.44 ± 0.41	42.14 ± 0.27	47.44 ± 0.43	62.72
S +LL	A	-	29.77 ± 0.47	41.85 ± 0.41	46.3 ± 0.37	57.72
LS+WRN†	TF	Canonique	$\mathbf{43.60} \pm 0.87$	$\mathbf{63.13} \pm 0.29$	70.14 ± 0.26	93.61
LS+WRN†	TF	Équivariant	39.86 ± 1.59	62.85 ± 0.32	69.52 ± 0.23	92.57
LS+WRN	TF	Par pixel	39.20 ± 0.80	54.14 ± 0.68	57.59 ± 0.48	92.97
S +WRN	TF	-	43.16 ± 0.78	61.66 ± 0.32	68.16 ± 0.27	92.27
LS+WRN†	A	Canonique	41.42 ± 0.65	59.84 ± 0.40	67.40 ± 0.28	93.36
LS+WRN†	A	Équivariant	40.84 ± 1.02	60.81 ± 0.40	68.62 ± 0.31	92.53
LS+WRN	A	Par pixel	31.49 ± 0.63	45.85 ± 0.43	50.72 ± 0.28	91.86
S +WRN	A	-	32.08 ± 0.46	46.84 ± 0.21	52.76 ± 0.33	85.35
WRN-16–	-	-	38.78 ± 0.72	62.97 ± 0.41	$\mathbf{71.37} \pm 0.31$	**96.84**
ResNet-50	-	-	33.17 ± 0.92	52.13 ± 0.74	64.42 ± 0.40	91.23

paramètres : 156k pour S+LL; 155k pour LS+LL; 22.6M pour S+WRN; 22.6M pour LS+WRN; 22.3M pour WRN; et 22.5M pour ResNet
†: nôtre; TF: Repère ajusté; LS: *Scattering* apprise; S: *Scattering*; A: Aléatoire

apprenables dans la transformée de *scattering* paramétrée, ne dégrade pas les performances dans le régime des données limitées. Lorsque l'ensemble d'apprentissage complet est utilisé, nous observons que le paramétrage canonique donne une meilleure précision que le paramétrage équivariant, montrant que lorsque le nombre de données étiquetées n'est pas limité, l'ajout de plus de contraintes peut diminuer les performances.

Nous observons que l'apprentissage aléatoire initialisé avec un paramétrage canonique n'atteint des performances similaires à l'apprentissage canonique TF que lorsqu'il est entraîné sur tout l'ensemble de données. Ces résultats suggèrent que l'initialisation TF, dérivée de principes de traitement de signal rigoureux, est empiriquement bénéfique comme point de départ pour un petit nombre d'échantillons, mais peut être améliorée par apprentissage.

Parmi les modèles linéaires, nos réseaux de *scattering* appris initialisés en TF (autant canonique qu'équivariant) surpassent de manière significative tous les autres dans les différents paramètres d'échantillonnage. Cela démontre que les réseaux de *scattering* appris obtiennent une représentation plus linéairement séparable que leurs homologues fixes, peut-être en créant une plus grande invariance intra-classe spécifique au jeu de données.

La figure 2.1 montre la partie réelle des filtres d'ondelettes canoniques avant et après optimisation sur CIFAR-10.

Parmi les modèles hybrides WRN, la transformée de *scattering* paramétrée canonique initialisée en TF est la plus performante. Celle TF canonique apprise s'améliore toujours par rapport à celle TF fixe lorsqu'elle est associé à un WRN, ce qui indique qu'une certaine

perte d'informations dans la représentation de la transformée de *scattering* fixe est atténuée par un réglage ou une optimisation basés sur les données. Enfin, notre approche surpasse le ResNet-50 et surpasse le WRN-16-8 sur 100 et 500 échantillons d'entraînement, démontrant l'efficacité de la transformée de *scattering* paramétrée dans le régime des données limitées. Cependant, et en concordance avec nos hypothèses, le WRN-16-8 surpasse notre modèle lorsque le nombre de données devient plus important, c'est-à-dire sur 1 000 échantillons et 50 000 échantillons.

3.1.4.3. COVIDx CRX-2.

Dans nos expériences sur COVIDx CRX-2, nous entraînons les modèles sur un sous-ensemble équilibré en classes de l'ensemble d'apprentissage. Nous redimensionnons les images à une taille de 260×260 et entraînons notre réseau avec des recadrages aléatoires de taille 224×224. La seule augmentation de données que nous utilisons est le retournement horizontal aléatoire.

L'échelle spatiale de la transformée de *scattering* est fixée à $J = 4$. Le tableau 3.3 rapporte notre évaluation sur des tailles d'échantillon de 100, 500 et 1K images. Nous utilisons le même protocole que pour CIFAR-10. Tous les modèles ont été entraînés sur 400 époques en utilisant un taux d'apprentissage maximal de 0.01. Tous les modèles hybrides sont entraînés avec une taille de mini-lot de 128.

Le paramétrage canonique de Morlet donne des performances similaires au paramétrage équivariant (c'est-à-dire que la plupart des erreurs standard se chevauchent), comme cela a également été observé avec CIFAR-10.

Lorsque les réseaux de *scattering* sont ajoutés à une couche linéaire, les réseaux paramétrés initialisés en TF (autant canonique qu'équivariant) fonctionnent mieux que le réseau TF fixe, ce qui montre la viabilité de notre approche sur des données du monde réel. Nous observons que l'apprentissage initialisé aléatoirement donne des performances inférieures à l'apprentissage TF sur 100 et 500 échantillons. Sur 1K, il atteint des performances similaires, démontrant que l'initialisation aléatoire peut atteindre des performances comparables à TF avec suffisamment de données. WRN-16-8 fonctionne moins bien que l'apprentissage initialisé en TF suivi d'une couche linéaire. Lorsqu'il est combiné avec un CNN, TF appris fonctionne également mieux que TF fixe et surpasse WRN-16-8 et ResNet-50.

3.1.4.4. KTH-TIPS2.

Le tableau 3.3 présente également les précisions de classification pour KTH-TIPS2. Dans nos expériences, nous redimensionnons les images de KTH-TIPS2 à une taille de 200×200 et entraînons notre réseau avec des recadrages aléatoires de taille 128×128 pixels. Les

Tableau 3.3. COVIDx CRX-2 et KTH-TIPS2 - Précision moyenne et erreur std. avec $J = 4$ sur 10 graines et 16 graines respectivement. (COVIDx CRX-2) Le réseau de *scattering* appris initialisé en TF fonctionne mieux que les modèles qui n'intègrent pas d'à priori de *scattering*. (KTH-TIPS2) De même, le WRN-16-8 et ResNet-50 fonctionnent extrêmement mal par rapport aux modèles hybrides entraînés sur KTH-TIPS2.

Arch.	Init.	Param.	C-100 ex.	C-500 ex.	C-1000 ex.	KTH-1188 ex.
LS+LL†	TF	Canonique	82.30 ± 1.78	$\mathbf{88.50} \pm 0.71$	$\mathbf{89.90} \pm 0.40$	66.09 ± 1.05
LS+LL†	TF	Équivariant	$\mathbf{83.06} \pm 1.53$	87.56 ± 0.94	89.15 ± 0.60	$\mathbf{66.41} \pm 1.24$
S +LL	TF	-	81.08 ± 1.88	87.20 ± 0.77	89.23 ± 0.69	66.17 ± 1.10
LS+LL†	A	Canonique	76.85 ± 1.50	86.45 ± 0.95	89.70 ± 0.65	65.79 ± 0.85
LS+LL†	A	Équivariant	76.73 ± 1.57	85.64 ± 1.38	87.98 ± 0.55	65.31 ± 1.42
S +LL	A	-	76.08 ± 1.56	84.13 ± 0.91	86.80 ± 0.41	61.37 ± 0.82
LS+WRN†	TF	Canonique	81.20 ± 1.73	90.50 ± 0.70	93.68 ± 0.35	$\mathbf{69.23} \pm 0.67$
LS+WRN†	TF	Équivariant	$\mathbf{81.86} \pm 2.07$	$\mathbf{91.56} \pm 0.52$	$\mathbf{93.97} \pm 0.34$	68.55 ± 0.80
S +WRN	TF	-	80.85 ± 1.85	89.05 ± 0.59	91.90 ± 0.54	68.84 ± 0.71
LS+WRN†	A	Canonique	80.95 ± 1.54	88.08 ± 0.70	91.65 ± 0.55	68.30 ± 0.47
LS+WRN†	A	Équivariant	80.12 ± 1.76	87.44 ± 1.17	91.40 ± 0.67	67.50 ± 0.72
S +WRN	A	-	80.63 ± 1.73	86.68 ± 0.59	90.60 ± 0.50	66.29 ± 0.36
WRN-16-	-	-	80.50 ± 1.15	85.95 ± 2.04	88.82 ± 1.64	51.24 ± 1.37
ResNet-50	-	-	74.04 ± 1.35	86.45 ± 0.51	90.86 ± 0.57	44.95 ± 0.65

C: COVIDx CRX-2 # params : 493K pour LS/S+LL; 23.05M pour LS/S+WRN; 22.3M pour WRN; 23.5M pour ResNet
KTH: KTH-TIPS2 # params : 883K pour LS/S+LL; 23.7M pour LS/S+WRN; 22.3M pour WRN; 23.5M pour ResNet
†: nôtre; TF: Repère ajusté; LS: *Scattering* apprise; S: *Scattering*; A: Aléatoire

données d'entraînement sont complétées par des retournements horizontaux aléatoires et des rotations aléatoires.

Tous les réseaux de *scattering* utilisent une échelle spatiale de 4. Nous fixons le taux d'apprentissage maximal des paramètres de *scattering* à 0.1 alors qu'il est fixé à 0.001 pour tous les autres paramètres. Tous les modèles hybrides sont entraînés avec une taille de mini-lot de 128. Les modèles linéaires hybrides sont entraînés pour 250 époques, tandis que les modèles WRN hybrides sont entraînés pour 150 époques. Nous évaluons chaque modèle en l'entraînant avec quatre graines différentes sur chaque échantillon de matériaux, ce qui représente un total de 16 exécutions.

