

Rapport du Projet de Fin d'Études

**Classification d'Images par Réseaux Diffusants
et Réseaux Convolutifs**

CentraleSupélec
Majeure SIR

Jonas Maison (jonas.maison@supelec.fr)
Antoine Deleuze (antoine.deleuze@supelec.fr)

Encadrants CentraleSupélec : Jean-Luc Collette et Michel Ianotto

25 mars 2019, 09:47

Table des matières

1	Introduction	3
2	Objectifs et Contexte	4
3	Réseaux de Neurones Convolutifs	5
3.1	Introduction	5
3.2	Composants de Base des Réseaux Convolutifs	7
3.3	État de l'Art et Innovations Récentes	9
4	Réseaux Diffusants	14
4.1	Théorie des Ondelettes	14
4.2	Transformée en Ondelettes 1D	15
4.3	Transformée Diffusante	16
4.3.1	Transformée Diffusante d'Ordre 2	16
4.3.2	Transformée Diffusante d'Ordre « m »	18
4.3.3	Propriétés Théoriques	18
4.4	Transformée Diffusante 2D	19
4.5	Applications des Réseaux Diffusants	23
5	Démarche et Travail Réalisé	25
5.1	Implémentation des Réseaux Diffusants	25
5.2	Expérimentations	27
5.2.1	Propriétés des Réseaux Diffusants	27
5.2.1.1	Perte d'information de la transformée diffusante	27
5.2.1.2	Impact des paramètres de la transformée diffusante	29
5.2.1.3	Impact de la taille de la base d'entraînement	34
5.2.1.4	Essai d'une représentation alternative des images	35
5.2.2	Comparaison des Réseaux Diffusants et des Réseaux Convolutifs	36
5.2.2.1	Temps de calcul et occupation mémoire GPU	36
5.2.2.2	Précision en fonction de la taille du set d'entraînement	37
5.2.2.3	Classification de textures	40
5.2.2.4	Imitation de réseau diffusant par un réseau convolutif	45
5.2.2.5	Comparaison des attributs extraits	46
5.2.2.6	Robustesse aux « adversarial examples »	49
5.2.2.7	Propriétés d'invariance	51
6	Conclusion	60

Chapitre 1

Introduction

Le projet de fin d'études « PFE » a pour but de mettre en application les enseignements reçus à CentraleSupélec, et en particulier ceux de la troisième année, dans le cadre d'un travail d'ingénieur débutant.

Le projet se déroule sur plus de 170 heures sur une durée de 4 mois et fait l'objet d'un rapport et d'une soutenance devant un jury composé de professeurs de Supélec.

Ce rapport présente donc le travail et les diverses expérimentations réalisés durant toute la durée du PFE.

En vue de rendre compte de manière structurée son déroulement, nous aborderons au préalable les objectifs ainsi que le contexte du projet. Ensuite, nous expliquerons la démarche choisie pour le mener à bien ainsi que les résultats obtenus. Enfin, nous conclurons sur le travail effectué tout au long du projet.

Chapitre 2

Objectifs et Contexte

Le projet porte sur l'application des réseaux diffusants « Scattering Networks » et des réseaux de neurones convolutifs « Convolutional Neural Networks » à un problème de classification d'images.

Les réseaux convolutifs sont des modèles de réseaux constitués de couches de neurones de prétraitement, qui analysent l'image à classifier au travers de fonctions de convolution et des couches de neurones de décision qui classifient les attributs extraits. Ils ont de nombreuses applications dans la reconnaissance d'images, de vidéos ou le traitement du langage naturel par exemple.

Les opérateurs de « scattering », introduits par Stéphane Mallat en 2010 [1], utilisent comme noyau de convolution des filtres en forme d'ondelettes, et comme opérateur non linéaire le module. On construit ainsi pour les images des représentations invariantes, stables et caractéristiques : ces opérateurs sont stables par déformation, ce qui les rend très intéressants pour l'analyse d'image en vision par ordinateur.

Le but du projet est donc de comparer les performances des deux modèles de réseaux sur l'analyse d'image, et en particulier la classification d'images.

Chapitre 3

Réseaux de Neurones Convolutifs

Dans cette partie nous abordons les réseaux de neurones convolutifs « CNNs », leurs fonctionnements, leurs intérêts dans l’analyse d’image ainsi que l’évolution des architectures de CNNs développées jusqu’à nos jours.

3.1 Introduction

Les réseaux neuronaux convolutifs sont très similaires aux réseaux neuronaux ordinaires : ils sont composés de neurones qui ont des poids et des biais qui peuvent être appris. Chaque neurone reçoit des entrées, réalise un produit scalaire et applique ensuite une non-linéarité. L’ensemble du réseau continue aussi d’exprimer une seule fonction de score différentiable : des pixels de l’image brute d’un côté, aux scores de classe de l’autre.

Cependant, l’utilisation de couches entièrement connectées pour classer les images ne prend pas en compte la structure spatiale de celles-ci, les perceptrons multicouches sont donc très inefficaces en terme de nombre de paramètres et en temps de calcul. La particularité des CNNs réside dans la façon dont les connexions entre les neurones sont structurées.

D’après les premiers travaux de Hubel et Wiesel sur le cortex visuel du chat et du singe [2] dans les années 1960, nous savons que le cortex visuel contient une disposition complexe de cellules. Ces cellules sont sensibles aux petites sous-régions du champ visuel, appelées champs réceptifs et ces sous-régions sont carrelées pour couvrir l’ensemble du champ visuel. Ces cellules agissent comme des filtres locaux sur l’espace d’entrée et sont bien adaptées pour exploiter la forte corrélation spatiale locale présente dans les images naturelles.

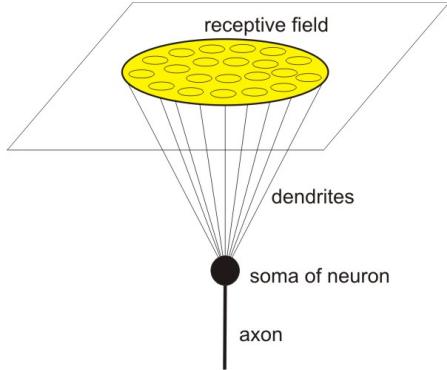


FIGURE 3.1 – Modèle de neurone avec son champ réceptif. Source.

Les CNNs s'inspirent donc du système du cortex visuel humain illustré sur la figure 3.2. Le cortex visuel primaire **V1** effectue la détection des bords à partir de l'entrée visuelle brute de la rétine. Le cortex visuel secondaire **V2** reçoit lui les caractéristiques trouvées par **V1** et extrait des propriétés visuelles simples comme l'orientation, la fréquence spatiale et la couleur. La zone visuelle **V4** gère les attributs d'objets plus complexes. Toutes les caractéristiques visuelles traitées sont transférées dans l'unité logique finale, le gyrus temporal inférieur « **IT** », pour la reconnaissance des objets.

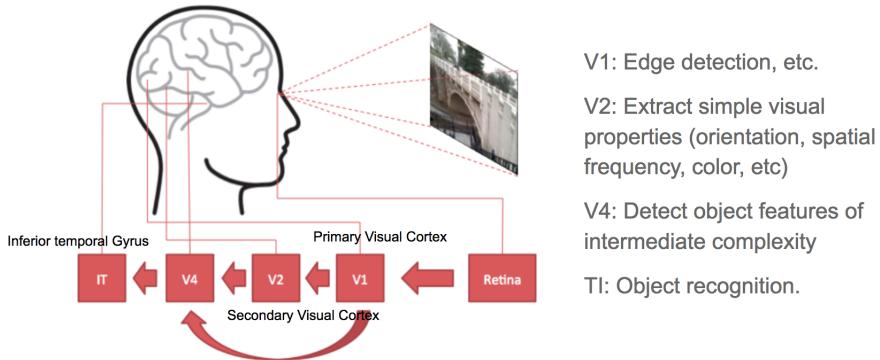


FIGURE 3.2 – Illustration du cortex visuel humain [3].

Une des premières architectures de CNNs, le Neocognitron [4] (1982), s'inspire fortement de cette structure hiérarchique.

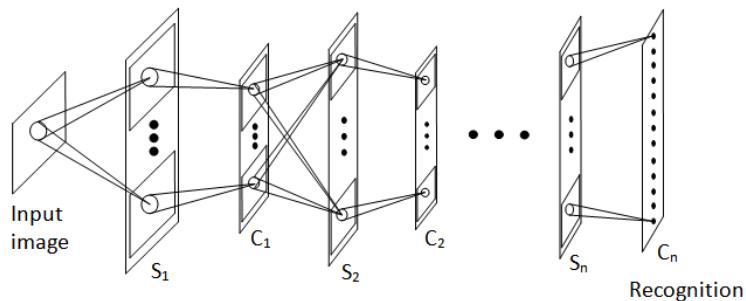


FIGURE 3.3 – Architecture du Neocognitron.

En 1998, [5] propose l'architecture LeNet. Les couches de convolutions et de pooling agissent comme les unités de cortex visuel **V1**, **V2** et **V4**. L'étape de la classification d'images se produit dans les dernières couches entièrement connectées qui classifient les caractéristiques extraites.

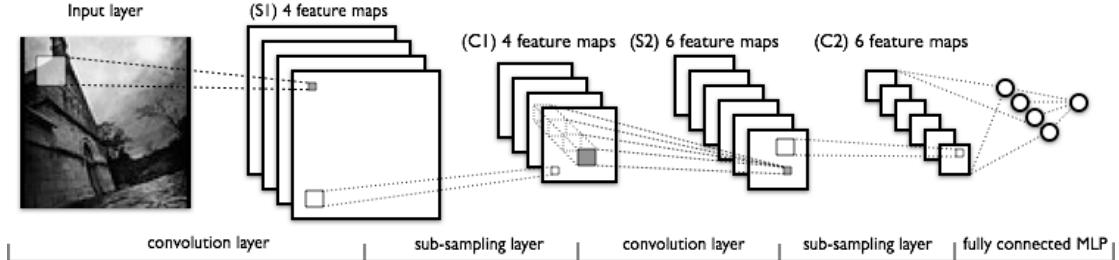


FIGURE 3.4 – Architecture LeNet.

3.2 Composants de Base des Réseaux Convolutifs

L'architecture de base d'un CNN de classification d'images est composée de différentes « briques » assemblées d'une certaine manière.

Couche de convolution C'est la composante clé des réseaux de neurones convolutifs. Son but est de repérer la présence d'un ensemble de « features » dans les images reçues en entrée. Pour cela, on fait « glisser » sur l'image d'entrée un filtre avec des poids entraînables, et on réalise le produit entre les pixels présents dans le champ récepteur du filtre et les poids de ce dernier. On obtient donc une nouvelle « feature map » pour chaque filtre défini dans la couche de convolution.

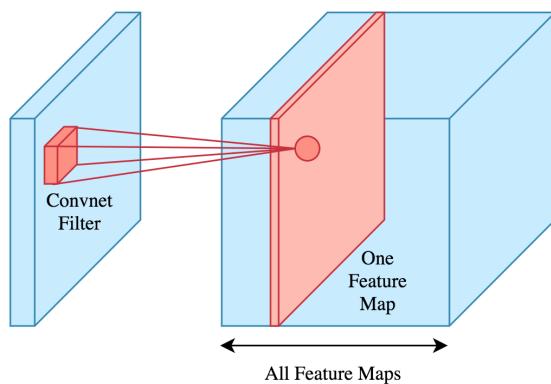


FIGURE 3.5 – Opération de convolution. Source.

Pour résumer, une couche de convolution accepte en entrée un tenseur de taille (L, H, P) où L désigne la largeur de la « feature-map », H la hauteur et P la profondeur. La sortie de la couche de convolution est un tenseur de taille (L', H', P') où P' est le nombre de filtres définis dans la couche de convolution, et $L' = \frac{L - K + 2P}{S} + 1$ où K est la largeur du filtre, P est le paramètre de « padding » et S est le pas de décalage des convolutions. H' se calcule de la même manière. Le choix des hyper-paramètres

K , S et P ainsi que le type de « padding » permettent d'obtenir différents types de convolutions [6].

Contrairement aux méthodes traditionnelles, les filtres ne sont pas pré-définis selon un formalisme particulier, mais appris par le réseau lors la phase d'entraînement. Ils sont initialisés puis mis à jour par rétro-propagation du gradient. C'est là toute la force des réseaux de neurones convolutifs : ceux-ci sont capables d'extraire eux-mêmes les attributs nécessaires pour classifier l'image en s'adaptant au problème posé via une fonction de coût.

De plus, les poids sont partagés sur toute l'image, ce qui réduit le nombre de paramètres et permet d'obtenir une invariance, c'est-à-dire qu'un objet déplacé dans l'image d'entrée décalera simplement les réponses correspondantes dans la « feature map » de la même manière.

Couche de pooling Ce type de couche est souvent placé entre les couches de convolution : il consiste à réduire la taille des « feature-maps » tout en préservant leurs caractéristiques importantes.

Pour cela, on découpe l'image en cellules régulières, puis on garde au sein de chaque cellule la valeur maximale (dans le cas du max-pooling). En pratique, on utilise souvent des cellules carrées de petite taille pour ne pas perdre trop d'informations. On obtient en sortie le même nombre de feature maps qu'en entrée, mais celles-ci sont bien plus petites.

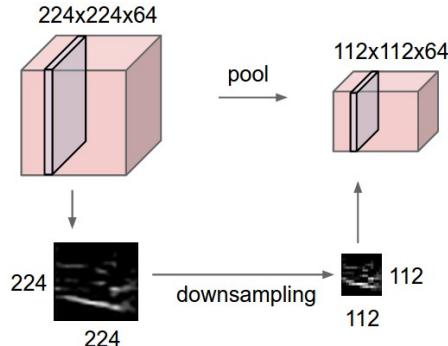


FIGURE 3.6 – Exemple de pooling. [Source](#).

Aujourd’hui, on comprend encore assez mal pourquoi la couche de pooling est efficace mais celle-ci permet en tout cas de réduire le nombre de calculs et l’occupation mémoire. Cette réduction de la taille spatiale est importante pour des raisons de calcul car la complexité des convolutions est quadratique en taille spatiale tout en étant linéaire en nombre de canaux. Il est aussi possible de réaliser un pooling grâce à des convolutions avec un pas de décalage de 2 par exemple.

L'inconvénient de cette opération est que l'on perd au fur et à mesure de la résolution spatiale ce qui peut poser des soucis pour certaines applications (détection d'objets, segmentation d'images, etc.).

Nous étudierons plus amplement les répercussions du pooling dans la partie 5.2.2.7.