Avec l'initialisation TF et une couche linéaire, nous observons que les performances sont similaires pour les différentes architectures. Les performances de l'apprentissage initialisé de manière aléatoire sont également similaires à celles en TF. Le modèle fixe et initialisé de manière aléatoire est le moins performant, ce qui montre l'apprentissage est requis et que même des filtres mal initialisés peuvent être efficacement optimisés. Dans l'ensemble, ces résultats corroborent davantage nos découvertes précédentes, notamment que l'initialisation TF agit comme un bon a priori pour les réseaux de *scattering*. Parmi tous les modèles hybrides WRN, le modèle paramétré initialisé en TF utilisant la paramétrisation canonique atteint la précision moyenne la plus élevée. Nous notons que si WRN augmente les performances par rapport à la couche linéaire, il augmente également de manière significative le nombre total de paramètres, illustrant ainsi le compromis entre performances et complexité du modèle. Le WRN-16-8 et le ResNet-50 fonctionnent extrêmement mal par rapport aux

modèles hybrides, ce qui montre l'efficacité des a priori de *scattering* pour la discrimination de texture.

3.1.4.5. Fonction de perte cosinus.

Nous reproduisons les expériences précédentes avec des réseaux de *scattering* appris suivis d'un WRN sur CIFAR-10, COVIDx-CRX2 et KTH-TIPS2. Nous utilisons les mêmes paramètres à l'exception de l'utilisation de la fonction de perte cosinus [**5**] au lieu de l'entropie croisée. La perte de cosinus est décrite dans la section 1.5.2.1. Les filtres d'ondelettes sont initialisés à l'aide de la construction en repère ajusté. Le tableau 3.4 montre la précision moyenne sur les trois jeux de données. Pour CIFAR-10 et COVIDx-CRX2, les performances sont inférieures lorsque les modèles sont entraînés à l'aide de la fonction de perte cosinus. Le même comportement n'est pas observé lorsque les modèles sont entraînés sur KTH-TIPS2 où, en fait, les performances augmentent légèrement. Ainsi, la perte de cosinus peut améliorer les performances pour certains jeux de données, mais peut nuire également.

Tableau 3.4. CIFAR-10, COVIDx-CRX2 et KTH-TIPS2 - Précision moyenne et erreur std. en utilisant la fonction de perte cosinus.

Init.	Arch.	Données	Perte	100 ex.	500 ex.	1000 ex.	1188 ex.
TF	LS+WRN	CIFAR-10	CE	$\mathbf{43.6} \pm 0.87$	$\mathbf{63.13} \pm 0.29$	$\mathbf{70.14} \pm 0.26$	-
TF	LS+WRN	CIFAR-10	Cosinus	42.94 ± 0.77	61.42 ± 0.26	68.29 ± 0.18	-
TF	LS+WRN	COVIDx	CE	$\mathbf{81.20} \pm 1.73$	$\mathbf{90.50} \pm 0.70$	$\mathbf{93.68} \pm 0.35$	-
TF	LS+WRN	COVIDx	Cosinus	80.03 ± 2.16	89.53 ± 0.89	92.75 ± 0.65	-
TF	LS+WRN	KTH-TIPS2	CE	-	-	-	69.23 ± 0.67
TF	LS+WRN	KTH-TIPS2	Cosinus	-	-	-	$\mathbf{70.86} \pm 0.67$

TF: Repère ajusté LS: *Scattering* apprise S: *Scattering* CE: Entropie croisée

3.1.4.6. Nombre de filtres par échelle spatiale.

Dans cette section, nous étudions l'effet de la modification du nombre de filtres (L) par échelle spatiale sur CIFAR-10. Nous utilisons les mêmes paramètres et hyperparamètres que ceux décrits dans les sections précédentes. Jusqu'à présent, dans toutes les expériences, nous avons fixé le nombre de filtres par échelle à 8 pour une comparaison équitable.

Pour cette expérience, nous formons un réseau de *scattering* paramétré suivi d'une couche linéaire où les ondelettes sont initialisées à l'aide de la construction en repère ajusté et du paramétrage canonique. L'échelle spatiale est fixée à 2. Nous n'utilisons pas l'auto-augmentation sur l'ensemble d'apprentissage pour isoler la variable étudiée. Le tableau 3.5 montre la précision du modèle entraîné sur l'ensemble d'entraînement complet pour différentes valeurs de L. Nous observons que les performances augmentent lorsque le nombre de filtres par échelle augmente également. À autour de 14-16 filtres par échelle spatiale, les performances semblent cesser d'augmenter.

Le tableau 3.6 montre la précision moyenne sur différentes tailles d'échantillons d'apprentissage où le nombre de filtres par échelle est défini à 8 ou 16. Nous considérons également la version fixe de la transformée de *scattering*. Sur toutes les tailles d'échantillons d'apprentissage, nous observons que les performances les plus élevées sont obtenues avec une transformée de *scattering* apprise utilisant 16 filtres par échelle spatiale. Lorsque le réseau de *scattering* est fixe, on observe que les performances sont moindres avec 16 filtres au lieu de 8. Il semble que l'augmentation du nombre de filtres par échelle n'est bénéfique que dans la version apprise du réseau de *scattering*, où il est plus facile d'exploiter cette liberté supplémentaire.

Tableau 3.5. CIFAR-10 - Précision de la transformée de *scattering* apprise suivie d'une couche linéaire (LS + LL) selon plusieurs nombres de filtres par échelle (L) entraînés sur tout l'ensemble d'apprentissage. Les filtres d'ondelettes sont initialisés à l'aide de la construction en repère ajusté et du paramétrage canonique. L'échelle spatiale est fixée à 2. Aucune augmentation automatique n'est utilisée pour cette expérience. Nous observons que les performances augmentent lorsque le nombre de filtres par échelle (L) augmente également. À autour de 14 filtres par échelle spatiale, les performances semblent cesser d'augmenter.

L	Tout
2	63.59
4	70.94
6	74.03
8	74.94
10	76.40
12	77.01
14	**77.36**
16	77.33

Tableau 3.6. CIFAR-10 - Précision moyenne et erreur std. sur 10 graines avec plusieurs tailles d'échantillons d'apprentissage et différentes valeurs de L. Les filtres d'ondelettes sont initialisés à l'aide de la construction en repère ajusté. L'échelle spatiale est fixée à 2. Aucune augmentation automatique n'est utilisée pour cette expérience. Sur toutes les tailles d'échantillons d'apprentissage, nous observons que les performances les plus élevées sont obtenues avec une transformée de *scattering* paramétrée utilisant 16 filtres par échelle spatiale.

Arch.	Paramétrage	L	100 ex.	500 ex.	1000 ex.	Tout
LS+LL†	Canonique	8	39.70 ± 0.62	50.74 ± 0.30	54.76 ± 0.22	74.94
LS+LL†	Canonique	16	39.73 ± 0.39	$\mathbf{54.17} \pm 0.36$	$\mathbf{58.36} \pm 0.29$	**77.33**
S +LL	-	8	37.55 ± 0.62	49.67 ± 0.33	53.96 ± 0.48	70.71
S +LL	-	16	35.85 ± 0.48	48.2 ± 0.27	52.74 ± 0.25	70.64

3.2. Modèle de diffusion sur le modèle CATS

3.2.1. Jeu de données

Présentons la construction du jeu de données pour notre adaptation des modèles de diffusion appliqué à la modélisation par apprentissage profond du modèle économique CATS.

Le jeu de données est obtenu en utilisant le modèle économique CATS pour effectuer des simulations et en extrayant les quantités entrantes et sortantes du marché de travail. Un échantillon est composé de trois composantes - deux entrées et une sortie : un vecteur d'emploi au temps $t-1$ (noté $\boldsymbol{x}_{prec}$), une matrice d'informations H et le vecteur d'emploi au temps t (noté $\boldsymbol{x}$) à prédire. Nous référons à la section 1.6.3 pour une description plus précise de ces éléments.

Nous fixons le nombre de travailleurs à $W = 1000$ et le nombre de firmes à $n_F = 120$, dont 100 sont des firmes de consommations (firmes-C) et 20 sont des firmes de capital (firmes-K). En ajoutant la catégorie des sans-emplois, nous obtenons des vecteurs d'emploi $\boldsymbol{x}_t$ de dimension 121 dont la somme des composantes doit être égale à 1000. Sa première composante correspond aux sans-emplois, alors que les composantes 1 à 100 correspondent aux firmes-C et que celles de 101 à 120 correspondent aux firmes-K.

Nous normalisons les données entre $[-1,1]$ pour pouvoir appliquer le modèle de diffusion à l'aide d'une normalisation Min-Max. Les résultats peuvent être reconvertis en entier entre [0,1000] pour l'analyse des résultats. De plus, afin d'éviter une mauvaise concordance entre l'entraînement et l'échantillonnage (similairement à ce qui est discuté à la section 1.8.3.4), nous transformons toutes les prédictions $\hat{\boldsymbol{x}}_\theta$ du réseau de sorte que lorsque reconverties à [0,1000], la somme de leurs composantes donne 1000. Seulement lors de la conversion finale nous nous assurons en plus que les valeurs converties dans [0,1000] soient en fait des entiers (pour ce faire, nous considérons le vecteur des parties entières et ajoutons 1 aux composantes qui ont les plus grandes parties fractionnaires jusqu'à obtenir la somme voulue).