Couche d'activation La couche d'activation est identique à celle utilisée dans les perceptrons multicouches. Elle permet d'introduire des non-linéarités car l'oracle que l'on cherche à apprendre est non-linéaire.

L'activation « ReLU » $y = \max(0, x)$ est devenue très populaire au cours des dernières années, car on a constaté qu'elle accélérerait considérablement l'entraînement par rapport aux fonctions *sigmoïde* ou *tanh* qui satureraient le gradient. Un autre avantage est qu'elle implique des opérations calculatoires simples par rapport aux exponentielles coûteuses.

La majorité des architectures récentes utilise désormais des couches de « batch-normalization » [7] après la couche de convolution et avant la couche d'activation de manière à accélérer l'entraînement du réseau.

Couche de neurones denses Dans les architectures de classification d'images, la couche finale possède N neurones où N est le nombre de classes. Il est donc nécessaire de convertir la « feature-map » avant cette couche finale en une couche de neurones dense. Les premiers réseaux re-dimensionnaient en général le tenseur de taille (L, H, P) en une couche de neurones de taille $L \times H \times P$. Les architectures récentes utilisent désormais des couches de « global average pooling » permettant de réduire drastiquement le nombre de paramètres.

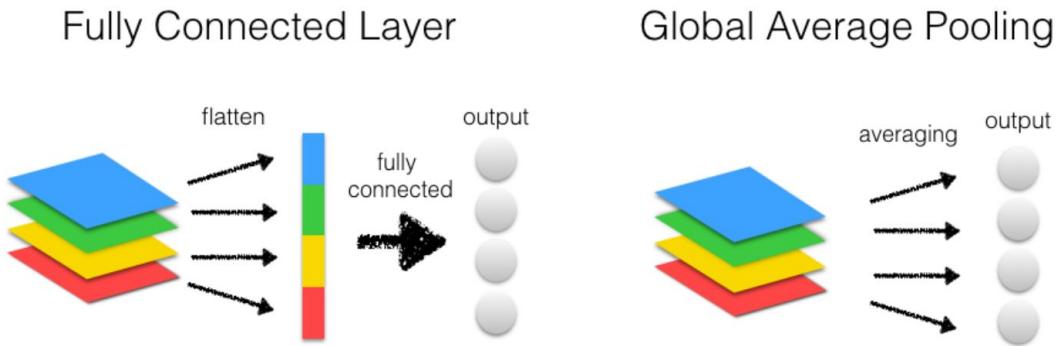


FIGURE 3.7 – Illustration d'une couche de neurones dense et du Global Average Pooling.
Source.

Nous étudierons plus amplement cette couche dans la partie 5.2.2.7.

3.3 État de l'Art et Innovations Récentes

AlexNet [8] La publication « AlexNet » est considérée comme l'un des articles les plus influents publiés dans le domaine de la vision par ordinateur, ayant stimulé la publication de nombreux autres articles utilisant les CNNs et les GPUs pour accélérer l'apprentissage profond.

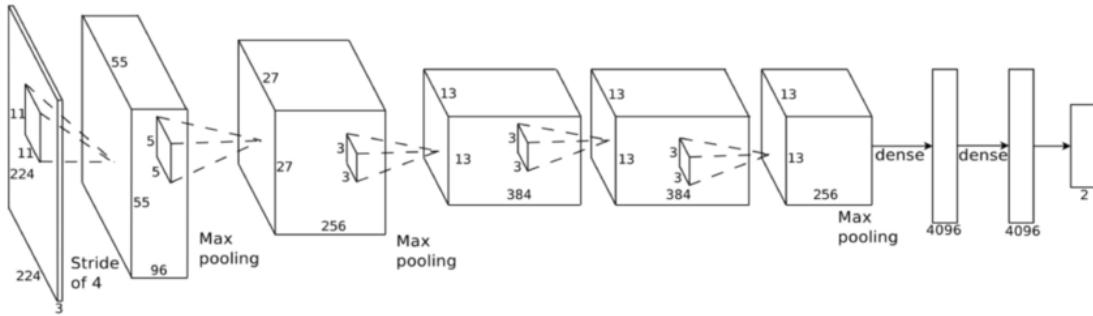


FIGURE 3.8 – Architecture AlexNet.

La contribution majeure à l'état de l'art de cette publication est le « dropout », une technique de régularisation pour réduire le sur-apprentissage dans les réseaux de neurones.

VGG [9] Le « Visual Geometry Group » (VGG), a été le finaliste de la compétition ILSVRC en 2014. La principale contribution de ce travail est de montrer que la profondeur du réseau est un élément critique pour obtenir une meilleure reconnaissance ou précision de classification dans les CNNs.

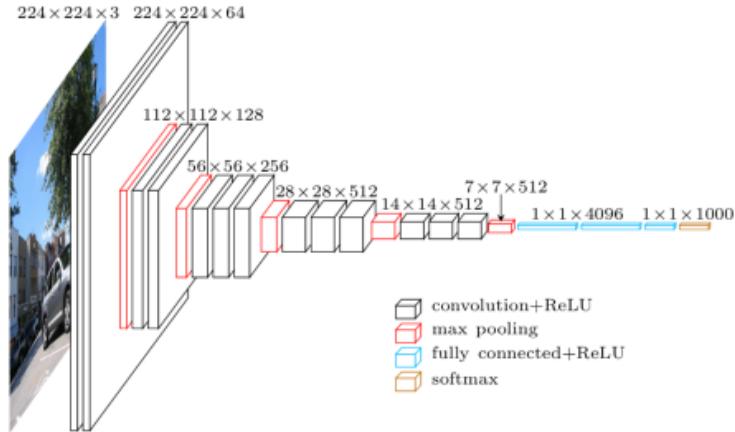


FIGURE 3.9 – Architecture VGG16.

ResNet [10] Les auteurs de cette publication constatent que plus les réseaux sont profonds plus ils sont durs à entraîner car le gradient se propage mal. L'idée consiste donc à ajouter des connexions résiduelles en court-circuitant les couches de convolutions de manière à mieux propager le gradient.

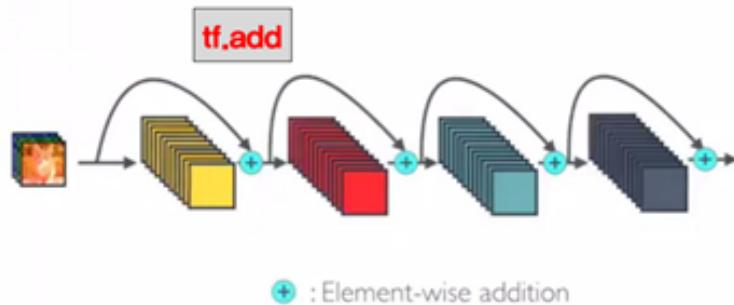


FIGURE 3.10 – Connexion résiduelle. [Source](#).

DenseNet [11] L’architecture DenseNet ressemble à un CNN classique :

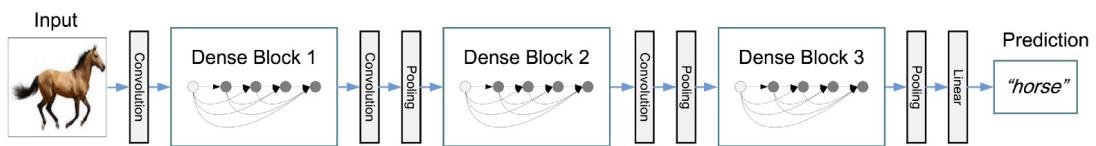


FIGURE 3.11 – Architecture DenseNet. [Source](#).

L’intérêt de ces réseaux réside à l’intérieur des blocs :

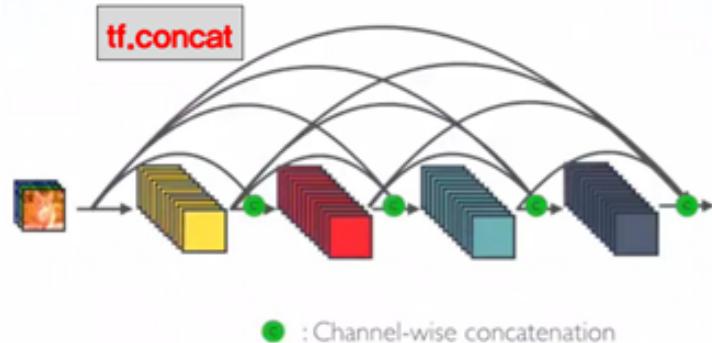


FIGURE 3.12 – Bloc dense d’un réseau DenseNet. [Source](#).

Toutes les couches d’un bloc DenseNet sont interconnectées entre elles, ce qui a plusieurs avantages :

- atténue le problème de disparition du gradient
- renforce et encourage la propagation des caractéristiques
- réduit drastiquement le nombre de paramètres.

Finalement, les réseaux DenseNet obtiennent des résultats similaires aux réseaux ResNet mais contiennent beaucoup moins de paramètres et sont donc plus rapides.

Convolutions déformables [12] [13] En 2017, Microsoft Research introduit un nouveau type de convolution avec des filtres déformés.

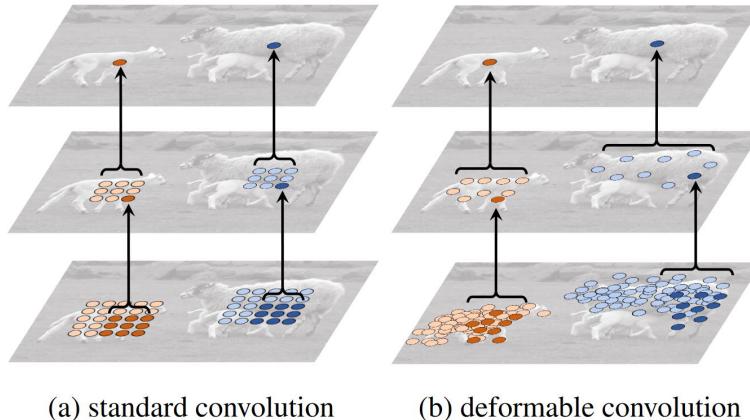


FIGURE 3.13 – Convolution standard et convolution déformée.

La forme du filtre de convolution n'est donc plus carrée ou rectangulaire mais est modifiée pendant l'entraînement du réseau. Les auteurs montrent que la convolution déformable est capable de modifier la forme du champ réceptif de manière à détecter des objets de grande/petite taille ou occultés par exemple.

Réseaux de compression et d'excitation [14] Introduits en 2018, les réseaux de compression et d'excitation (SENets) introduisent une nouvelle couche pour les réseaux CNN qui améliore l'interdépendance entre les canaux des « feature maps ».

Ils ont été utilisés lors du concours ImageNet et ont contribué à grandement améliorer les résultats par rapport à l'année précédente. En plus de ce gain de performance, ils peuvent être facilement ajoutés aux architectures existantes avec un coût de calcul supplémentaire négligeable.

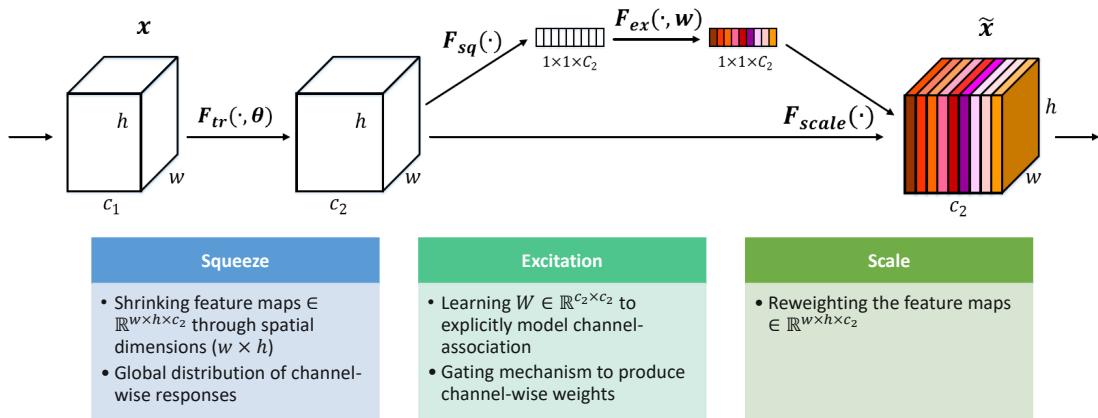


FIGURE 3.14 – Opération de compression et d'excitation.

Comparaison des architectures existantes La figure ci-dessous permet une visualisation des réseaux les plus connus ainsi que de leurs performances.

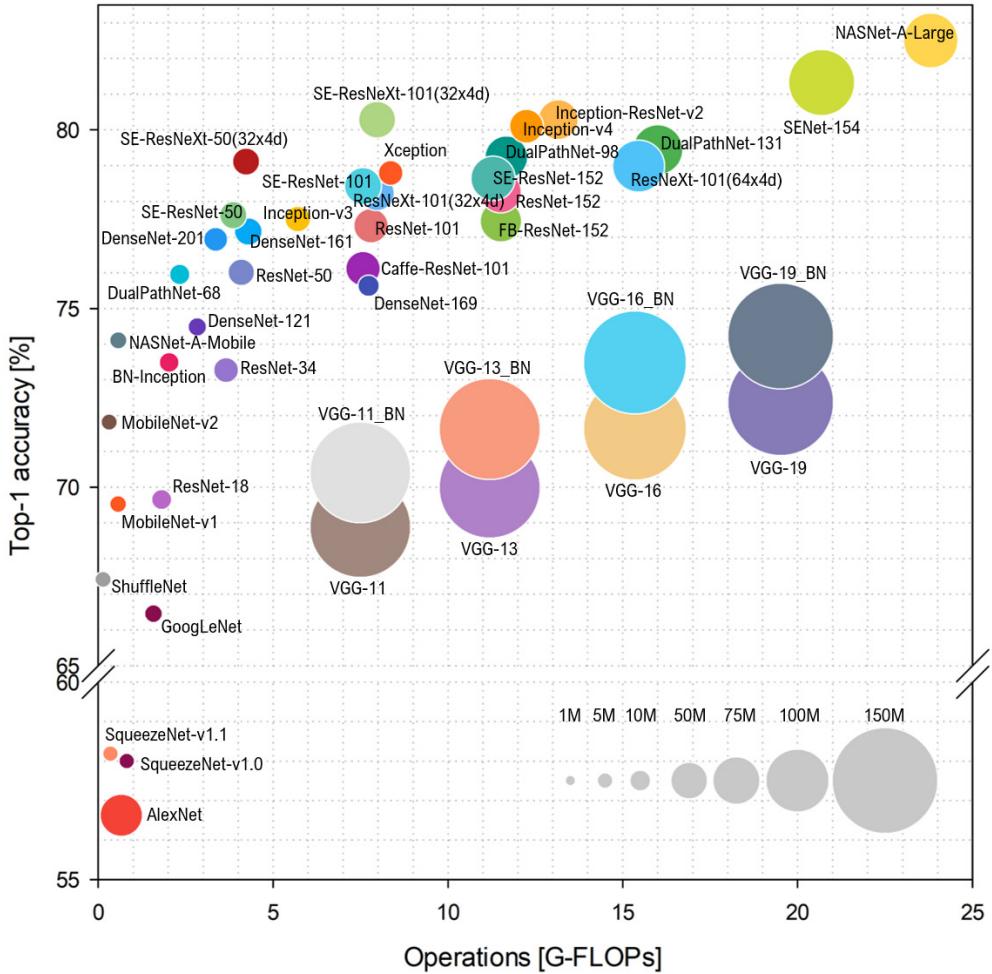


FIGURE 3.15 – Comparaison de la précision et du nombre d’opérations des différentes architectures sur le challenge ImageNet-1k[15].