Nous créons des séries temporelles de 1500 temps à partir de CATS, desquels nous conservons que les 700 derniers temps, afin que les valeurs obtenues représentent des quantités plus réalistes provenant d'une économie mûre. En effet, pour les premiers temps, le modèle est encore en train de se calibrer et la dynamique économique n'est pas plausible. Pour générer ces séries, nous fixons les hyperparamètres de CATS à des valeurs standards, à l'exception de l'expérience de la section 3.2.2.4, où nous explorons des perturbations de ces hyperparamètres et vérifions que les modèles de diffusion réussissent à s'adapter à ceux-ci. À la section 3.2.2.2, les expériences sont conduites à l'aide d'une unique série temporelle générée à partir de l'ensemble d'hyperparamètres standard. Cela nous place dans le cadre des données limitées. À la section 3.2.2.3, nous explorons l'idée d'élargir le jeu de données à 10 séries temporelles issues des mêmes hyperparamètres. Cela permet d'avoir un nombre de données plus important et cela reste réaliste, considérant que l'on peut imaginer créer des modèles en regroupant les données de plusieurs villes ou pays, par exemple. À la section 3.2.2.4, encore 10 séries sont utilisées dans cette expérience d'entraînement conjoint, mais chaque série est issue d'une perturbation des hyperparamètres standards. Dans toutes les expériences, nous réservons 256 exemples aléatoires pour la validation tandis que le reste est utilisé pour l'entraînement.

3.2.2. Résultats

3.2.2.1. Protocole.

Dans chaque expérience présentée à la section 3.2.2, nous entraînons un modèle de diffusion sur le jeu de donné correspondant, puis l'évaluons sur l'ensemble de validation. Les modèles de diffusion sont généralement évalués à partir de score issus de l'évaluation d'un réseau de neurones sur la sortie produite (score FID [**26**], score IS [**4, 53**], score CLIP [**25**], etc.). Néanmoins, comme la tâche est différente (1D avec structure contraignante en opposition à des images), et que la sortie prédite doit correspondre plus exactement à la sortie attendue (CATS), nous allons évaluer les résultats autrement. Nous souhaitons que les prédictions soient :

- Non biaisées, c'est-à-dire que pour chaque firme, il doit y avoir autant de sur-évaluation que de sous-évaluation;
- D'erreur faible, c'est-à-dire que pour chaque firme, la prédiction doit être similaire à la valeur du modèle de référence CATS;
- Variables, c'est-à-dire que selon différentes graines, les prédictions varient.

En particulier, un modèle qui prédirait exactement les résultats de CATS ne serait pas un bon modèle, car il n'introduirait pas de variabilité. Ce constat rend difficile l'interprétation des deux derniers critères : nous souhaitons de la variation, mais il faut qu'elle soit contrôlée. Le premier critère est sans doute le plus important car il est une condition nécessaire pour que le modèle puisse reproduire le comportement de CATS. Notons que l'analyse doit se faire par firmes car le nombre de travailleurs et celui de firmes sont fixes. En effet, si la firme A a un employé de plus que dans le modèle CATS selon la prédiction, alors il existe une firme B (ou la catégorie des sans-emplois) qui en a un de moins. Les tableaux et graphiques de résultats présentent une analyse par type de firmes, soit sans-emplois (s.e., ou firme #0), firmes-C (C, ou firmes #1-100) et firmes-K (K, ou firmes #101-120).

Détaillons maintenant les hyperparamètres du modèle de diffusion commun à toutes les expériences. Typiquement, le nombre d'étapes de diffusion est fixé à 1000. Puisque la tâche est plus simple que celle de génération d'images (en termes de dimensions), nous nous contentons de prendre un nombre d'étapes de diffusion $T = 500$, et nous n'observons pas de diminution de performances. Nous utilisons le paramétrage $\boldsymbol{v}$ afin de palier au problème potentiel que les images peu bruités se voient attribuer un poids proche de 0 dans l'entraînement (voir sec. 1.8.4.1. En effet, considérant qu'on utilise un nombre d'étapes de diffusion plus petit que typiquement, ce problème peut être plus important. Le reste des paramètres est usuel : une fonction de perte L_2, une séquence de β linéaire, un poids de guidage $w = 2$ (nous penchons vers moins de créativité attribuable au le guidage - $w = 1$ - plutôt que plus $w >> 1$, afin de rester plus proche des valeurs de CATS). Nous utilisons un régularisateur

L_2 de paramètre 0.00001 et une moyenne mobile exponentielle (EMA) de décroissance 0.995 afin d'aider l'apprentissage. Les réseaux de neurones ont une dimension cachée $d = 256$, accédée par interpolation. Le réseau personnalisé est constitué de quatre blocs dans la partie encodeur, et quatre blocs dans la partie décodeur. Le U-Net a quatre blocs descendants, un bloc au milieu, et quatre blocs montants. Une convolution initiale 7×7 avec *padding* égale à 3 porte le nombre de canaux à $d = 256$. Une convolution 1×1 porte le nombre final de canaux à 1. L'entrée du U-Net est la concaténation du triplet $(\boldsymbol{x}_{prec},H,\boldsymbol{x}_t)$, où $\boldsymbol{x}_t$ est la variable latente bruitée à force t. Les modules d'attention ont quatre têtes. Ces derniers choix ont été observés comme procurant un compromis adéquat entre la complexité des modèles et leur efficacité.

La phase d'entraînement s'étale sur 50000 pas d'entraînement, avec des lots de taille 256. Le taux d'apprentissage est fixé à 0.00008. La grandeur des lots pour la phase de validation est aussi de 256. Pour chaque exemple de validation, 20 échantillons sont générés à partir de graines distinctes. On procède à la validation en considérant des triplets $(\boldsymbol{x}_{prec},H,\boldsymbol{x})$ issus de l'ensemble de validation. Puis, pour chaque graine, on échantillonne $\boldsymbol{x}_T \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$, et on applique le modèle de diffusion au triplet $(\boldsymbol{x}_{prec},H,\boldsymbol{x}_T)$. On compare finalement le résultat généré avec $\boldsymbol{x}$.

3.2.2.2. Comparaison entre U-Net et réseau personnalisé.

Dans cette section, nous contrastons les résultats obtenus sur une série temporelle à partir de chacun des réseaux de neurones étudiés. Commençons par visualiser le résultat d'un exemple précis à la figure 3.6. Nous y voyons en rouge le nombre d'employés selon le modèle CATS, qui est notre objectif. En bleu se trouve les 20 prédictions par firme, possiblement confondues. Nous voyons que la catégorie des sans-emplois à plus de volatilité et que les valeurs sont plus grandes. Une raison à cela est que la catégorie des sans-emplois peut être vue comme le « reste » qui n'a pas réussi à être apparié, et est donc intrinsèquement plus volatil que les firmes qui ont des besoins et des demandes bien établis. En effet, la matrice d'information H n'a pas d'entrées pour cette catégorie. Pour les firmes, le modèle est une bonne approximation des valeurs de CATS.

Poursuivons notre étude en étudiant les trois graphiques qui décrivent la fonction de perte durant l'entraînement et qui sont associés au U-Net. Dans le graphique 3.7, nous observons tout d'abord que durant l'apprentissage, la fonction de perte diminue globalement sur l'ensemble d'entraînement, mais avec beaucoup de variation. Le premier aspect est positif : le modèle apprend durant l'entraînement. Le second aspect mérite une explication, à laquelle nous reviendrons après l'analyse du graphique 3.8.

Dans ce dernier, nous constatons qu'à la validation, la fonction de perte tend également à diminuer avec l'entraînement. Néanmoins, il y a une grande variation dans la capacité

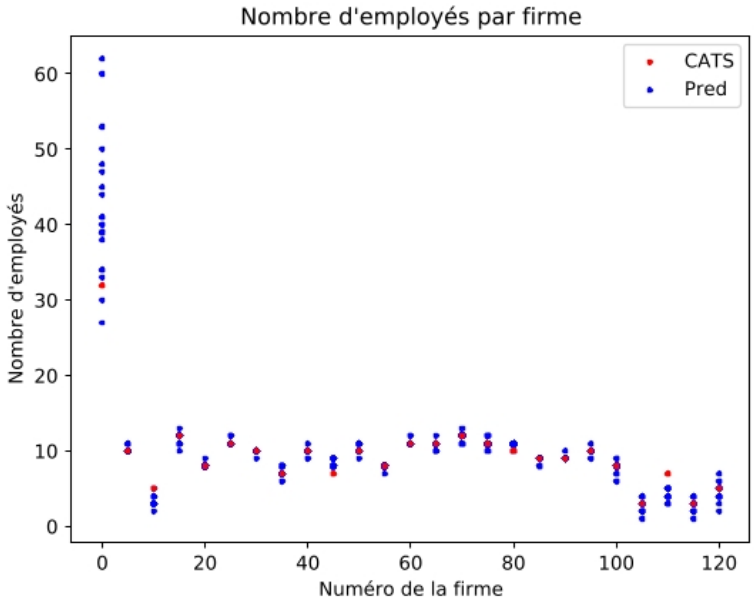


Fig. 3.6. Résultat de l'application du modèle de diffusion (U-Net) sur un exemple particulier. Une firme sur cinq est représentée par souci de lisibilité. 20 prédictions sont effectuées (et donc 20 points bleus par firme, possiblement confondus). Le modèle de diffusion génère des entrées sensiblement similaires aux valeurs de CATS.

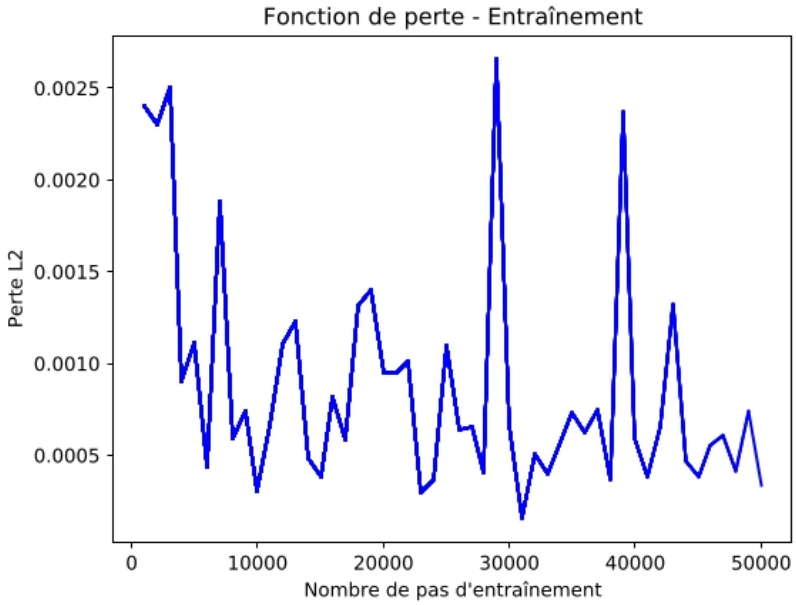


Fig. 3.7. U-Net - La fonction de perte L_2 sur les données d'entraînement (espace $\boldsymbol{v}$) diminue en moyenne avec l'entraînement.

du modèle à accomplir la tâche selon la quantité de bruit qui a été ajoutée au vecteur. En effet, pour un vecteur très bruité ($T = 495$), la fonction de perte reste proche de 1, tandis

que pour un vecteur peu bruité ($T = 5$), la fonction de perte est pratiquement de 0, ce qui est logique considérant la complexité de la tâche demandée. De plus, nous observons que l'apprentissage tend à faire diminuer la perte davantage pour les vecteurs les moins bruités. Cela est cohérent avec le choix de paramétrage utilisé. En effet, le paramétrage $\boldsymbol{v}$ donne plus de poids aux exemples moins bruités, tel que discuté à la section 1.8.4.1.

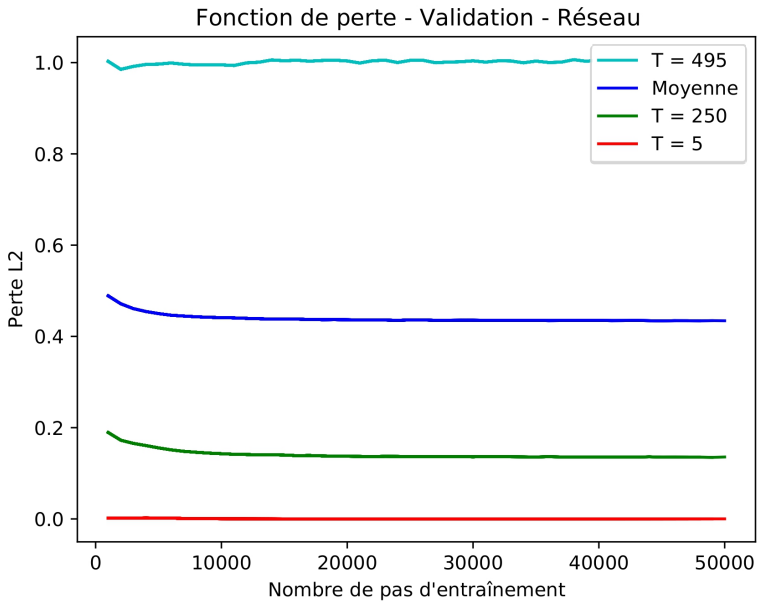


Fig. 3.8. U-Net - La fonction de perte L_2 sur les données de validation diminue avec l'entraînement. Les courbes indiquent la perte à différents stages du modèle de diffusion (espace $\boldsymbol{v}$).

Nous pouvons maintenant expliquer le comportement oscillatoire observé dans le graphique de gauche. La courbe de la fonction de perte en fonction du pas d'entraînement est une moyenne sur les exemples constituant un lot d'exemples. L'entraînement est effectué en échantillonnant aléatoirement les valeurs de t dans la chaîne de diffusion, pour chaque exemple. Ainsi, si un grand nombre d'exemple sont bruités avec une grande force, la fonction de perte sera plus grande. Néanmoins, l'aspect global de la courbe devrait tout de même diminuer, tel qu'on l'observe.

Finalement, le graphique 3.9 présente la fonction de perte après la prédiction finale du modèle de diffusion. Alors que le graphique 3.8 illustre la perte moyenne entre les prédictions $\boldsymbol{v}$ et les valeurs correspondantes de $\boldsymbol{v}$ calculées à partir de la valeur du modèle CATS et du bruit ajouté, ce graphique compare le vecteur généré par le modèle avec le résultat de CATS directement, dans l'espace paramétré par $\boldsymbol{x}$. Nous y voyons que le modèle parvient à généraliser à l'ensemble de validation.

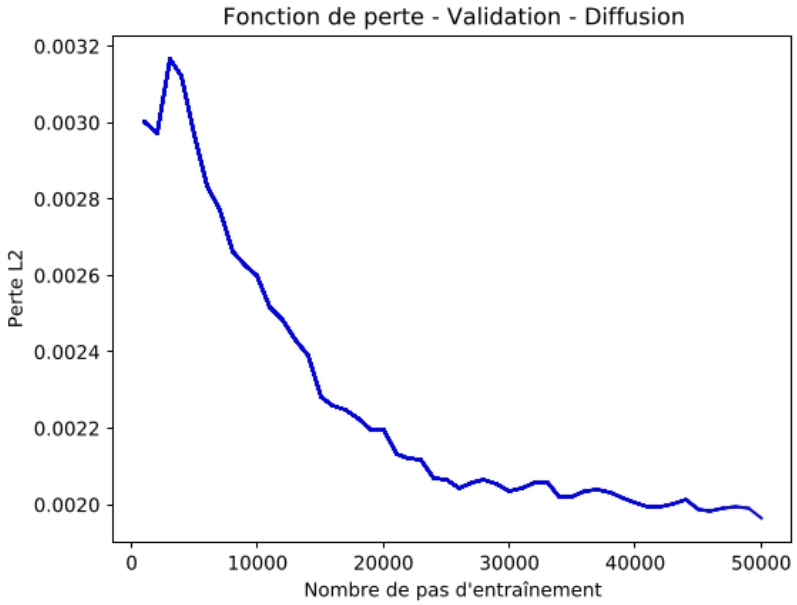


Fig. 3.9. U-Net - La fonction de perte L_2 sur les données de validation diminue avec l'entraînement. La courbe indique la perte après la prédiction finale du modèle de diffusion (espace $\boldsymbol{x}$).

Nous pouvons maintenant analyser le modèle personnalisé à l'aide des graphiques qui décrivent la fonction de perte durant l'entraînement. Nous observons en premier lieu (graphique 3.10) que la fonction de perte pour les données d'entraînement diminue avec le nombre de pas d'entraînement effectué. Toutefois, l'initialisation est bien moins intéressante : la perte est de plusieurs ordres de grandeur supérieur que pour le U-Net. À la fin de l'entraînement, le U-Net présente une perte d'entraînement de 0.0003 comparativement à 0.003 pour le modèle personnalisé, ce qui reste 10 fois moindre.

Dans le graphique 3.11, nous observons un comportement similaire au U-Net pour les pertes en $\boldsymbol{v}$ de l'ensemble de validation à différents stages du modèle de diffusion. Nous observons une fois de plus que le modèle au début de l'entraînement présente une perte plus importante. Un phénomène intéressant à remarquer est que la perte moyenne diminue au début, puis augmente. En observant plus précisément, nous remarquons que les courbes pour $T = 250$ et $T = 5$ sont strictement décroissantes. Cela renforce l'idée que le modèle s'adapte aux exemples peu bruités, quitte à obtenir des prédictions de moindre qualité sur les exemples plus bruités et une perte moyenne plus importante.

Le dernier graphique (3.12) confirme que l'apprentissage se généralise à l'ensemble de validation. De plus, contrairement à la perte d'entraînement qui est très différente entre les modèles, la perte de validation est similaire, avec une valeur de 0.0002 dans les deux cas. Rappelons que cette valeur n'est pas directement comparable avec celle de l'ensemble d'entraînement, n'étant pas calculée à partir du même paramétrage.

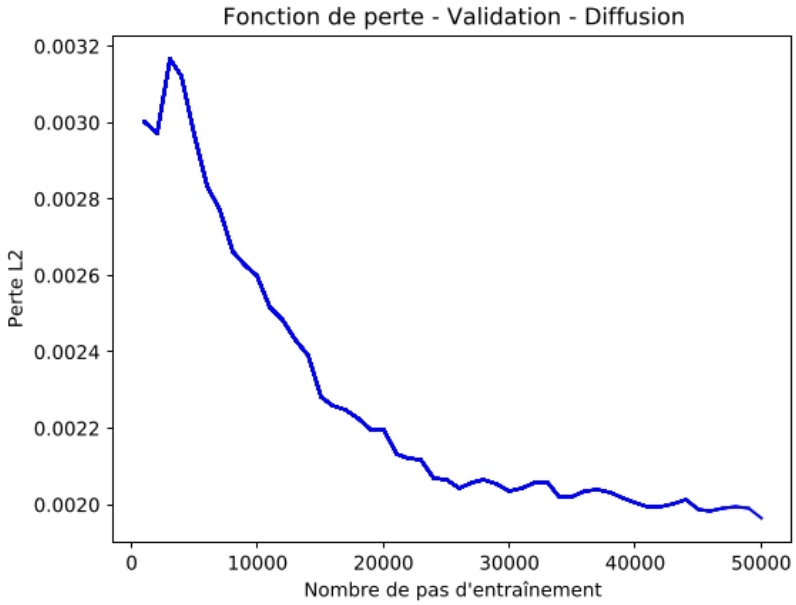


Fig. 3.10. Personnalisé - La fonction de perte L_2 sur les données d'entraînement (espace $\boldsymbol{v}$) diminue en moyenne avec l'entraînement. L'initialisation est moins efficace que pour le U-Net.