Chapitre 4

Réseaux Diffusants

Dans cette partie nous abordons les réseaux diffusants aussi appelés « Scattering Networks », leur fonctionnement ainsi que leurs intérêts dans l’analyse d’image. Nous introduirons également la transformée en ondelettes ou « wavelet transform », l’opération à l’origine de la construction de ces réseaux.

4.1 Théorie des Ondelettes

Une ondelette est une fonction oscillante ψ présentant pour principales caractéristiques un support fini (ce qui explique le mot « ondelette », qui veut dire petite onde) avec une moyenne nulle. Il en existe plusieurs formes et nous nous intéresserons ici aux ondelettes 1D pour poser le cadre théorique de la transformée en ondelettes plus facilement.

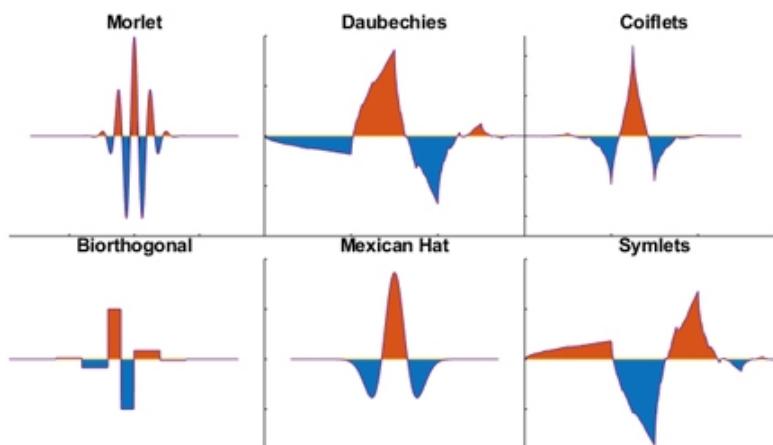


FIGURE 4.1 – Différentes ondelettes. [Source](#).

Une opération commune sur les ondelettes et qui servira à définir la transformée en

ondelettes est la dilatation. Ainsi on note ψ_λ l'ondelette dilatée de pulsation centrale λ .

$$\psi_\lambda(t) = \lambda\psi(\lambda t), \text{ et dans le domaine fréquentiel : } \hat{\psi}_\lambda(\omega) = \hat{\psi}\left(\frac{\omega}{\lambda}\right)$$

On remarque ainsi qu'une ondelette sera d'autant plus étirée dans le temps que son coefficient de dilatation λ sera petit.

En notant Q le nombre d'ondelettes par octave, c'est à dire le nombre d'ondelettes dont on peut intercaler la fréquence centrale entre une fréquence donnée et son double, on peut écrire le coefficient de dilatation $\lambda = 2^{k/Q}$ avec $k \in \mathbb{Z}$. On peut alors introduire la bande passante de ψ_λ qui est λ/Q . On note donc que plus λ sera petit, plus sa bande passante sera faible.

Pour résumer, en augmentant (resp. en diminuant) la valeur de λ :

- on contracte (resp. étire) l'ondelette dans le domaine temporel
- on décale la fréquence centrale vers les hautes (resp. basses) fréquences
- on augmente (resp. diminue) la bande passante de l'ondelette.

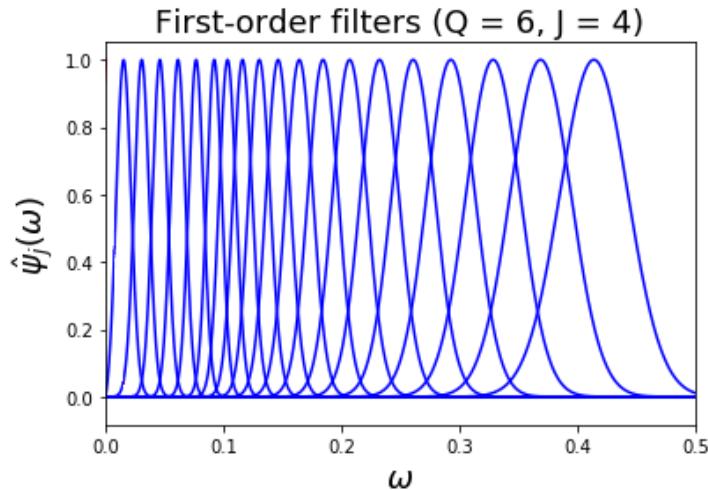


FIGURE 4.2 – Exemple d'ondelettes de Morlet avec une dilatation maximale d'un facteur $2^{J-1} = 8$ et $Q = 6$ ondelettes par octave soit un total de 19 ondelettes ($\lambda = 2^{k/6}$ pour $\lambda \in [1, 8]$).

Généralement, la valeur de Q est de 8 (pour les signaux audio), 10, 12, 16 ou 32. Plus Q est grand, plus l'analyse fréquentielle de notre signal sera fine.

4.2 Transformée en Ondelettes 1D

Maintenant que nous avons défini les ondelettes, nous pouvons introduire la transformée en ondelettes d'un signal 1D. Cette opération consiste grossièrement en une suite de convolutions du signal avec des ondelettes de différentes échelles visant à récupérer des informations à différentes fréquences.

Avant toute chose, il faut définir Q_1 la résolution fréquentielle en terme de nombre de filtres par octave. On peut alors introduire Λ_1 l'ensemble de coefficients de dilatation (donc de fréquences centrales) qui permettent de définir les ondelettes $(\psi_{\lambda_1})_{\lambda_1 \in \Lambda_1}$

Il nous faut également définir un filtre passe-bas ϕ couvrant le reste de la bande de fréquence non occupée par les autres ondelettes. La transformée en ondelettes d'un signal x est alors :

$$Wx = (x * \phi(t), x * \psi_{\lambda_1}(t))_{\lambda_1 \in \Lambda_1, t \in \mathbb{R}}$$

La convolution du signal avec des ondelettes qui font office de filtres passe-bandes va permettre de détecter des changements plus ou moins abrupts d'amplitude du signal. Ainsi, une ondelette contractée sera utile pour capturer des changements rapides quand une ondelette étirée sera utile pour la détection de changements lents.

4.3 Transformée Diffusante

4.3.1 Transformée Diffusante d'Ordre 2

La transformée diffusante 1D (ou scattering transform) d'ordre 2 nécessite l'introduction d'une nouvelle résolution fréquentielle Q_2 , d'un nouvel ensemble de coefficients de dilatation Λ_2 et d'un nouveau jeu d'ondelettes $(\psi_{\lambda_2})_{\lambda_2 \in \Lambda_2}$. On peut alors écrire la transformée diffusante :

$$Sx = (S_0x(t), S_1x(t), S_2x(t))_{t \in \mathbb{R}}$$

qui est la concaténation des coefficients calculés aux ordres 0, 1 et 2 :

$$\begin{aligned} S_0x(t) &= x * \phi(t) \\ S_1x(t) &= (|x * \psi_{\lambda_1}| * \phi(t))_{\lambda_1 \in \Lambda_1} \\ S_2x(t) &= (||x * \psi_{\lambda_1}| * \psi_{\lambda_2}| * \phi(t))_{\lambda_1 \in \Lambda_1, \lambda_2 \in \Lambda_2} \end{aligned}$$

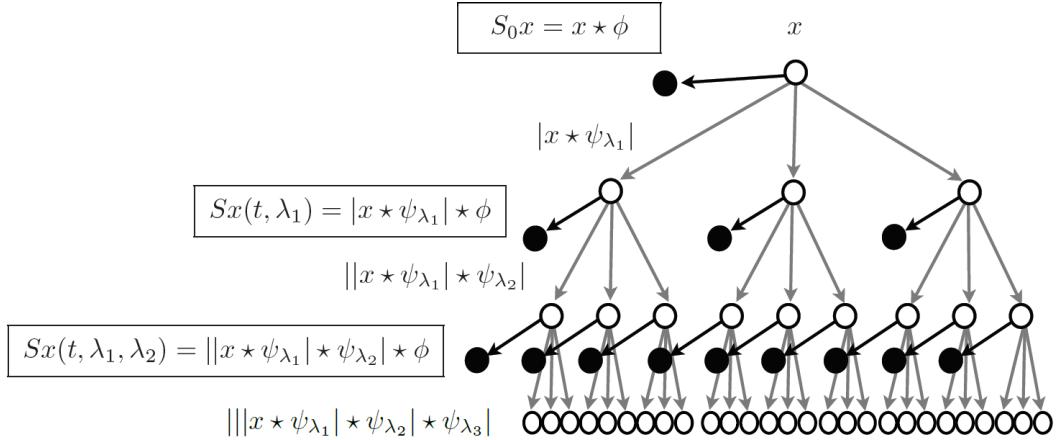


FIGURE 4.3 – Illustration des coefficients obtenus par une transformée diffusante d'ordre 2. Les points noirs représentent les coefficients extraits, c'est à dire les coefficients issus d'un filtrage passe-bas des modules de transformées en ondelettes [16].

On remarque que l'on utilise pour le calcul de la transformée diffusante le module de la transformée en ondelettes. Stéphane Mallat justifie ce choix de plusieurs manières.

L'idée est de chercher une représentation invariante par translation d'un signal x . Or si x translate, $x * \psi_{\lambda_1}$ est modifié. Cependant, $\int x * \psi_{\lambda_1}(t) dt$ reste inchangée. Malheureusement cette grandeur est nulle à cause de la répartition d'amplitude de l'ondelette (moyenne nulle), et on peut prouver que $\int M(x * \psi_{\lambda_1})(t) dt$ où M est un opérateur linéaire vaudra toujours 0.

Il faut donc trouver un opérateur non linéaire, « stable » (Mallat évoque différents critères de stabilité que nous aborderons succinctement dans la partie 4.3.3) pour les petites déformations de x et l'ajout d'un bruit. La seule solution est alors de choisir le module comme opérateur non linéaire.

L'action intégrale est quant à elle réalisée par le filtre passe-bas ϕ_{2^J} qui permet d'obtenir une certaine invariance aux translations inférieures à 2^J , fenêtre spatiale de ϕ_{2^J} . On utilise le filtre passe-bas uniquement pour calculer les coefficients en sortie de la transformée diffusante, et surtout pas pour les calculs intermédiaires car le moyennage fait perdre les hautes fréquences, il ne serait donc plus possible de discriminer les différents signaux.

Les coefficients du second ordre permettent de récupérer des informations discriminantes qui ont été perdues par le calcul de la moyenne des coefficients du premier ordre. La figure 4.4 montre deux images de textures qui présentent des coefficients similaires à l'ordre 1.

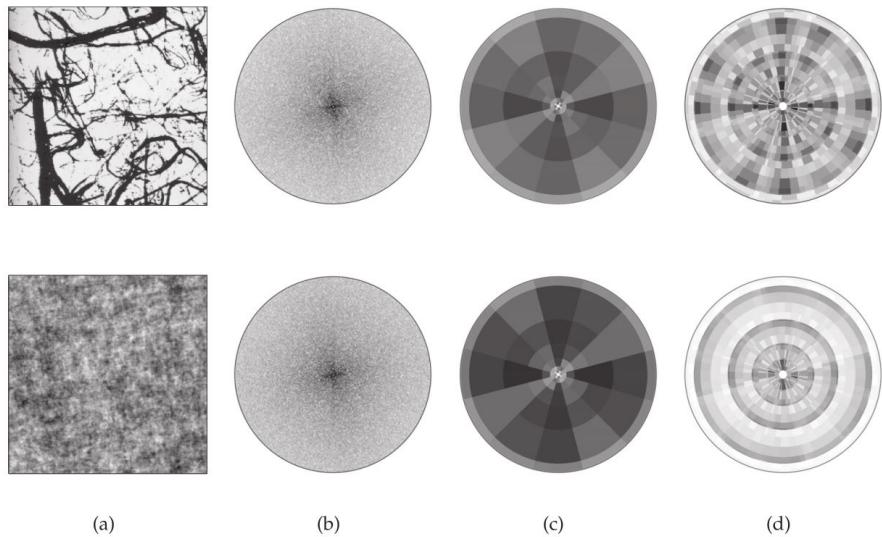


FIGURE 4.4 – (a) Deux réalisations de processus stationnaires. (b) Densité spectrale de puissance. (c) Coefficients de la transformée diffusante à l’ordre 1. (d) Coefficients de la transformée diffusante à l’ordre 2. [17].

Pour la transformée diffusante d’ordre 1, les coefficients dépendent principalement des moments de second ordre et sont donc presque égaux pour les deux textures. Contrairement au spectre de puissance de Fourier, la transformée diffusante d’ordre 2 donne donc des informations sur les moments d’ordre supérieur (jusqu’à l’ordre 4), et peut ainsi distinguer les textures non gaussiennes ayant la même densité spectrale de puissance. On notera cependant que la transformée diffusante à l’ordre 2 produit un signal plus grand que le signal d’origine.

Pour des ondelettes appropriées, [17] montre que les coefficients du premier ordre de la transformée diffusante sont équivalents aux coefficients SIFT [18]. Par conséquent, la transformée diffusante ne fournit pas seulement des caractéristiques d’invariances par translation, mais elle récupère également l’information à haute fréquence perdue dans SIFT, ce qui permet d’obtenir des descripteurs plus riches dans les cas complexes.

4.3.2 Transformée Diffusante d’Ordre « m »

On peut en théorie définir la transformée diffusante à n’importe quel ordre $m \in \mathbb{N}$ en introduisant les résolutions fréquentielles, coefficients de dilatations et jeux d’ondelettes associés. On verra qu’en pratique, on se contente d’une transformée à l’ordre 2 voire à l’ordre 1 qui capturent déjà une très grande partie de l’énergie et donc de l’information contenue dans un signal [17].