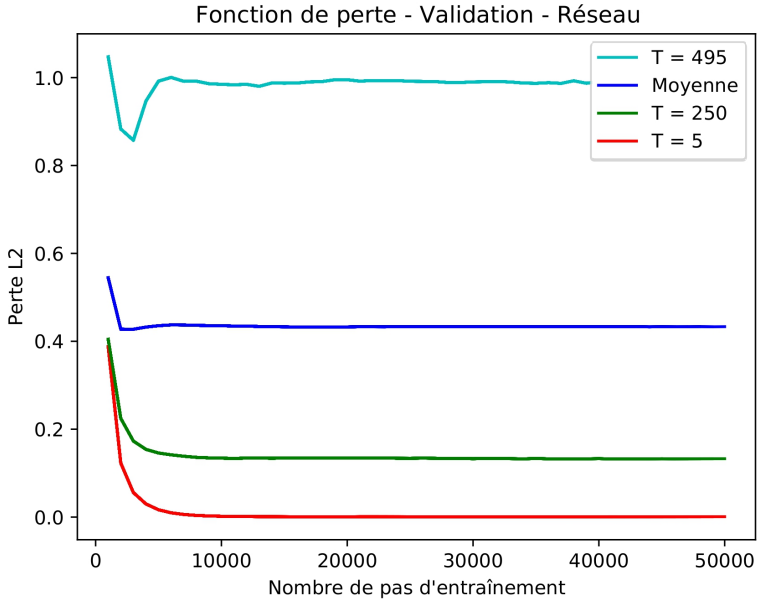


Fig. 3.11. Personnalisé - La fonction de perte L_2 sur les données de validation diminue avec l'entraînement.Le modèle s'adapte aux exemples peu bruités. Les courbes indiquent la perte à différents stages du modèle de diffusion (espace $\boldsymbol{v}$).

Pour pouvoir mieux interpréter les résultats et analyser les différences, nous étudions l'erreur absolue moyenne et cherchons à vérifier si les prédictions sont biaisées. L'étude qui

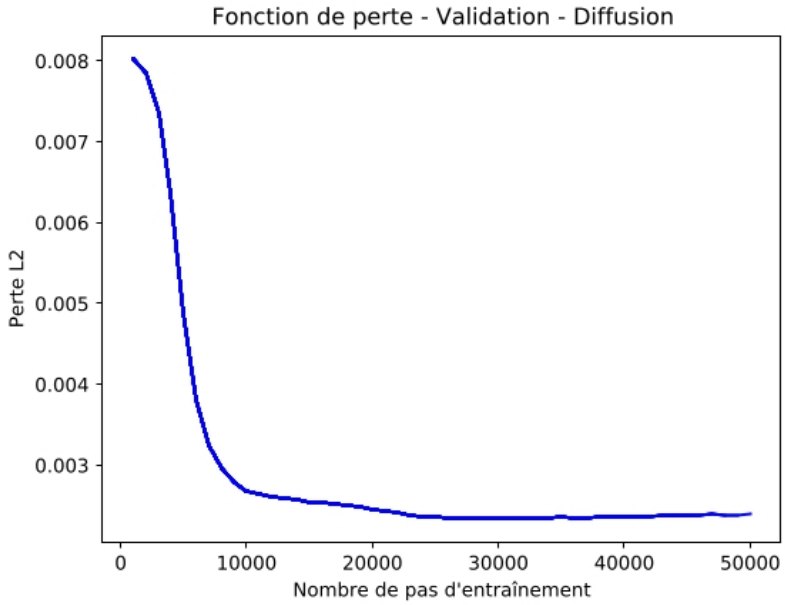
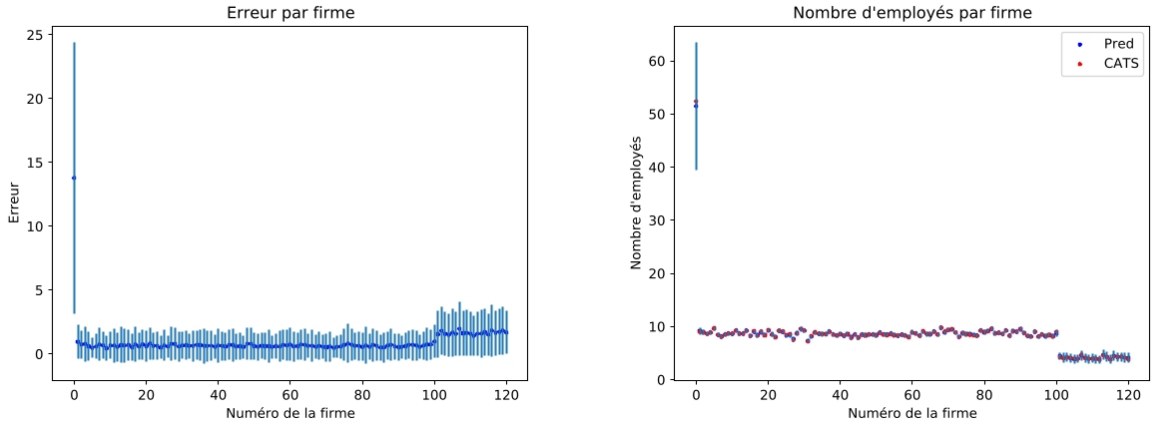


Fig. 3.12. Personnalisé - La fonction de perte L_2 sur les données de validation diminue avec l'entraînement. La perte de validation est similaire au U-Net. La courbe indique la perte après la prédiction finale du modèle de diffusion (espace $\boldsymbol{x}$).

suit est faite par firme. Inspiré par les résultats, nous distinguons trois groupes de firmes : les sans-emplois (s.e.), les firmes-C (C) et les firmes-K (K).



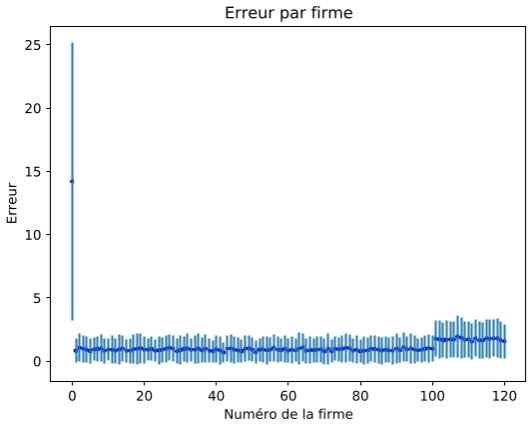
(a) Erreur absolue avec écart-type par firme.

(b) Comparaison entre la cible (CATS) et la moyenne des prédictions (Pred) avec écart-type par firme.

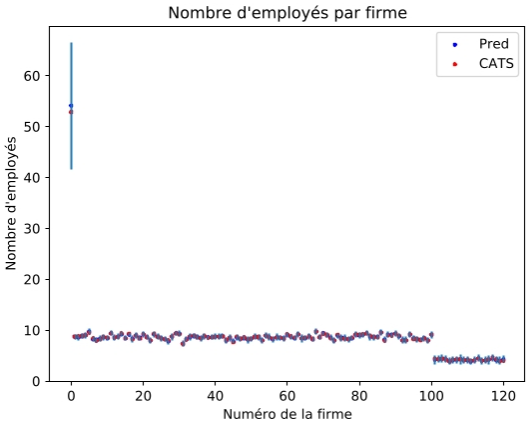
Fig. 3.13. U-Net - Le modèle de diffusion apprend la distribution des données d'entraînement. Le type de firme influence sur l'erreur.

Commençons par discuter des graphiques pour le U-Net, à la figure 3.13. Le graphique de gauche présente l'erreur absolue moyenne (selon les graines et les exemples) par firme. On

constate immédiatement que la catégorie des sans-emplois est la plus complexe, et que les différences intra-catégories sont faibles. À la figure de droite, nous observons que le modèle s'enligne très bien sur les données de CATS. Le modèle est donc non biaisé, ce qui est notre principal critère d'évaluation. En d'autres mots, l'erreur observée dans le graphique de gauche est répartie également en sous-évaluation et en sur-évaluation. De plus, nous observons que la principale variabilité provient de la catégorie des sans-emplois en relation avec les firmes-K. Les firmes-C sont quant à elle presque exactement prédite. Le modèle personnalisé, dont les graphiques sont présentés à la figure 3.14, a le même comportement que le U-Net.



(a) Erreur absolue avec écart-type par firme.



(b) Comparaison entre la cible (CATS) et la moyenne des prédictions (Pred) avec écart-type par firme.

Fig. 3.14. Personnalisé - Le modèle de diffusion apprend la distribution des données d'entraînement de façon similaire au U-Net.

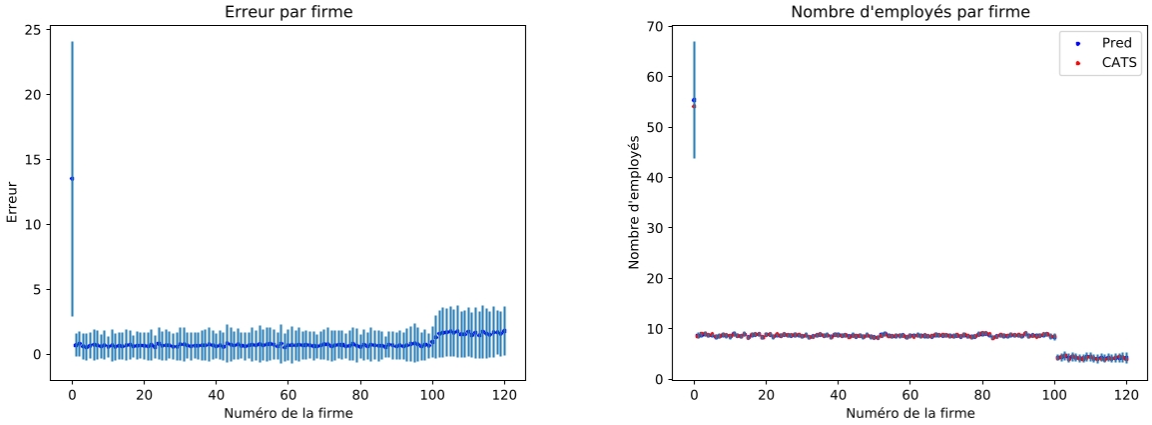
Nous comparons les valeurs précises dans le tableau 3.7. Nous observons que le U-Net obtient des résultats légèrement plus précis que le modèle personnalisé, bien que les deux soient très similaires. Notons toutefois que le modèle personnalisé est plus rapide à entraîner. Nous ne tirons pas de conclusion pour autant, car aucune tentative d'optimisation en ce sens n'a été faite. Puisque le U-Net obtient des résultats plus précis, les expériences subséquentes seront réalisées avec ce dernier.