4.3.3 Propriétés Théoriques

[17] démontre plusieurs propriétés :

- Préservation de l’énergie : $\|S_Jx\|^2 = \|x\|^2$ car l’opérateur module préserve l’énergie $\|x\|^2 = \| |x| \|^2$;

- Stable au bruit additif : $\|S_Jx - S_Jy\| \leq \|x - y\|$, car tous les opérateurs sont non-expansifs ;
- Invariance par translation locale : $\forall |a| \ll 2^J, S_J(\mathcal{L}_a.x) \approx S_Jx$ où $\mathcal{L}_a.x(u) \triangleq x(u - a)$;
- Stable au petites déformations : $\exists C > 0, \|S_J(\mathcal{L}_\tau x) - S_Jx\| \leq C \|\nabla \tau\|_\infty \|x\|$ où $\mathcal{L}_\tau.x(u) \triangleq x(u - \tau(u))$

4.4 Transformée Diffusante 2D

La transformée diffusante 2D est définie sur la base des mêmes concepts introduits dans les parties précédentes. On utilise pour définir la transformée en diffusante d'un signal 2D des ondelettes complexes en 2D qui présentent toujours les mêmes caractéristiques (moyenne nulle et support fini) que les ondelettes 1D.

La différence est l'ajout d'un paramètre de rotation : une ondelette de base peut maintenant être dilatée $j \geq 0$ mais également être sujette à une rotation r d'angle $\theta = 2k\pi/K$ où $0 \leq k < K$.

$$\psi_{j,\theta}(u) = 2^{-2j}\psi(r_{-\theta}2^{-j}u)$$

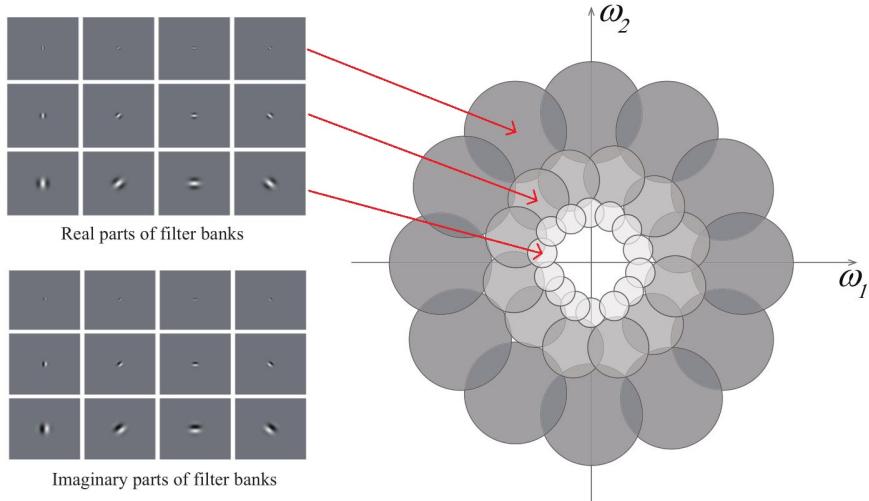


FIGURE 4.5 – Illustration des parties réelles et imaginaires des ondelettes 2D dans le domaine spatial et correspondance dans le domaine fréquentiel. On remarque que toute la surface du plan fréquentiel est occupée par des ondelettes (ϕ n'est pas représentée) [19].

Supposons que le tenseur en entrée ait la taille suivante : (C, N_1, N_2) où C est le nombre de « channels » et (N_1, N_2) la taille de l'image. Alors, le tenseur de sortie de la « scattering transform » en deux dimensions aura comme taille :

$$\left(C, 1 + LJ + \frac{L^2 J(J-1)}{2}, \frac{N_1}{2^J}, \frac{N_2}{2^J} \right)$$

où J est le facteur d'échelle et L le nombre de rotations des ondelettes.

La première dimension du tenseur de sortie est de taille C , car on réalise la transformée diffusante sur chaque canal de l'image, indépendamment des autres.

Sur la deuxième dimension du tenseur, le premier coefficient correspond au calcul de $S_J[\emptyset]x = x \star \phi$ où x est le signal d'entrée et ϕ le filtre passe bas. À l'ordre 1 on obtient $J \times L$ coefficients, ce qui correspond bien au nombre d'ondelettes présentes dans la banque de filtres.

À l'ordre suivant on n'obtient pas tout à fait $LJ \times LJ$ coefficients car on ne calcule pas toutes les branches de l'arbre 4.3. En effet, comme l'opérateur de module prend l'enveloppe du signal et enlève les oscillations, il pousse les signaux intermédiaires vers les basses fréquences ce qui nous amène donc à ne considérer que les chemins vers les fréquences décroissantes. Par conséquent, on ne calcule que $\frac{L^2 J(J-1)}{2}$ coefficients à l'ordre 2.

Les troisième et quatrième dimensions du tenseur, $\left(\frac{N_1}{2^J}, \frac{N_2}{2^J}\right)$, sont dues à une opération de sous-échantillonnage propre à l'implémentation des réseaux diffusants [17] [20]. En effet, afin de réduire la complexité des calculs des prochaines opérations, les signaux doivent être sous-échantillonnés par un facteur de $\alpha 2^j$ pour $\psi_{j,\theta}$ et $\alpha 2^J$ pour ϕ_{2^J} , avec $\alpha = 1$ ou $\alpha = 0.5$ pour éviter les effets d'aliasing. Cette opération est effectuée par une périodisation dans le domaine de Fourier, ce qui est équivalent à un sous-échantillonnage dans le domaine spatial.

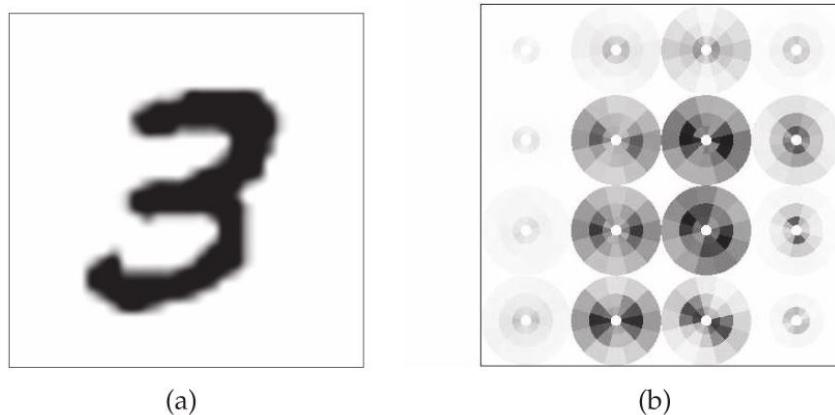


FIGURE 4.6 – Image d'entrée de taille 32×32 . Transformée diffusante d'ordre 1 avec $J = 3$ et $L = 12$. On obtient donc un tenseur 3D de largeur et hauteur 4×4 et de profondeur JL [17]. Chaque cercle de la figure (b) possède J anneaux et L tranches.

Illustrons le résultat de la transformée diffusante sur un exemple. Considérons dans toute la suite une image d'un mandrill de 512×512 pixels contenue dans une matrice x .

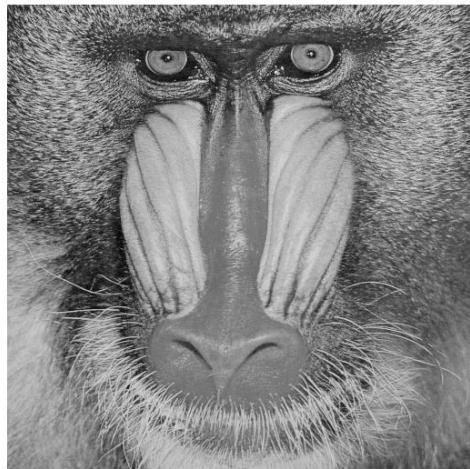


FIGURE 4.7 – Image d'entrée (mandrill).

Prenons les filtres définis ci-dessous pour calculer la transformée et affichons les images intermédiaires obtenues pour chaque couche de la transformée diffusante.

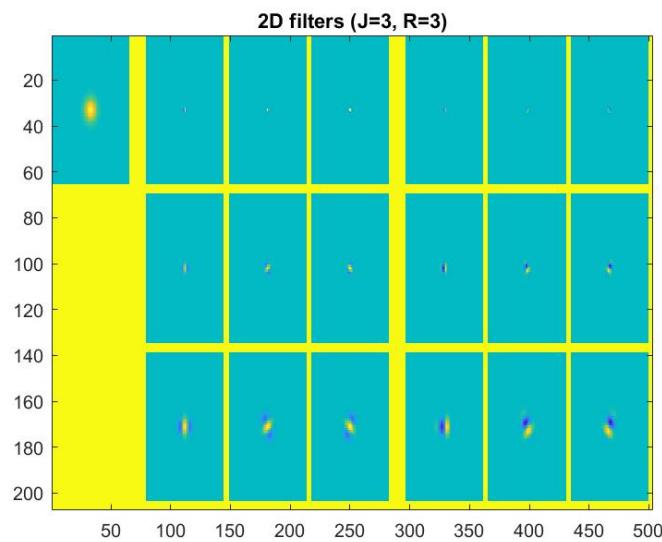


FIGURE 4.8 – Ondelettes de Morlet 2D avec un facteur de dilatation $J = 3$ et $L = R = 3$ angles de rotation (de gauche à droite, filtre passe-bas, parties réelles des ondelettes, parties imaginaires des ondelettes).

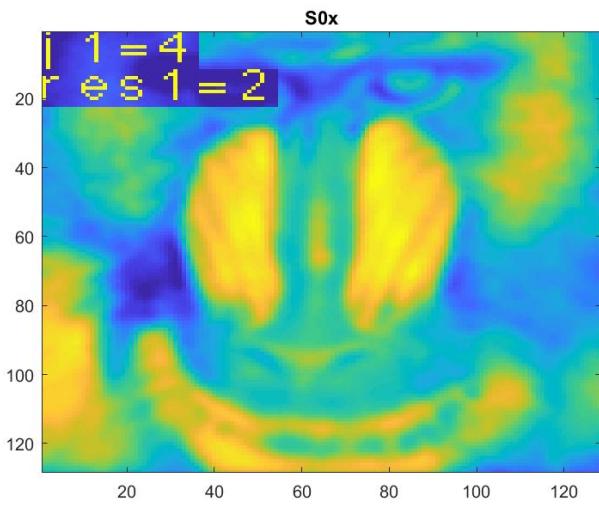


FIGURE 4.9 – Première couche (ordre 0) de la transformée diffusante.

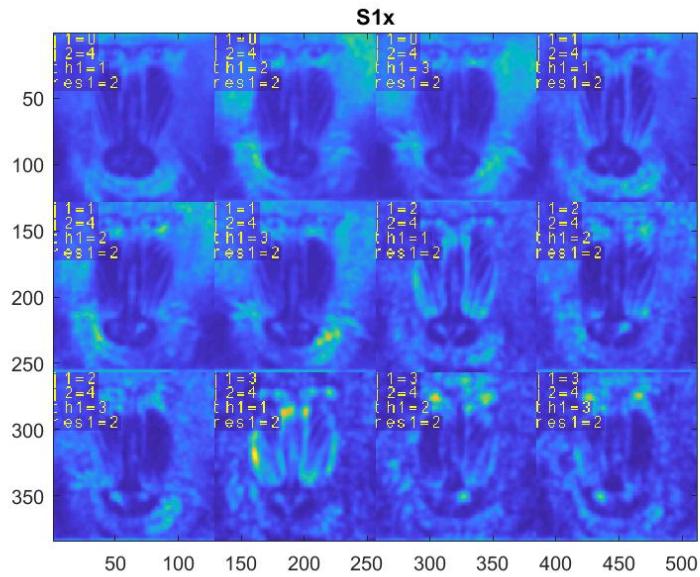


FIGURE 4.10 – Deuxième couche (ordre 1) de la transformée diffusante.

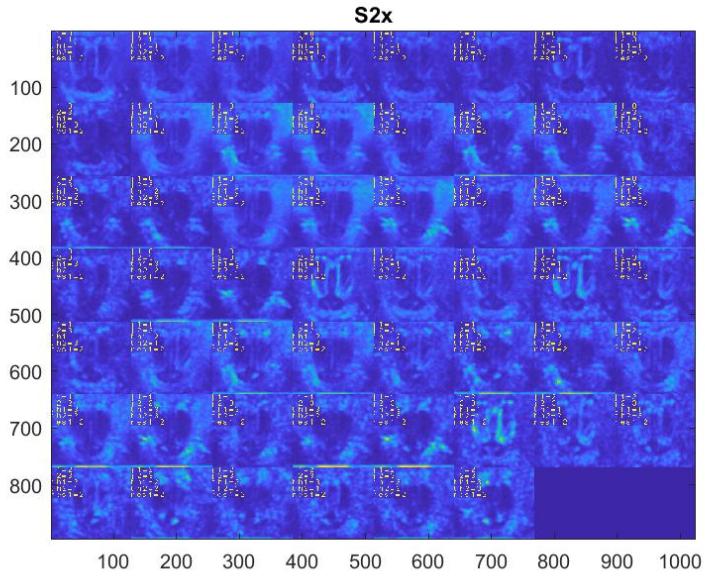


FIGURE 4.11 – Troisième couche (ordre 2) de la transformée diffusante.

4.5 Applications des Réseaux Diffusants

Outil d'extraction de caractéristiques La transformée diffusante peut simplement être envisagée comme un outil d'extraction d'attributs visant à obtenir une représentation parcimonieuse des signaux à classer. Ainsi, on peut utiliser les coefficients obtenus par la transformée diffusante comme des « features » pour entraîner n'importe quel algorithme de machine-learning classique comme les machines à vecteurs de support par exemple. Il est aussi tout à fait possible d'utiliser un algorithme de clustering non-supervisé comme K-Means par exemple.

Réseau d'apprentissage profond hybride Les réseaux diffusants sont en fait des couches de convolutions où l'on aurait remplacé les filtres entraînables par des filtres passe-bande pré-définis comme des ondelettes 2D. Cela revient donc à avoir une architecture avec la transformée diffusante en entrée du réseau puis d'autres couches de convolution classiques suivie de couches « totalement connectées » avant la couche de sortie. Ces réseaux hybrides seront étudiés plus amplement dans la suite de ce rapport.

Signaux 1D Pour des signaux 1D, les réseaux diffusants se sont déjà illustrés de manière efficace pour la classification de genres musicaux par exemple [21].

Signaux 2D Pour ce qui est des images, les réseaux diffusants sont connus pour leur efficacité lorsqu'il s'agit de reconnaître des textures [17]. Prenons par exemple une image d'un mur de briques et la banque de 20 filtres passe-bandes suivante.