Modèle	Erreur abs.			Biais		
	s.e.	C	K	s.e.	C	K
U-Net	13.80 ± 10.62	0.65 ± 1.13	1.65 ± 1.79	-0.83 ± 11.99	0.01 ± 0.40	0.01 ± 0.87
Pers.	14.22 ± 10.97	0.92 ± 1.03	1.73 ± 1.47	1.23 ± 12.45	-0.01 ± 0.56	-0.01 ± 0.71

Tableau 3.7. Comparatif de l'erreur abs. et du biais entre le U-Net et le modèle personnalisé. Le U-Net est légèrement plus précis que le modèle personnalisé.

3.2.2.3. Apprentissage à partir de plusieurs séries temporelles.

Dans cette section, le jeu de données est élargi pour inclure 10 séries temporelles plutôt qu'une. La figure 3.15 présente les résultats sous forme de graphiques. Les résultats sont similaires à ceux de la section précédente. Nous observons que la variabilité intra type d'emplois est plus faible que lorsqu'une seule série est considérée.



(a) Erreur absolue avec écart-type par firme.

(b) Comparaison entre la cible (CATS) et la moyenne des prédictions (Pred) avec écart-type par firme.

Fig. 3.15. U-Net - Le modèle de diffusion s'adapte à plusieurs séries temporelles.

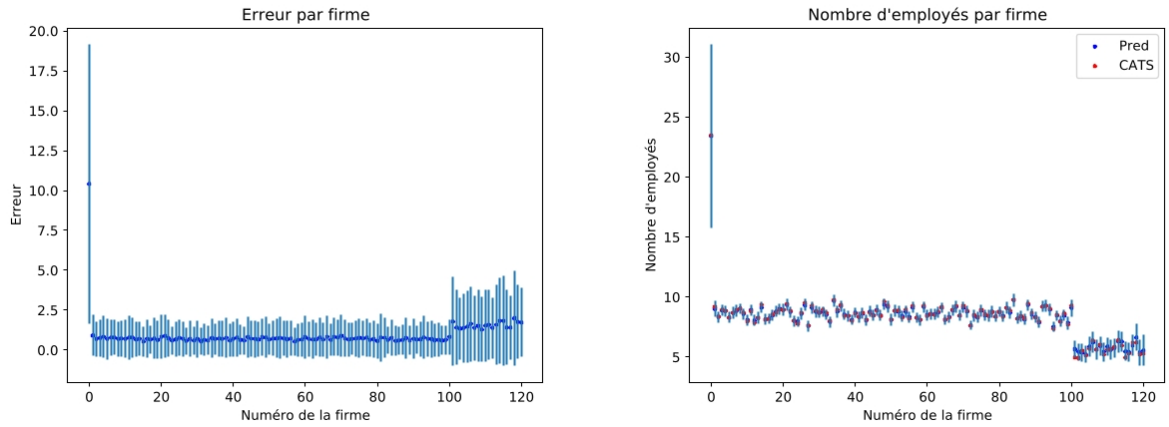
Le tableau 3.8 présente les résultats des deux approches à fin de comparaison. Les résultats très similaires permettent de conclure que le modèle est en mesure de s'adapter à différentes séries issues de la même distribution d'hyperparamètres. Nous observons que l'erreur est diminuée, tandis que le biais est augmenté. Cela est cohérent : le plus grand nombre d'exemples aide à l'apprentissage et diminue l'erreur, mais la plus grande diversité d'exemples amplifie le biais.

# séries	Erreur abs.			Biais		
	s.e.	C	K	s.e.	C	K
1	13.80 ± 10.62	0.65 ± 1.13	1.65 ± 1.79	-0.83 ± 11.99	0.01 ± 0.40	0.01 ± 0.87
10	13.50 ± 10.58	0.67 ± 1.12	1.60 ± 1.85	1.24 ± 11.63	-0.02 ± 0.45	0.04 ± 0.85

Tableau 3.8. Comparatif de l'erreur abs. et du biais entre le scénario à 1 série et celui à 10 séries. Les deux cas ont des performances similaires. L'erreur diminue et le biais augmente dans le cadre du scénario à 10 séries.

3.2.2.4. Adaptation à différents scénarios de CATS.

Nous explorons ici l'adaptabilité du modèle à des séries temporelles issues de scénarios distincts, chacun avec ses propres hyperparamètres. Nous observons dans le graphique de droite de la figure 3.16 que la distribution des valeurs de CATS est différente des cas précédents. Il y a une grande variabilité intra-catégories de firmes et, par exemple, le nombre de travailleurs sans-emplois est bien plus faible. Le modèle parvient à s'adapter adéquatement aux différents cas, comme en témoigne les deux graphiques. Cela valide l'utilité d'une approche d'apprentissage profond dans le cadre des modèles ABM, car l'apprentissage d'hyperparamètres variés est nécessaire.



(a) Erreur absolue avec écart-type par firme.

(b) Comparaison entre la cible (CATS) et la moyenne des prédictions (Pred) avec écart-type par firme.

Fig. 3.16. U-Net - Le modèle de diffusion apprend la distribution des données de CATS issues de séries temporelles d'hyperparamètres perturbés malgré la plus grande variabilité.

Le tableau 3.9 rapporte les valeurs précises. On y voit des valeurs similaires, exception faite de la catégorie des sans-emplois, où les valeurs d'erreurs et les écart-types sont plus faibles. Cela s'explique par le fait que la valeur réelle est également plus faible. L'erreur relative est en fait plus importante, avec 26.3% pour les paramètres standards contre 44.4% pour les paramètres perturbés. Néanmoins, nous considérons cette erreur relative élevée comme étant acceptable pour deux raisons. Tout d'abord, le biais reste proche de 0. Ensuite, nous pouvons interpréter la catégorie des sans-emplois comme le « reste » : chaque employé cherche du travail, mais d'une fois à l'autre, il peut y avoir une grande variance. Ainsi, cette erreur importante est moins grave qu'une erreur importante au niveau des firmes.

Params.	Erreur abs.			Biais		
	s.e.	C	K	s.e.	C	K
Std.	13.80 ± 10.62	0.65 ± 1.13	1.65 ± 1.79	-0.83 ± 11.99	0.01 ± 0.40	0.01 ± 0.87
Variés	10.41 ± 8.75	0.71 ± 1.11	1.57 ± 2.31	-0.06 ± 7.66	-0.02 ± 0.47	0.14 ± 0.85

Tableau 3.9. Comparatif de l'erreur abs. et du biais entre le scénario aux hyperparamètres standards et celui aux hyperparamètres variés.

Chapitre 4

Conclusion

Nous avons évalué deux méthodes d'apprentissage profond avec structure additionnelle. Les réseaux de *scattering* paramétrés ont été employés dans le contexte des données limitées. Les modèles de diffusion ont été utilisés dans le cadre du modèle à base d'agents CATS.

Dans un premier axe d'étude, nous avons constaté que les réseaux de *scattering* paramétrés ont des performances intéressantes sur les trois jeux de données de classification étudiés lorsque les données sont limitées. Nous avons vu qu'un compromis entre l'imposition d'une structure stricte en la présence de filtres pré-établis et la flexibilité en la possibilité de l'apprentissage des paramètres des ondelettes générant les filtres est souhaitable. En effet, nous avons démontré que les réseaux de *scattering* paramétrés performent mieux que ceux non paramétrés. Nous avons étudié l'efficacité de la construction canonique de la banque de filtres et avons constaté que des banques de filtres adaptés aux jeux de données permettent d'améliorer l'apprentissage. Nous avons tout de même constaté que la construction canonique est un bon point de départ grâce à sa propriété de repère ajusté, particulièrement lorsque les données sont limitées. Nous avons étudié l'importance de couvrir l'ensemble des orientations à travers la construction équivariante qui ajoute une contrainte (et donc une structure) supplémentaire et concluons qu'il s'agit d'un point au minimum non préjudiciable et possiblement bénéfique pour les données limitées, et d'un point légèrement préjudiciable lorsque les données sont nombreuses. Nous avons vu qu'il n'est pas souhaitable d'enlever toute contrainte et d'apprendre les filtres à l'aide du paramétrage par pixels.

Nous avons également vu que les filtres appris sont spécifiques au jeu de données. Nous avons pu vérifier expérimentalement la propriété de robustesse aux petites déformations lisses. Nous avons constaté qu'augmenter le nombre de filtres permet d'améliorer les résultats sur CIFAR-10, mais uniquement pour la version paramétrée de la transformée de *scattering*. L'utilisation de la fonction de perte cosinus permet d'améliorer les résultats sur KTH-TIPS2, mais pas sur CIFAR-10 ni sur COVIDx-CRX2, et son utilisation est donc recommandée uniquement sur certains jeux de données.