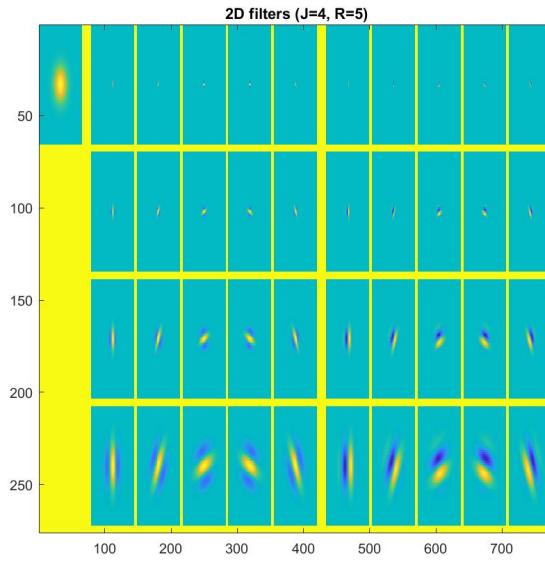


FIGURE 4.12 – Banque de filtres.

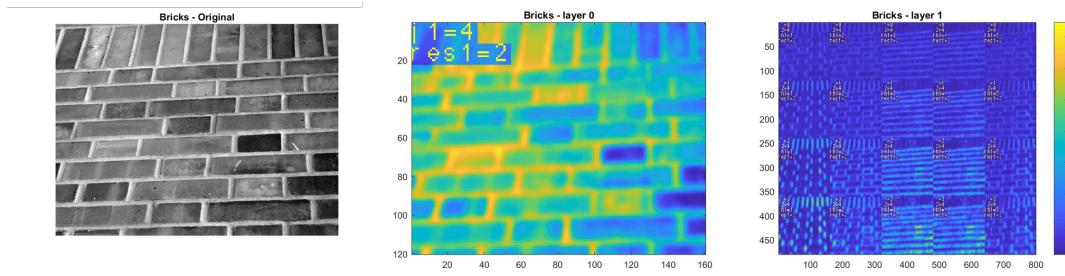


FIGURE 4.13 – Transformée en ondelettes avec de gauche à droite : l'image originale, l'image moyennée et le résultat de la transformée diffusante d'ordre 1.

Il y a correspondance entre les réponses impulsionnelles des filtres (fig 4.12) et les différents coefficients obtenus à l'ordre 1 (fig 4.13). Ceci permet de remarquer que l'on arrive effectivement à identifier la texture « mur de briques » avec les bons coefficients de dilatation (une partie réelle supérieure à 0 de l'ondelette sur un intervalle similaire à la largeur des sillons inter-briques) et la bonne rotation (alignement de la crête de la partie réelle de l'ondelette avec les sillons).

Signaux 3D Il existe également des applications des réseaux diffusants en 3D, par exemple pour la classification de maladies neurodégénératives à partir d'examens IRM [22].

Chapitre 5

Démarche et Travail Réalisé

Cette partie présente le cœur de notre étude sur les réseaux diffusants. Dans un premier temps nous présenterons les outils utilisés pour implémenter et étudier les réseaux diffusants. Dans un second temps nous aborderons les expérimentations que nous avons effectuées de manière à analyser les propriétés des réseaux diffusants ainsi que leurs avantages et inconvénients par rapport aux réseaux de neurones convolutifs.

Les diverses expérimentations ont été réalisées sur les clusters de Supélec, composés de GPU NVIDIA 1080, 1080ti et 2080ti.

5.1 Implémentation des Réseaux Diffusants

La table 5.1 fournit une comparaison détaillée des implémentations de réseaux diffusants existantes.

	Dimension	GPU	Différentiable	Core Devs.	Langage
ScatNet	1D, 2D			5	MATLAB
ScatNetLight	2D			2	MATLAB
PyScatWave	2D	✓		3	Python
Scattering.m	1D			1	MATLAB
PyScatHarm	3D	✓		1	Python
Wavelet Toolbox	1D			N/A	MATLAB
Kymatio	1D, 2D, 3D	✓	✓	15	Python

TABLE 5.1 – Comparaison des librairies de réseaux diffusants. [23]

Notre choix s'est naturellement porté sur Kymatio [23]. Bien que récente, elle rassemble la majorité des fonctionnalités présentes dans les autres librairies citées ci-dessus. De plus, elle est utilisable sur GPU ce qui accélère drastiquement les calculs. L'intégration des calculs de coefficients dans un réseau CNN se fait de manière très simple car la librairie est basée sur PyTorch [24]. Comme Kymatio ne propose pour l'instant que les ondelettes de Morlet, tous les résultats donnés plus loin dans ce rapport sont basés sur cette ondelette. Kymatio évoluant rapidement, les résultats de notre étude ne se-

ront peut-être pas parfaitement reproductibles si quelqu'un souhaite utiliser une version autre que la *0.1.0b0*.

La librairie Kymatio propose actuellement deux backends pour les calculs :

- torch : implémentation basée sur PyTorch et qui est donc différentiable au cas où on devrait propager un gradient. Cependant, elle s'appuie sur des noyaux CUDA polyvalents pour le calcul GPU, ce qui réduit les performances.
- skcuda : implémentation utilisant des noyaux CUDA optimisés (à travers cupy) et scikit-cuda. Cette implémentation ne fonctionne que sur GPU. Puisqu'elle utilise des noyaux optimisés pour les différentes étapes de la transformée diffusante, elle atteint de meilleures performances par rapport au backend torch par défaut. Une implémentation différentiable est en cours d'étude.¹

Les implémentations précédentes de la transformée diffusante, telles que ScatNet [25], reposaient sur le calcul des coefficients couche par couche. Dans Kymatio, l'arbre est parcouru en profondeur. Cela limite l'utilisation de la mémoire et rend l'implémentation plus adaptée à l'exécution sur un GPU. La différence entre les deux approches est illustrée dans la figure 5.1.

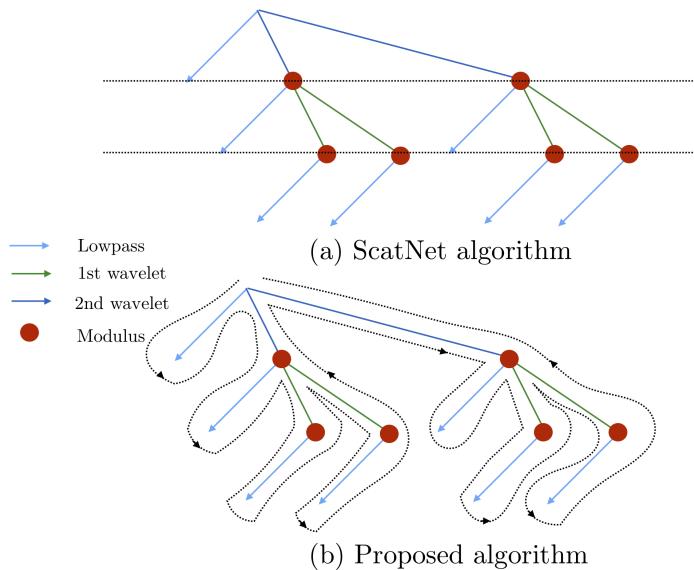


FIGURE 5.1 – Stratégie de parcours de l'arbre. [Source](#).

Tous les calculs étant réalisés dans le domaine spectral avec des FFT, les modules d'ondelettes du premier et du second ordre sont donc calculés en $O(N \log(N))$. L'opération de filtrage passe-bas est également calculée avec $O(N \log(N))$ opérations, toujours avec une FFT.

Nous avons aussi utilisé la librairie ScatNet [25] en langage Matlab pour certaines expériences car Kymatio ne dispose pas encore de toutes les fonctionnalités de ScatNet.

1. <https://github.com/kymatio/kymatio/pull/327>

5.2 Expérimentations

Dans cette partie nous abordons les diverses expérimentations conduites durant notre étude de manière à analyser les réseaux diffusants (partie 5.2.1) ainsi que les différences par rapport aux réseaux convolutifs (partie 5.2.2).

5.2.1 Propriétés des Réseaux Diffusants

5.2.1.1 Perte d'information de la transformée diffusante

On cherche ici à quantifier la perte d'information de la transformée diffusante aux ordres 1 et 2 et pour différents facteurs d'échelle J .

Pour cela, on calcule la transformée diffusante Sx à partir d'une image x et on cherche à reconstruire l'image originale x à partir des coefficients calculés précédemment. Le problème d'optimisation est donc le suivant :

$$\hat{x} = \inf_y \|Sx - Sy\|$$

Comme l'implémentation des réseaux diffusants est différentiable, on utilise une descente de gradient pour obtenir une estimation \hat{x} de l'image originale et on quantifie la perte d'information avec le PSNR.

	Ordre	$J = 1$	$J = 2$	$J = 3$	$J = 4$	$J = 5$
Image 5.2b	1	41.8	35.0	30.2	26.5	24.0
	2	42.2	38.0	34.6	29.8	26.5
Image 5.2a	1	27.0	22.7	19.3	17.3	15.4
	2	27.1	23.3	21.0	18.4	16.3

TABLE 5.2 – PSNR (dB) pour différents J aux ordres 1 et 2 ($L = 8$ pour toutes les mesures).



(a) Image de taille 512×384 . Milieu : Ordre 1 et $J = 3$. Droite : Ordre 1 et $J = 5$



(b) Image de taille 384×384 . Milieu : Ordre 1 et $J = 3$. Droite : Ordre 1 et $J = 5$

FIGURE 5.2 – Images originales et reconstructions.

On constate tout d’abord que la transformée diffusante à l’ordre 1 implique légèrement plus de perte d’informations que l’ordre 2 ce qui était prévisible. Cependant, l’écart est relativement faible et la reconstruction est tout de même satisfaisante, la structure de l’image est conservée et on arrive à reconnaître les objets présents dans l’image sans problème. Il semblerait que la majorité de la perte d’information soit réalisée sur les bordures des objets, donc vers les hautes fréquences.

Les expérimentations montrent que pour des images de quelques centaines de pixels de côté, J doit être choisi strictement inférieur à 4 pour assurer une bonne reconstruction. On rappelle que le facteur d’échelle J divise la résolution spatiale par 2^J , plus J est grand et plus on perd d’information spatiale.

Dans un second temps, on cherche à entraîner un réseau de classification hybride « Scattering + CNN » de manière à déterminer la différence de score entre l’ordre 1 et l’ordre 2. Pour cela, on choisit comme jeu de données le STL-10 [26] composé d’images de la vie courante de taille 96×96 , avec 5000 images d’entraînement réparties en 10 classes.

Le réseau hybride est basé sur une architecture ResNet34 où l’on remplace *conv1*, *conv2_x* et *conv3_x* [10] par la « scattering transform » avec $J = 2$, car la sortie de la transformée diffusante a une résolution beaucoup plus faible que la résolution de l’image d’entrée prévue pour les réseaux CNNs classiques. Le réseau obtenu possède environ 20 millions de paramètres et on l’entraîne avec une taille de batch de 64, un learning rate de 0.01 avec l’optimiseur Adam pendant 250 epochs. Les mesures ont été effectuées 6 fois pour chaque ordre.

On obtient finalement un score « accuracy » de $69.21 \pm 1\%$ pour l’ordre 1 et $69.75 \pm$

1.1% pour l'ordre 2.

Les deux expériences précédentes permettent donc de conclure que pour la classification d'image ou des applications similaires, l'ordre 1 est amplement suffisant et permet de garder une bonne partie de l'information contenue dans le signal d'entrée. Les conclusions de la publication [27] semblent concorder avec nos résultats.

5.2.1.2 Impact des paramètres de la transformée diffusante

Dans cette section, on s'intéresse à l'influence que peuvent avoir les paramètres internes de la transformée diffusante avec ondelettes de Morlet sur les performances d'un algorithme de classification. Les paramètres testés sont J (nombre de coefficients de dilatation appliqués à l'ondelette de base), L (nombre de rotations appliquées à l'ondelette de base) et M (l'ordre de la transformée diffusante).

La « scattering transform » est implémentée avec la bibliothèque ScatNet et la classification des coefficients est réalisée avec une SVM à l'aide de la bibliothèque scikit-learn [28].

Le data set utilisé dans cette section est la banque de textures de l'université de l'Illinois (UIUC) [29]. Cette dernière est constituée de 1000 images de textures de taille 480×640 (en niveaux de gris) réparties en 40 classes équilibrées (25 images/classe). Voici un échantillon d'une douzaine d'images prises aléatoirement dans la banque d'images.

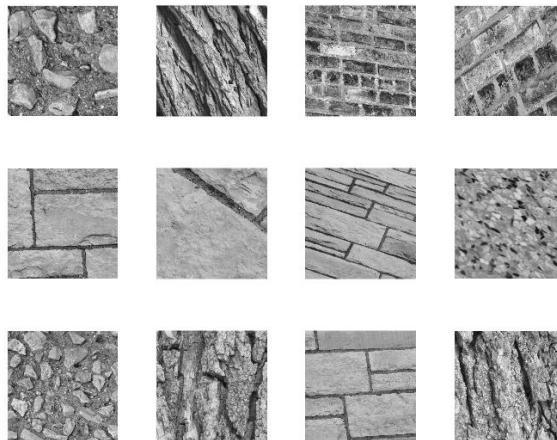


FIGURE 5.3 – Échantillon d'images du dataset de l'UIUC.

Les features associés à une image considérées pour entraîner la SVM sont définies de la manière suivante :

- une transformée diffusante de l'image de paramètres J , L et M donnés est calculée et donne comme expliqué précédemment $1 + LJ + \frac{L^2 J(J-1)}{2}$ imagettes
- on calcule pour chaque imagette la somme de ses coefficients.

On obtient donc finalement pour chaque image du data set $1 + LJ + \frac{L^2 J(J-1)}{2}$ coefficients qui serviront à entraîner la SVM. Cette façon de définir les features peut paraître assez étrange dans la mesure où on a le sentiment de perdre beaucoup d'information en procédant de la sorte mais elle donne de bons résultats comme illustré dans la suite.

Influence de M On commence par regarder l'influence de l'ordre de la transformée diffusante. Comme déjà évoqué, la transformée diffusante d'ordre 2 est beaucoup plus gourmande en temps de calcul que la transformée d'ordre 1. Ainsi, avec un processeur Intel Core I5 et deux coeurs mobilisés grâce à la Parallel Computing toolbox de Matlab, ScatNet prend 27 minutes pour calculer la transformée diffusante des 1000 images de la banque avec $J = 5$ et $L = 5$ à l'ordre 2 pour seulement 9 minutes à l'ordre 1.

On utilise la fonction Grid-Search de scikit-learn pour trouver le meilleur paramétrage de la SVM qui servira pour les prochaines comparaisons. Le Grid-Search porte sur les paramètres γ (la SVM est à kernel gaussien) et C qui peuvent respectivement prendre leur valeur dans $[0.1, 0.01, 0.001, 0.0001]$ et $[1, 10, 100, 1000]$. La précision des classificateurs obtenus est calculée avec une cross-validation d'ordre 5 pour limiter le risque d'overfitting. On obtient les résultats suivant :

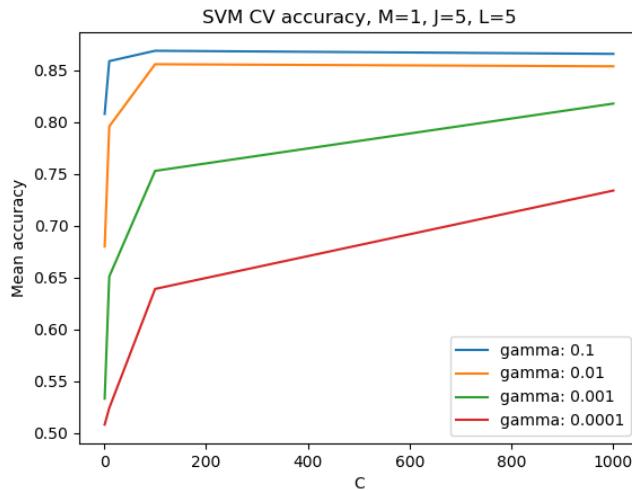


FIGURE 5.4 – Précisions obtenues à l'ordre 1.

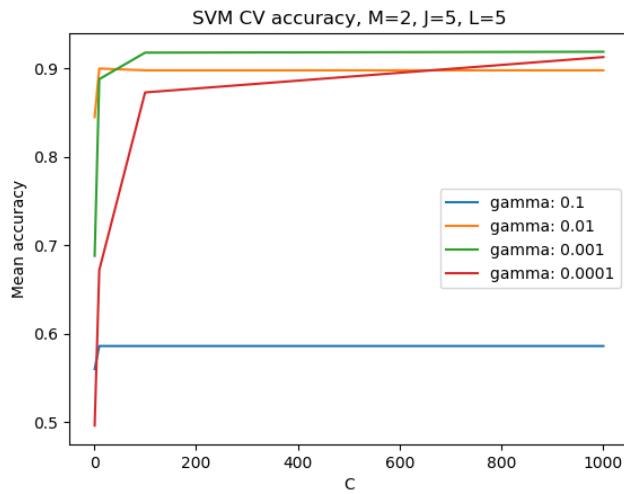


FIGURE 5.5 – Précisions obtenues à l’ordre 2.

On remarque que la transformée diffusante à l’ordre 2 fournit des résultats légèrement meilleurs (91% contre 86% pour l’ordre 1) ce qui paraît logique. Le temps d’entraînement des SVM est de l’ordre du dixième de seconde pour l’ordre 1 et de la seconde pour l’ordre 2. Dans la suite, on réalisera nos tests avec $M = 1$ pour réduire le temps de calcul des transformées diffusantes.

Influence de J Pour cette expérience, on fixe $M = 1$ et $L = 5$ et on fait varier J de 1 à 7. À noter que pour ScatNet, le paramètre J correspond au nombre de coefficients de dilatation en incluant le coefficient unité. Ainsi, $J = 3$ signifie que la banque d’ondelettes sera constituée de l’ondelette de base, de l’ondelette dilatée d’un facteur 2 et de l’ondelette dilatée d’un facteur 4.

Les hyperparamètres de la SVM sont fixés à $C = 100$ et $\gamma = 0.1$ qui fournissaient de bons résultats lors des tests sur l’ordre de la transformée diffusante. Chaque test de valeur de J fait encore une fois l’objet d’une cross-validation d’ordre 5 pour limiter l’overfitting. Voici les résultats obtenus :

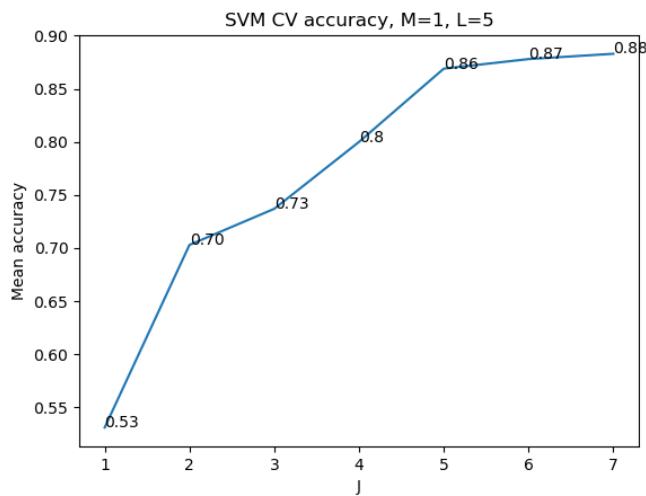


FIGURE 5.6 – Précisions obtenues en faisant varier le nombre de coefficients de dilatation de la transformée diffusante.

Pour rappel, plus on augmente le nombre de coefficients de dilatation, plus on réalise en fait une étude fine en fréquence. On comprend donc bien la différence importante entre la précision calculée pour une transformée diffusante $J = 1$ qui ne s'intéresse finalement qu'à une seule fréquence et pour une transformée diffusante $J = 7$ qui s'intéresse à 7 fréquences.

On peut prendre l'exemple des classes 24 et 25 du data set dont voici l'apparence :

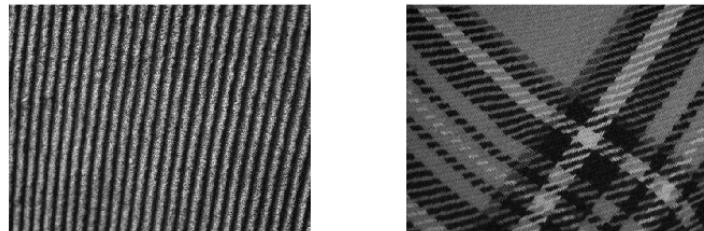


FIGURE 5.7 – Exemples pour les classes 24 et 25.

Avec $J = 1$, ces classes sont confondues par la SVM, en témoigne la matrice de confusion suivante :

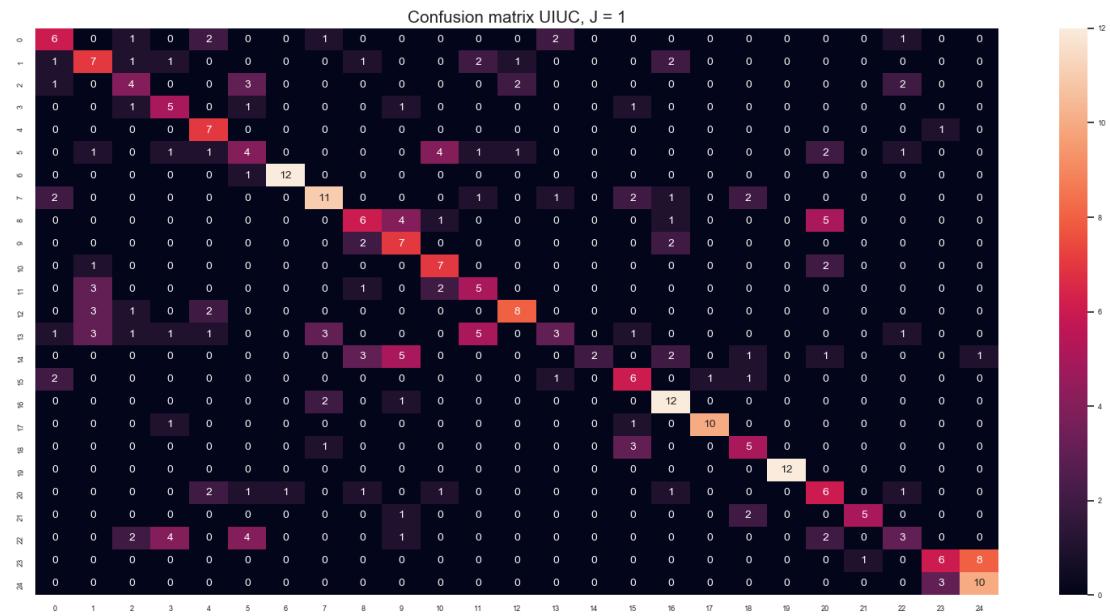


FIGURE 5.8 – Matrice de confusion avec une transformée diffusante à un coefficient de dilatation.

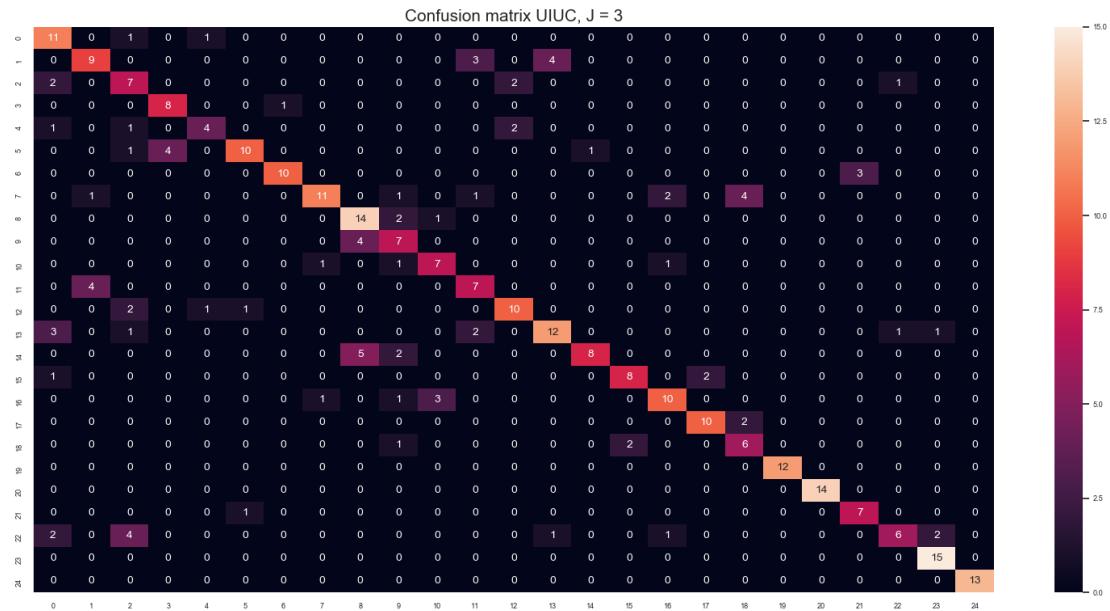


FIGURE 5.9 – Matrice de confusion avec une transformée diffusante à trois coefficients de dilatation.

On comprend bien que ne considérer qu'une seule ondelette de coefficient de dilatation donné va peut-être permettre de capturer la fréquence correspondant au tissage des deux tissus ci-dessus, mais on ne pourra dans ce cas pas capturer la complexité du motif de la classe 25 qui est constitué de plusieurs autres fréquences.

Il est ainsi difficile de faire la différence entre ces deux classes pour $J = 1$ comme

l'illustre le coin inférieur droit de la figure 5.8. Avec $J = 3$ (figure 5.9), on arrive à bien différencier ces deux classes.

Influence de L Pour tester l'influence du nombre de rotations appliquées à l'ondelette de base pour construire la banque de filtres, on procède de la même manière en fixant $M = 1$ et $J = 6$. On utilise toujours une SVM paramétrée de la même façon et une cross-validation d'ordre 5 pour l'évaluation de la précision. On fait varier L de 1 à 5 sachant que pour ScatNet l'angle θ correspondant à une valeur de $k \in [1, L]$ vaut $\pi \frac{k}{L}$. On obtient les résultats suivants :

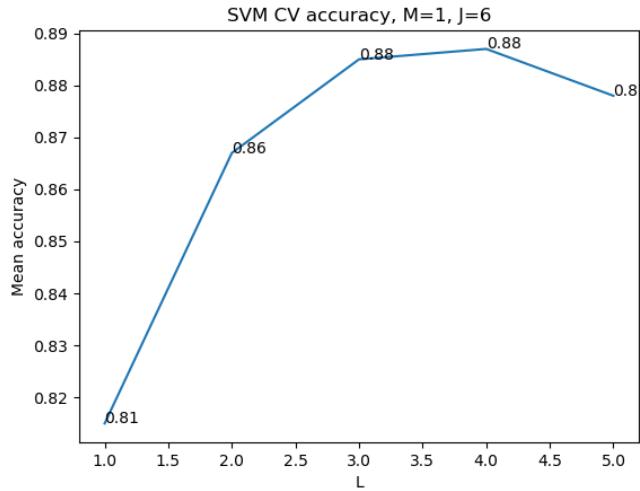


FIGURE 5.10 – Précisions obtenues en faisant varier le nombre de rotations de la transformée diffusante.

On remarque cette fois-ci que le paramètre L a beaucoup moins d'impact sur les résultats en terme de précision de la SVM. On peut expliquer cela par le fait que ce qui semble discriminer chaque classe de texture est plus souvent lié aux fréquences des motifs que l'on trouve dans chaque image qu'à leur orientation.

5.2.1.3 Impact de la taille de la base d'entraînement

Dans cette section, on s'intéresse à l'influence du nombre d'échantillons dans le set d'entraînement sur la précision de la classification opérée par la SVM. On considère dans le cadre de cette expérience les paramètres de la transformée diffusante et de la SVM utilisés précédemment, à savoir une transformée diffusante avec $J = 6$, $L = 5$ et $M = 1$ et une SVM avec $C = 100$ et $\gamma = 0.1$.

On exclut pour commencer 200 images du data set qui serviront d'ensemble de validation. Ensuite, on réalise un apprentissage sur la SVM avec un nombre croissant d'échantillons, chaque SVM étant testée sur le set de validation. On obtient la courbe en précision suivante :

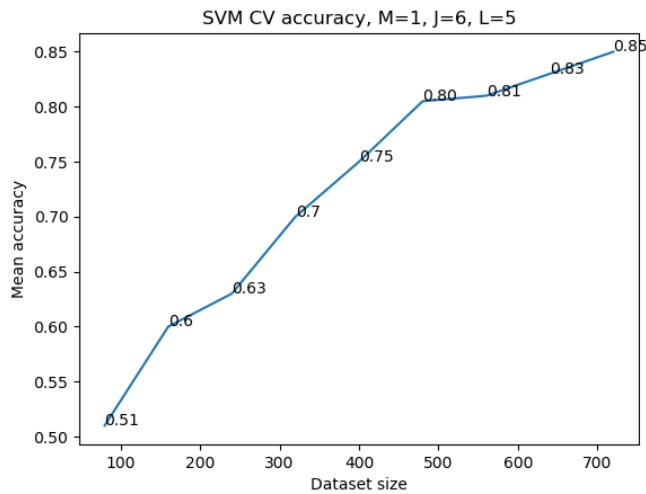


FIGURE 5.11 – Précisions obtenues en faisant varier le nombre d’images utilisées pour entraîner la SVM.