Nous pouvons énoncer trois limitations principales à la recherche sur les réseaux de *scattering* paramétrés. Premièrement, l'implémentation des réseaux de *scattering* paramétrés est pour l'instant limité à deux dimensions; il serait possible de l'élargir. Deuxièmement, l'analyse n'intègre pas de comparaisons avec des modèles pré-entrainés. Une telle analyse pourrait être menée dans de futurs travaux. Troisièmement, les bonnes performances des réseaux de *scattering* paramétrés sont vraiment restreintes au régime des données limitées. En effet, nous avons vu par exemple que sur CIFAR-10, les performances du WRN-16 dépasse celle des réseaux de *scattering* paramétrés dès que 1000 images sont utilisées dans l'ensemble d'entraînement. L'apprentissage paramétré que nous proposons est une innovation qui permet de mieux apprendre à partir des jeux de données de petite taille, mais il serait intéressant de poursuivre dans cette voie de relâchement progressif de contraintes afin d'obtenir des modèles efficaces pour des jeux de données de l'ordre de plusieurs milliers de données.

Finalement plusieurs questions subsistent concernant la mise en application des réseaux de *scattering* paramétrés. Quel est l'impact des couches de sous-échantillonnage dans les réseaux de *scattering* paramétrés? Serait-il possible et efficace d'avoir une construction similaire à la construction équivariante pour d'autres paramètres, telle l'échelle? Est-ce que dans l'initialisation équivariante, les filtres doivent être alignés avec les axes cartésiens? Une investigation plus en profondeur des réseaux de *scattering* paramétrés permettrait de cerner ces questions et plusieurs autres.

Dans un second axe d'étude, nous avons étudié les modèles de diffusion appliqués au marché de l'emploi du modèle CATS et concluons qu'il s'agit d'une approche efficace pour cette tâche. Nous statuons que l'utilisation d'une chaîne de prédictions de plus en plus précises permet de générer efficacement des données, même dans le cadre de l'apprentissage d'une unique série temporelle de quelques centaines de données. Nous avons vérifié qu'autant le U-Net que le modèle personnalisé sont efficaces. Nous avons testé différents scénarios. Dans un premier temps, nous avons vérifié que le modèle de diffusion parvient à intégrer plusieurs séries temporelles dans le même modèle. Dans un second temps, nous avons vérifié qu'il s'adaptait à des variations d'hyperparamètres. Nous concluons que le modèle de diffusion est un bon outil pour les tâches de génération de données requises dans le cadre de processus de recherche et d'appariement dans l'exercice de modélisation par réseaux de neurones des modèles à base d'agents.

Une des principales limitations est le manque de critère d'évaluation objectif. En effet, il serait pertinent d'adapter les différents scores (FID, CLIP, etc.) pour cette tâche. L'adaptation devrait permettre de respecter les différents objectifs d'évaluation fixés à la section 3.2.2.1. Une autre limitation est que l'entraînement est effectué à l'aide des données de CATS, lesquelles peuvent être erronées en soi, étant des données issues d'un modèle. Il serait par conséquent intéressant de vérifier si le transfert vers des données de la vie réelle est

possible. Dans le but plus restreint de vérifier que les modèles ABM peuvent être remplacés par des réseaux de neurones, cela ne constitue pas un problème.

Pour des recherches futures, différentes avenues sont possibles. Le modèle CATS en entier pourrait être modéliser. Pour ce faire, une avenue intéressante est l'utilisation de réseau de neurones en graphes afin de modéliser les interactions entre les agents. Ensuite, les modèles de diffusion pourraient être testés avec d'autres ABM. En ce qui concerne la modélisation à proprement parler, l'échantillonnage DDIM pourrait être utilisé et la technique de distillation serait pertinente pour limiter la durée d'échantillonnage. Les techniques utilisées pour la génération d'images à partir de croquis à l'aide de modèles de diffusion pourrait être transférées au présent contexte, où le croquis serait remplacé par le vecteur d'emploi au temps précédent. D'autres architectures de réseaux de neurones pourraient être explorées. Cette recherche n'étant qu'à ses débuts, ceci n'est qu'un aperçu des voies possibles.

Références bibliographiques

[1] T Assenza, P Colzani, D Delli Gatti et J Grazzini : Does fiscal policy matter? Tax, transfer, and spend in a macro ABM with capital and credit. *Industrial and Corporate Change*, 27(6):1069–1090, 05 2018.

[2] Tiziana Assenza, Domenico Delli Gatti et Jakob Grazzini : Emergent dynamics of a macroeconomic agent based model with capital and credit. *Journal of Economic Dynamics and Control*, 50(C):5–28, 2015.

[3] Jimmy Lei Ba, Jamie Ryan Kiros et Geoffrey E. Hinton : Layer normalization, 2016.

[4] Shane Barratt et Rishi Sharma : A note on the inception score, 2018.

[5] Bjorn Barz et Joachim Denzler : Deep learning on small datasets without pre-training using cosine loss. *In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1371–1380, 2020.

[6] Björn Barz et Joachim Denzler : Deep learning on small datasets without pre-training using cosine loss, 2019.

[7] Richard Bellman : Dynamic programming. *Science*, 153(3731):34–37, 1966.

[8] Nihar Bendre, Hugo Terashima Marín et Peyman Najafirad : Learning from few samples: A survey. arXiv:2007.15484, 2020.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever et Dario Amodei : Language models are few-shot learners, 2020.

[10] Joan Bruna et Stéphane Mallat : Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.

[11] Joan Bruna et Stéphane Mallat : Invariant scattering convolution networks, 2012.

[12] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin et Alexandr A. Kalinin : Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[13] Ayush Chopra, Ramesh Raskar, Esma S. Gel, Jayakumar Subramanian, Balaji Krishnamurthy, Santiago Romero-Brufau, Kalyan S. Pasupathy et Thomas C. Kingsley : Deepabm: Scalable and efficient agent-based simulations via geometric learning frameworks - a case study for covid-19 spread and interventions. *In Proceedings of the Winter Simulation Conference*, WSC '21. IEEE Press, 2022.

[14] Wikimedia Commons : Réseau de neurones formels de type perceptron multicouche, 2009.

[15] Wikimedia Commons : Fully connected convolutional neural network, 2015.

[16] Wikimedia Commons : Exemple de max-pool 2x2, 2022.

[17] Ekin D. Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan et Quoc V. Le : Autoaugment: Learning augmentation strategies from data. *In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 113–123, 2019.

[18] Mainak Debnath : Conv2d operation in tensorflow, 2022.

[19] Prafulla Dhariwal et Alex Nichol : Diffusion models beat gans on image synthesis, 2021.

[20] Michael Eickenberg, Georgios Exarchakis, Matthew Hirn, Stéphane Mallat et Louis Thiry : Solid harmonic wavelet scattering for predictions of molecule properties. *The Journal of chemical physics*, 148(24):241732, 2018.

[21] D.D. Gatti, S. Desiderio, E. Gaffeo, P. Cirillo et M. Gallegati : *Macroeconomics from the Bottom-up*. New Economic Windows. Springer Milan, 2013.

[22] S. Gauthier, B. Therien, L. Alsene-Racicot, M. Chaudhary, I. Rish, E. Belilovsky, M. Eickenberg et G. Wolf : Parametric scattering networks. *In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5739–5748, Los Alamitos, CA, USA, jun 2022. IEEE Computer Society.

[23] Hamed Hassanzadeh, Mahnoosh Kholghi, Anthony N. Nguyen et Kevin H. Chu : Clinical document classification using labeled and unlabeled data across hospitals. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, 2018:545–554, 2018.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras et Yejin Choi : Clipscore: A reference-free evaluation metric for image captioning, 2022.

[26] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler et Sepp Hochreiter : Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[27] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever et Ruslan R. Salakhutdinov : Improving neural networks by preventing co-adaptation of feature detectors, 2012.

[28] Jonathan Ho, Ajay Jain et Pieter Abbeel : Denoising diffusion probabilistic models, 2020.

[29] Jonathan Ho et Tim Salimans : Classifier-free diffusion guidance, 2022.

[30] Sergey Ioffe et Christian Szegedy : Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[31] A.G. Ivakhnenko, V.G. Lapa, V.G. Lapa et R.N. McDonough : *Cybernetics and Forecasting Techniques*. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1967.

[32] M. I. Jordan et T. M. Mitchell : Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[33] Diederik P. Kingma et Jimmy Ba : Adam: A method for stochastic optimization, 2017.

[34] Diederik P. Kingma, Tim Salimans, Ben Poole et Jonathan Ho : Variational diffusion models, 2023.

[35] Alex Krizhevsky, Ilya Sutskever et Geoffrey E. Hinton : Imagenet classification with deep convolutional neural networks, Apr 2014.

[36] Harold W Kuhn : The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[37] Bryan Lim, Sercan O. Arik, Nicolas Loeff et Tomas Pfister : Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2020.

[38] Stéphane Mallat : *A wavelet tour of signal processing*. Elsevier, 1999.

[39] Stéphane Mallat : Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.

[40] Sté phane Mallat : Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150203, apr 2016.

[41] Stéphane Mallat : Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150203, 2016.

[42] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik P. Kingma, Stefano Ermon, Jonathan Ho et Tim Salimans : On distillation of guided diffusion models, 2023.

[43] Vinod Nair et Geoffrey E. Hinton : Rectified linear units improve restricted boltzmann machines. *In Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814. Omnipress, 2010.

[44] Andrew Y. Ng : Feature selection, l1 vs. l2 regularization, and rotational invariance. *In Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 78, New York, NY, USA, 2004. Association for Computing Machinery.

[45] Alex Nichol et Prafulla Dhariwal : Improved denoising diffusion probabilistic models, 2021.

[46] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, Ben Glocker et Daniel Rueckert : Attention u-net: Learning where to look for the pancreas, 2018.

[47] Edouard Oyallon, Sergey Zagoruyko, Gabriel Huang, Nikos Komodakis, Simon Lacoste-Julien, Matthew Blaschko et Eugene Belilovsky : Scattering networks for hybrid representation learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2208–2221, 2018.

[48] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser et Björn Ommer : High-resolution image synthesis with latent diffusion models, 2022.