On remarque qu’avec seulement 100 images pour entraîner la SVM, on parvient tout de même à un score de précision 50%, soit 46 points de plus qu’un classifieur naïf qui affecterait au hasard une des 25 classes à une image. Cet exemple témoigne bien de l’efficacité de la transformée diffusante comme outil de représentation parcimonieuse de nos images. En effet, en extrayant 31 coefficients ($1 + LJ$) des 480×640 coefficients qui décrivaient initialement chacune des images, on arrive à inférer à partir de seulement 4 exemples par classe une représentation juste une fois sur deux de chacune des classes.

5.2.1.4 Essai d’une représentation alternative des images

Toutes les expériences menées précédemment avec une SVM utilisaient les features décrites en section 5.2.1.2. Cette définition, bien qu’assez contre-intuitive, a donné de bons résultats de classification mais Bruna et Mallat [17] introduisent une autre représentation de la transformée diffusante que l’on a choisi d’utiliser pour définir de nouvelles features.

On a déjà évoqué le fait que la transformée diffusante d’une image de taille (N_1, N_2) avec J coefficients de dilatation donnait plusieurs imagettes de taille $\left(\frac{N_1}{2^J}, \frac{N_2}{2^J}\right)$. L’idée est ici de choisir $2^J = N_1 = N_2$ tel que chaque imagette ne soit en fait constituée que d’un seul coefficient. Or, les images du data set de textures de l’UIUC n’étant pas carrées, elles nécessitent un pré-traitement et sont toutes rognées à $(2^8, 2^8)$.

Une fois rognée, on peut calculer la transformée diffusante d’une image et récupérer les coefficients pour les arranger sous une forme similaire à celle de la représentation de Bruna et Mallat.

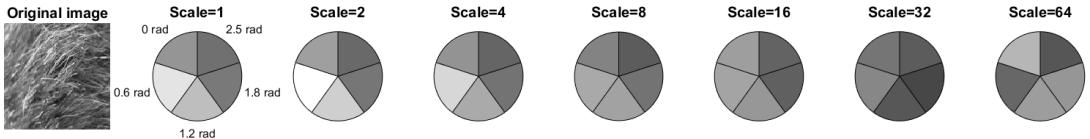


FIGURE 5.12 – Exemple de représentation d’une texture : chaque « tarte » correspond à un coefficient de dilatation et chaque « part de tarte » à une rotation.

On calcule les transformées diffusantes de chaque image avec $J = 8$, $L = 5$ et $M = 1$ pour les exporter vers une SVM. On réalise une étude par Grid-Search identique à celle menée pour étudier l’influence de l’ordre de la transformée diffusante et on obtient les résultats suivants :

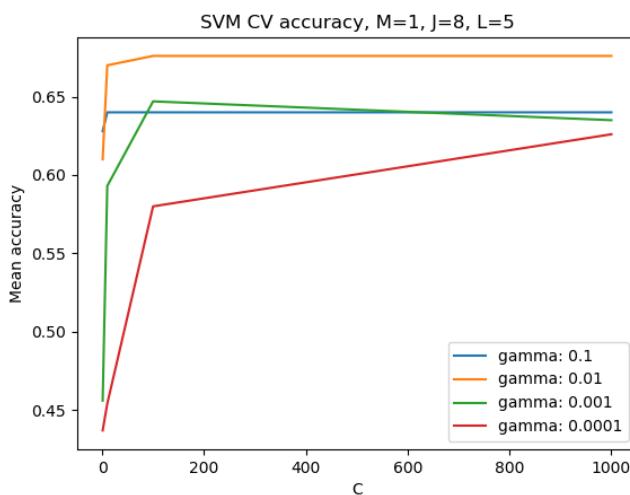


FIGURE 5.13 – Précisions obtenues à l’ordre 1.

On remarque que les résultats obtenus avec cette méthode sont bien moins bons (25%) en terme de précision que ceux obtenus avec la première définition des features.

En effet, il est démontré [30] que lorsque 2^J augmente, la quantité $\|S_Jx - S_Jy\|$ diminue ce qui ne permet plus de discriminer x et y .

5.2.2 Comparaison des Réseaux Diffusants et des Réseaux Convolutifs

5.2.2.1 Temps de calcul et occupation mémoire GPU

On cherche ici à comparer les réseaux diffusants et réseaux convolutifs en termes de temps de calcul et d’occupation mémoire.

On réalise donc des mesures de performances avec différentes tailles de batch et pour des images de taille 256×256 . Nous avons aussi comparé les deux backends de Kymatio, car le backend skcuda est censé être plus optimisé que le backend torch.

Pour comparer les réseaux convolutifs et les réseaux diffusants de manière équitable, les architectures des CNNs « équivalents » sont définies de manière à obtenir environ le même tenseur de sortie que la transformée diffusante correspondante. Nous avons aussi effectué les mesures pour les réseaux convolutifs en mode d’entraînement (avec gradient) et en mode inférence (calcul de la sortie uniquement). Les « \times » indiquent une saturation de la mémoire GPU. Les mesures ont toutes été faites sur un GPU NIVDIA 2080ti.

	Taille de batch			
	32	64	128	512
Scattering (backend torch)	0.16s	0.33s	0.65s	2.60s
Scattering (backend skcuda)	0.04s	0.09s	0.17s	0.67s
CNN (en inférence)	0.01s	0.02s	0.05s	0.17s
CNN (avec gradient)	0.06s	0.09s	0.18s	\times

TABLE 5.3 – Temps de calcul à l’ordre 1.

À l’ordre 1, on constate que les réseaux convolutifs (avec rétro-propagation du gradient) sont environ aussi rapides que les réseaux diffusants (avec le backend skcuda). Concernant l’occupation mémoire, les réseaux convolutifs saturent pour des grosses tailles de batch.

	Taille de batch			
	32	64	128	512
Scattering (backend torch)	0.72s	1.44s	2.86s	11.45s
Scattering (backend skcuda)	0.17s	0.35s	0.69s	2.76s
CNN (en inférence)	0.23s	0.43s	0.81s	\times
CNN (avec gradient)	0.68s	\times	\times	\times

TABLE 5.4 – Temps de calcul à l’ordre 2.

À l’ordre 2, les réseaux diffusants sont clairement compétitifs par rapport aux réseaux convolutifs pour ce qui est du temps de calcul. Même constatation qu’à l’ordre 1 au niveau de la mémoire.

On retiendra donc que les réseaux diffusants obtiennent des temps de calculs du même ordre de grandeur que les réseaux convolutifs mais une occupation mémoire réduite. Les résultats sont cependant susceptibles d’évoluer car Kymatio est mis à jour fréquemment.

5.2.2.2 Précision en fonction de la taille du set d’entraînement

Les filtres de convolution des CNNs nécessitent en général beaucoup de données pour être correctement entraînés. La transformée diffusante utilisant des banques de filtres pré-définis, il n’est donc pas nécessaire de les entraîner, celle-ci devrait donc obtenir de meilleures performances par rapport aux CNNs pour un jeu de données d’entraînement de petite taille.

On compare donc les performances d’un réseau CNN classique avec un réseau hybride « Scattering + CNN » où les premières couches du réseau sont remplacées par la transformée diffusante. On adapte le réseau hybride de manière à obtenir le même nombre

de paramètres pour les deux architectures. Les deux types de réseaux sont entraînés sur le même dataset d’entraînement généré aléatoirement à partir du dataset complet. Les réseaux sont entraînés avec une descente de gradient stochastique, un learning rate de 0.01, momentum de 0.9, weight decay de 0.0005. On divise le learning rate par 10 tous les 60 epochs, et on entraîne les réseaux sur environ 300 epochs.

Les figures ci-dessous donnent le score de classification sur trois jeux de données. Les abscisses sont en échelle logarithmique.

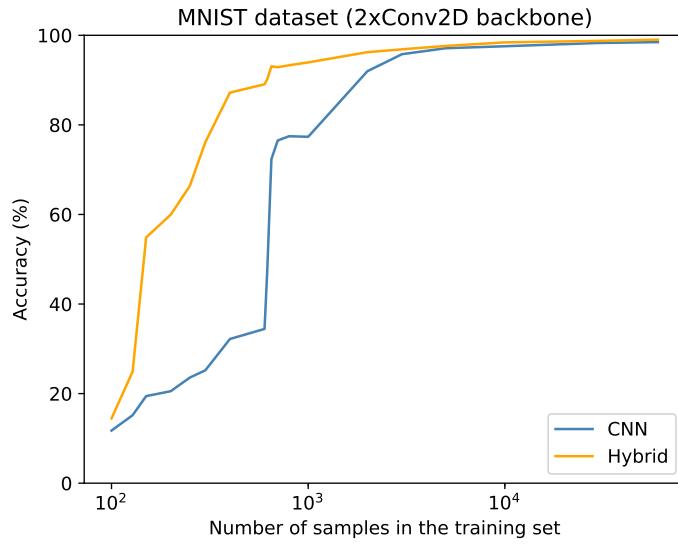


FIGURE 5.14 – Jeu de données MNIST. 10 classes, taille de batch de 128 images (28×28). On constate bien la supériorité du réseau hybride lorsque peu d’échantillons d’entraînement sont disponibles. Lorsqu’on atteint environ 3000 images, les scores des deux réseaux convergent vers les mêmes valeurs.

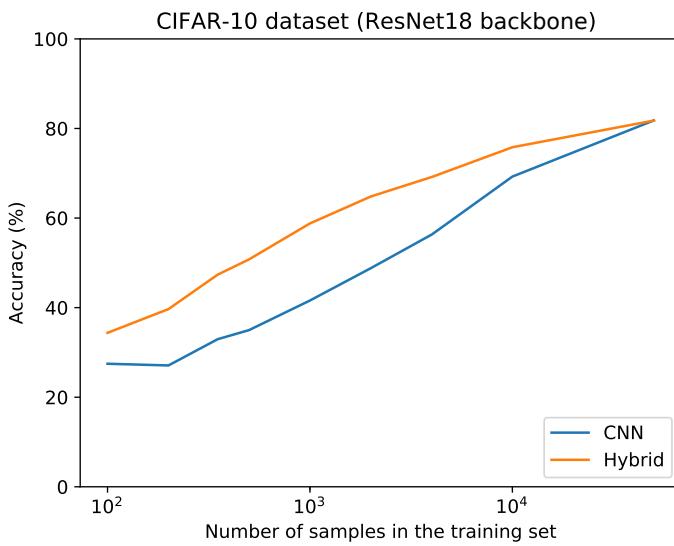


FIGURE 5.15 – Jeu de données CIFAR-10. 10 classes, taille de batch de 64 images (32×32). Le réseau hybride fonctionne mieux que le réseau classique pour un petit dataset d’entraînement. Lorsque beaucoup d’échantillons d’entraînement sont disponibles, les performances des deux réseaux sont similaires.

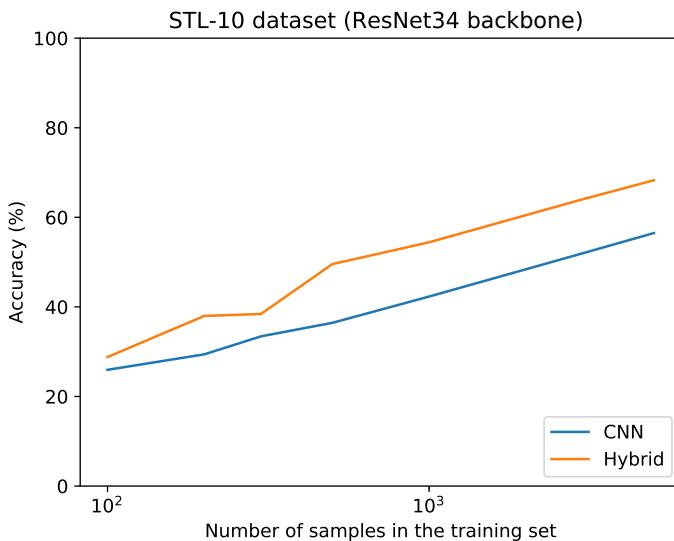


FIGURE 5.16 – Jeu de données STL-10. 10 classes, taille de batch de 64 images (96×96). Ici encore le réseau hybride fonctionne mieux que le CNN. Le dataset STL-10 possédant assez peu d’images (5000 pour 10 classes), le réseau hybride présente ici un réel avantage.

Dans les 3 cas on constate effectivement bien la supériorité du réseau hybride par rapport au réseau purement convolutif lorsque l’on manque d’échantillons d’entraînement. Cet écart de performance s’explique majoritairement par le fait que les filtres du CNN nécessitent beaucoup d’exemples pour être optimaux.

Il existe cependant diverses techniques qui permettent de pallier ce problème. On peut par exemple générer des données supplémentaires en utilisant de l'augmentation d'image, c'est à dire en appliquant des opérations (rotations, translations, zooms, déformations, bruits, etc.) aux images originales du set d'entraînement. On augmente donc artificiellement la taille du jeu de données ce qui permet de mieux entraîner les filtres de convolution.

La figure ci-dessous reprend les résultats obtenus sur la figure 5.14 de manière à comparer ceux-ci sur un entraînement avec augmentation d'image.

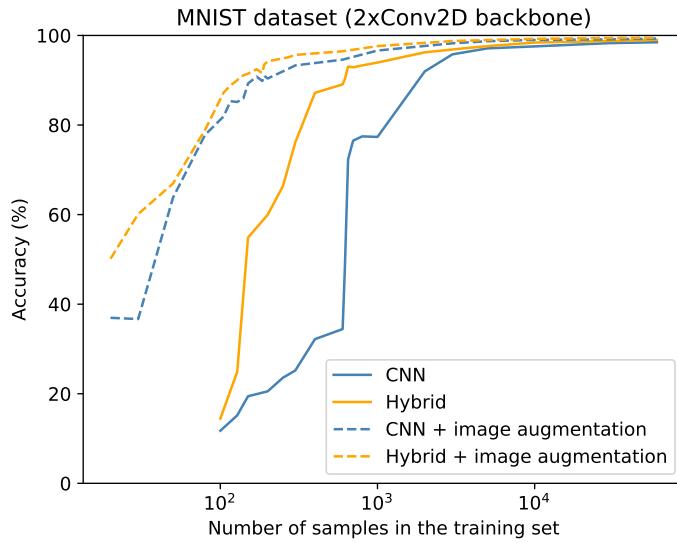


FIGURE 5.17 – Jeu de données MNIST avec augmentation d'image.