[49] Olaf Ronneberger, Philipp Fischer et Thomas Brox : U-net: Convolutional networks for biomedical image segmentation, 2015.

[50] Sebastian Ruder : An overview of gradient descent optimization algorithms, 2017.

[51] David E. Rumelhart, Geoffrey E. Hinton et Ronald J. Williams : Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[52] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet et Mohammad Norouzi : Photorealistic text-to-image diffusion models with deep language understanding, 2022.

[53] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen et Xi Chen : Improved techniques for training gans. *In* D. Lee, M. Sugiyama, U. Luxburg, I. Guyon et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[54] Tim Salimans et Jonathan Ho : Progressive distillation for fast sampling of diffusion models, 2022.

[55] Vikash Sehwag, Caner Hazirbas, Albert Gordo, Firat Ozgenel et Cristian Canton Ferrer : Generating high fidelity data from low-density regions using diffusion models. *In 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11482–11491, 2022.

[56] Anuj Shah : Through the eyes of gabor filter, Jun 2018.

[57] Rishu Shrivastava : Image classification, an overview, May 2021.

[58] Laurent Sifre et Stéphane Mallat : Rotation, scaling and deformation invariant scattering for texture discrimination. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1233–1240, 2013.

[59] Leslie N Smith et Nicholay Topin : Super-convergence: Very fast training of neural networks using large learning rates. *In Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612, 2019.

[60] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan et Surya Ganguli : Deep unsupervised learning using nonequilibrium thermodynamics. *In* Francis Bach et David Blei, éditeurs : *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 de *Proceedings of Machine Learning Research*, pages 2256–2265, Lille, France, 07–09 Jul 2015. PMLR.

[61] Jiaming Song, Chenlin Meng et Stefano Ermon : Denoising diffusion implicit models, 2022.

[62] Yang Song et Stefano Ermon : Generative modeling by estimating gradients of the data distribution. *In* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[63] Yang Song, Fan Zhang, Qing Li, Heng Huang, Lauren J O'Donnell et Weidong Cai : Locally-transferred fisher vectors for texture classification. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 4912–4920, 2017.

[64] Antti Tarvainen et Harri Valpola : Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results, 2018.

[65] Sik-Ho Tsang : Review-group norm (gn): Group normalization (image classification), Dec 2020.

[66] Arash Vahdat et Karsten Kreis : Improving diffusion models as an alternative to gans, part 1, 2022.

[67] V. Vapnik : Principles of risk minimization for learning theory. *In Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, page 831–838, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser et Illia Polosukhin : Attention is all you need, 2017.

[69] Linda Wang, Zhong Qiu Lin et Alexander Wong : Covid-net: A tailored deep convolutional neural network design for detection of covid-19 cases from chest x-ray images. *Scientific Reports*, 10(1):1–12, 2020.

[70] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov et Gabriel J. Brostow : Harmonic networks: Deep translation and rotation equivariance. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[71] Jason Yosinski, Jeff Clune, Yoshua Bengio et Hod Lipson : How transferable are features in deep neural networks?, 2014.

[72] Sergey Zagoruyko et Nikos Komodakis : Wide residual networks. *In Proceedings of the British Machine Vision Conference (BMVC)*, 2016.

[73] Xiaojin Zhu : Semi-supervised learning literature survey. *Comput Sci, University of Wisconsin-Madison*, 2, 07 2008.

Annexe A

Preuves liées au modèle de diffusion

A.1. Forme close d'une variable latente

Si $\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_{t-1}$, avec $\boldsymbol{\epsilon}_{t-1} \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})$ par récurrence, on obtient

$$\begin{aligned}
\boldsymbol{x}_t &= \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t}\boldsymbol{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} \\
&= \sqrt{\alpha_t}(\sqrt{\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_{t-1}}\boldsymbol{\epsilon}_{t-1}) + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-2} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\boldsymbol{\epsilon}_{t-1} + \sqrt{1-\alpha_t})\boldsymbol{\epsilon}_{t-2} \\
&= \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} \quad (*) \\
&= ... \\
&= \sqrt{\alpha_t\alpha_{t-1}...\alpha_0}\boldsymbol{x}_0 + \sqrt{1-\alpha_t\alpha_{t-1}...\alpha_0}\bar{\boldsymbol{\epsilon}}_0 \\
&= \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\bar{\boldsymbol{\epsilon}}_0.
\end{aligned}$$

(*) Si $\boldsymbol{X} \sim \mathcal{N}(\mathbf{0},\sigma_X^2\boldsymbol{I})$, $\boldsymbol{Y} \sim \mathcal{N}(\mathbf{0},\sigma_Y^2\boldsymbol{I})$ et $\boldsymbol{Z} = \boldsymbol{X} + \boldsymbol{Y}$, alors $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0},(\sigma_X^2 + \sigma_Y^2)\boldsymbol{I})$.

A.2. Justification de la fonction de perte

Pour justifier le choix de la fonction de perte, nous allons optimiser la majoration variationnelle sur le logarithme négatif de la vraisemblance (en anglais, *variationnal bound of the log likelihood*) [**60**] :

$$
\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &\leq -\log p_\theta(\mathbf{x}_0) + D_{\mathrm{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)\|p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)) \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})/p_\theta(\mathbf{x}_0)}\Big] \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log p_\theta(\mathbf{x}_0)\Big] \\
&= \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big],
\end{aligned}
$$

où $D_{\mathrm{KL}}(\cdot\|\cdot)$ dénote la divergence de Kullback–Leibler.

En posant $L := \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big]$, on voit que

$$
L \geq -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0). \tag{A.2.1}
$$

On peut exprimer L comme suit :

$$
\begin{aligned}
L &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] \\
&= \mathbb{E}_q\Big[\log \frac{\prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_T)\prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=1}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \Big(\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\cdot\frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}\Big) + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log \frac{q(\mathbf{x}_T|\mathbf{x}_0)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\Big] \\
&= \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_T|\mathbf{x}_0)}{p_\theta(\mathbf{x}_T)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\Big] \\
&= \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^T \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}].
\end{aligned}
$$

Observons que L_T et L_{t-1} sont des termes qui calculent la divergence de Kullback–Leibler entre deux gaussiennes. Pour L_T, comme $q(\mathbf{x}_T|\mathbf{x}_0)$ n'a pas de paramètres et que $p_\theta(\mathbf{x}_T)) = \mathcal{N}(0,1)$ n'en a pas non plus, le terme est constant et peut être ignoré dans l'optimisation de L.

De son côté, L_{t-1} s'exprime comme

$$\begin{aligned} L_{t-1} &= D_{\mathrm{KL}}(\mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t,x_0),\tilde{\beta}_t\boldsymbol{I}) \parallel \mathcal{N}(x_{t-1}; \mu_\theta(x_t,t,c),\beta_t\boldsymbol{I}) \\ &= \frac{1}{2\beta_t}\|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t,\mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t,t)\|^2 + C, \end{aligned}$$

où C est une constante. Nous voyons apparaître une erreur au carré pondérée. Le poids dépend de la paramétrisation de $\boldsymbol{\mu}_\theta(\mathbf{x}_t,t)$. Néanmmoins, tel que suggéré par [**28**], l'entraînement omet usuellement de pondérée la fonction de perte outre qu'implicitement par le paramétrage. Le terme L_0 peut lui aussi être connecté à l'erreur au carré [**28**].

A.3. Probabilité conditionnelle renversée conditionnée sur l'entrée

Nous cherchons à montrer l'équation 1.8.6, que nous reproduisons ci-dessous. Elle énonce que

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\boldsymbol{x}_t,\boldsymbol{x}_0),\tilde{\beta}_t\boldsymbol{I}), \tag{A.3.1}$$

$$\text{où } \tilde{\boldsymbol{\mu}}_t(\boldsymbol{x}_t,\boldsymbol{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t. \tag{A.3.2}$$

Rappelons que $f(x) = e^{-\frac{1}{2}(ax^2+bx+c)}$ est la fonction de densité d'une gaussienne de moyenne $\mu = -\frac{b}{2a}$ et de variance $\sigma^2 = \frac{1}{a}$. On a :

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0) = q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1},\boldsymbol{x}_0)\frac{q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\ ; \qquad \text{par Bayes} \tag{A.3.3}$$

$$\propto e^{-\frac{1}{2}\left(\frac{(\boldsymbol{x}_t-\sqrt{\alpha_t}\boldsymbol{x}_{t-1})^2}{\beta_t} + \frac{(\boldsymbol{x}_{t+1}-\sqrt{\bar{\alpha}_{t-1}}\boldsymbol{x}_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(\boldsymbol{x}_t-\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0)^2}{1-\bar{\alpha}_t}\right)}\ ; \qquad \text{par 1.8.1 et 1.8.3} \tag{A.3.4}$$

$$= e^{-\frac{1}{2}\left(\left(\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}}\right)\boldsymbol{x}_{t-1}^2 - \left(\frac{2\sqrt{\alpha_t}}{\beta_t}\boldsymbol{x}_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\boldsymbol{x}_0\right)\boldsymbol{x}_{t-1} + g(\boldsymbol{x}_t,\boldsymbol{x}_0)\right)}, \tag{A.3.5}$$

où $g(\boldsymbol{x}_t,\boldsymbol{x}_0)$ est une fonction indépendante de $\boldsymbol{x}_{t-1}$ qui n'influence pas sur la moyenne et la variance de la distribution, tel qu'il est apparent en comparant l'expression avec celle dans le rappel ci-haut.

Ainsi, la variance est déterminée par

$$\tilde{\beta}_t = \frac{1}{\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}} = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{A.3.6}$$

et la moyenne est

$$\tilde{\boldsymbol{\mu}}_t(\boldsymbol{x}_t,\boldsymbol{x}_0) = \frac{\frac{\sqrt{\alpha_t}}{\beta_t}\boldsymbol{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\boldsymbol{x}_0}{\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}} = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\boldsymbol{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x}_t. \tag{A.3.7}$$