On constate que l'augmentation artificielle du set d'entraînement est particulièrement efficace. Pour les réseaux CNNs, on obtient 95% précision avec seulement 500 images, alors qu'il en fallait plus de 3000 pour obtenir la même précision précédemment. Avec augmentation d'image, les réseaux diffusants sont toujours supérieurs aux CNNs mais la marge est plus faible.

Il est aussi tout à fait possible de faire du « transfer learning » ou du « fine-tuning » en utilisant les poids d'un réseau pré-entraîné sur ImageNet par exemple [31] [32].

5.2.2.3 Classification de textures

Comme évoqué dans la section théorique sur les réseaux diffusants, ces architectures donnent de bons résultats lorsqu'il s'agit de classifier des images représentant des textures. L'objet de cette section est de comparer les capacités d'un Scattering Network et d'un CNN simple en terme de précision pour de la classification de textures.

Pour toutes les expérimentations qui suivent dans cette section, on utilise la bibliothèque Scatnet pour calculer les transformées diffusantes sous Matlab et la Deep Learning Toolbox de Matlab pour concevoir les réseaux de neurones. Le data set utilisé est celui introduit en section 5.2.1.2 (UIUC textures).

Test d'un réseau de convolution avec transformée diffusante Voici l'architecture du réseau utilisé pour produire les résultats présentés ci-après :

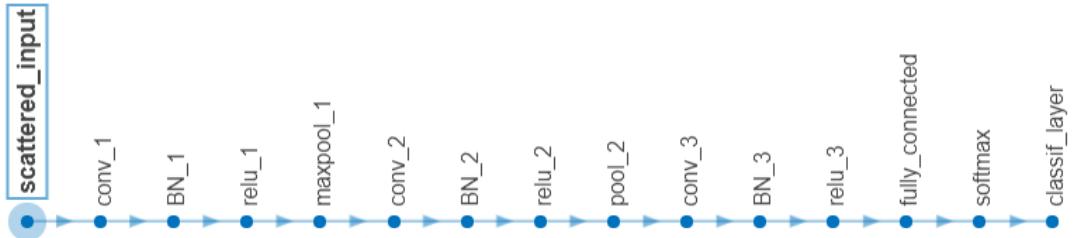


FIGURE 5.18 – Architecture du réseau de convolution avec transformée diffusante.

Le réseau prend en entrée les images de la base d'entraînement ayant déjà subi la transformée diffusante. La couche d'entrée « scattered input » remplace donc la ou les couches de la transformée diffusante et permet donc de réduire le temps d'entraînement du réseau de neurones qui n'a pas à recalculer à chaque époque la transformée diffusante de chaque image.

Cette couche d'entrée est suivie deux blocs classiques convolution, batch normalization, ReLu, max pooling (convolution 3x3 avec padding pour conserver la taille des images et stride de 1, max pooling avec stride de 2), puis du même bloc sans le max pooling. La première couche de convolution compte 8 filtres, ce nombre double à chaque couche de convolution. Enfin on rencontre un fully connected layer de 25 neurones correspondant aux 25 classes du data set.

On teste cette architecture sur les bases d'images déjà transformées d'abord avec la transformée diffusante d'ordre 1 et $J = L = 5$ puis avec la transformée diffusante d'ordre 2 et $J = L = 5$. Le réseau est entraîné sur 20 époques avec des mini-batches de 16 images, le learning rate décroît progressivement et une évaluation de la précision du réseau sur un set de validation constitué d'un quart de la base originale est réalisée toutes les dix itérations (courbe noire).

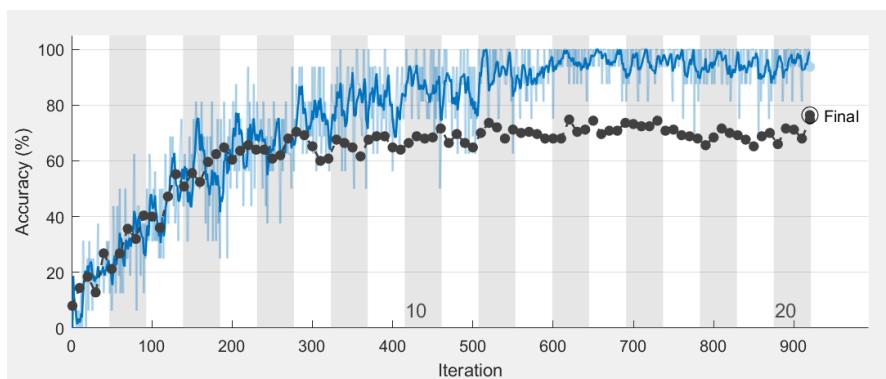


FIGURE 5.19 – Évolution de la précision du réseau au cours de l'entraînement avec transformée diffusante d'ordre 1 : précision sur le set de validation en noir, précision sur le set d'entraînement en bleu.

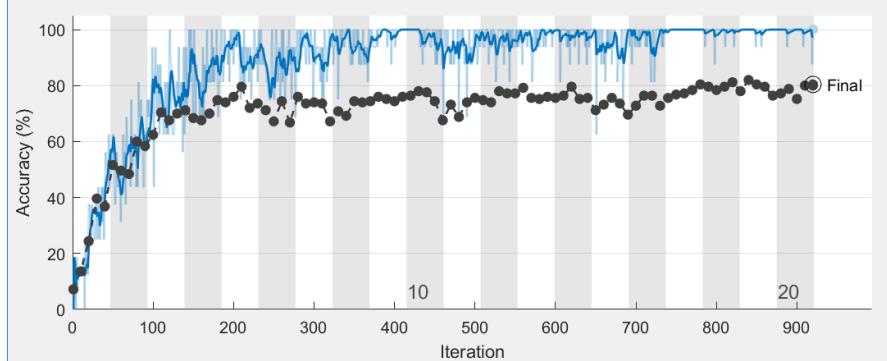


FIGURE 5.20 – Évolution de la précision du réseau au cours de l’entraînement avec transformée diffusante d’ordre 2 : précision sur le set de validation en noir, précision sur le set d’entraînement en bleu.

Les calculs sont une nouvelle fois réalisés sur deux coeurs grâce à la Parallel Computing Toolbox de Matlab. Pour une transformée diffusante d’ordre 1, la précision finale sur le set de validation est de 76.4% obtenue en 2 minutes et 22 secondes contre 80.4% obtenue en 4 minutes pour la transformée diffusante d’ordre 2. On peut donc en conclure que :

- L’ordre 2 est une nouvelle fois légèrement plus efficace mais pour un coût supplémentaire en terme de temps de calcul important.
- Le réseau de convolution avec transformée diffusante s’avère effectivement efficace pour la classification de textures.
- Le réseau de convolution avec transformée diffusante s’avère néanmoins moins efficace que la SVM avec des features bien choisies pour une petite banque d’images telle que UIUC Textures.

Test d’un réseau de convolution classique L’architecture du réseau utilisée est similaire à l’architecture du précédent réseau. On a seulement substitué deux couches de convolution à l’étape de calcul de la transformée diffusante qui était dans la section précédente hors-réseau.

Ce réseau est entraîné exactement de la même façon que le réseau précédent. Voici l’évolution de son entraînement :

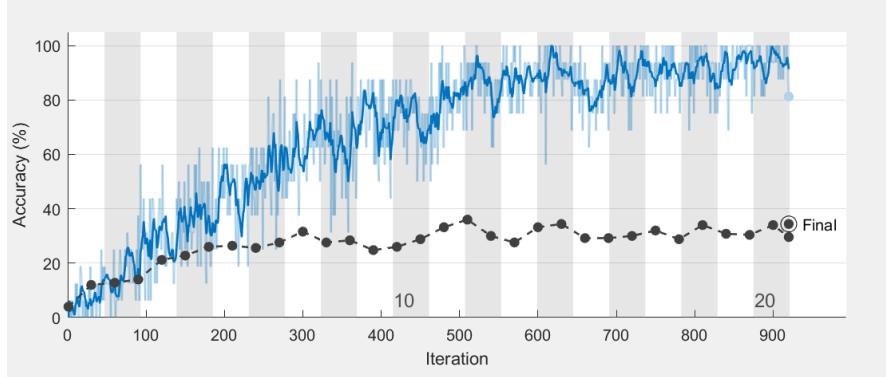


FIGURE 5.21 – Évolution de la précision du réseau de convolution au cours de l’entraînement : précision sur le set de validation en noir, précision sur le set d’entraînement en bleu.

On atteint une précision finale de 34.4% en plus de 32 minutes. Ainsi, le réseau de convolution simple semble bien moins précis que le réseau diffusant sur ce data set avec un écart de 42 points avec ce dernier qui ne s’explique peut-être pas uniquement par le fait que la base d’entraînement est peu peuplée. En effet, les tests menés sur le data set MNIST montrent pour environ 1000 images d’entraînement un écart de précision de l’ordre de 15 points (figure 5.14).

On cherche maintenant à confirmer la supériorité des réseaux hybrides pour la classification de textures avec un data set plus peuplé qui nous permettrait de mettre de coté l’argument de la taille du set de données : le DTD (Describable Textures Dataset).

Ce set de données est constitué de 47 classes contenant 120 images chacune soit un total de 5640 images. Ces dernières sont en couleurs et de dimension variables allant de 300×300 à 640×640 . De plus, ce data set a la particularité d’être composé d’images « de tous les jours » et non d’images obtenues en éclairant de différentes manières des surfaces comme cela pouvait être le cas pour le data set UIUC. Cette particularité et la nature déjà abstraite du concept de texture le rendent particulièrement compliqué à traiter pour un classifieur. Voici un échantillon quatre images appartenant à la même classe « porous » qui témoignent de cette difficulté.



FIGURE 5.22 – Exemples d’images provenant de la même classe du set de données DTD.

Les calculs menés sur ce set de données sont réalisés avec Kymatio et PyTorch sur des architectures de réseaux hybrides et de réseaux de convolution similaires à celles utilisées pour le set UIUC. Trois tests sont réalisés : d’abord avec un réseau de convolution classique, puis avec un réseau hybride avec transformée diffusante d’ordre 1 et enfin avec un réseau hybride avec transformée diffusante d’ordre 2. On obtient les précisions présentées dans le tableau ci-après.

	CNN	HybridNet M=1	HybridNet M=2
UIUC	34%	76%	80%
DTD	3%	7%	34%

Ces résultats peuvent sembler assez mauvais, mais comme mentionné précédemment, ce problème de classification est particulièrement compliqué et on obtient quand même pour un réseau hybride d’ordre 2 un bon gain en précision par rapport à l’aléatoire (qui donnerait tout juste 2% de précision). On peut alors poursuivre nos conclusions :

- Le réseau hybride d’ordre 2 est cette fois-ci beaucoup plus efficace que le réseau hybride d’ordre 1.
- Le réseau hybride est une nouvelle fois beaucoup plus efficace pour la classification de textures.
- Le réseau de convolution est cette fois-ci complètement inadapté à la détection de textures avec un score proche de l’aléatoire.

Les tests sur deux sets de données ne permettent pas de généraliser sans risque mais il semble tout de même que les réseaux hybrides soient bien meilleurs que les réseaux de convolution lorsqu’il s’agit de classifier des textures. Il semble également qu’un réseau hybride avec transformée diffusante d’ordre 2 ait un gain de performance assez variable comparé au réseau avec transformée diffusante d’ordre 1 en fonction du dataset.

5.2.2.4 Imitation de réseau diffusant par un réseau convolutif

Dans cette section on s'intéresse à la possibilité d'imiter les couches diffusantes d'un réseau hybride par une architecture de convolution [33]. Cela nous permettra de nous intéresser aux features extraites par les différents types de réseau dans la section suivante.

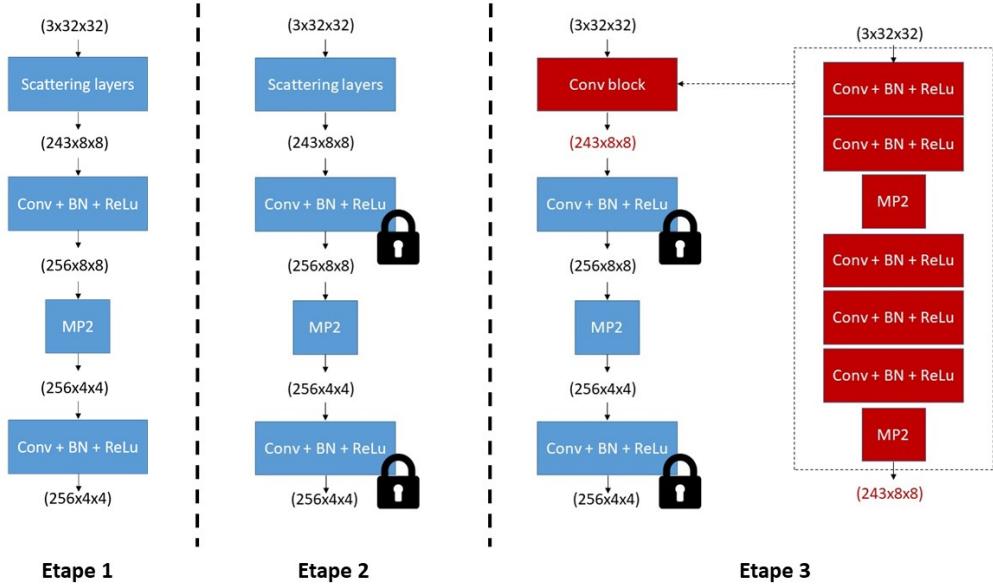


FIGURE 5.23 – Principe de l’expérience.

Pour mener cette expérience à bien, on utilise le data set CIFAR10 composé de 60000 images et on procède en trois étapes illustrées sur la figure ci-dessus :

Étape 1 On entraîne dans un 1er temps un réseau hybride avec une transformée diffusante d'ordre 2 et de paramètres $J = 2$ et $L = 8$ sur le data set. L'architecture de convolution qui suit la partie diffusante est décrite sur la figure ci-dessus. L'entraînement est réalisé avec 128 échantillons/batch et un optimiseur Adam avec un learning rate décroissant selon une courbe exponentielle au fil des 20 epochs de l'entraînement.

Étape 2 Une fois le réseau entraîné, on bloque tous les poids des couches de convolution.

Étape 3 On remplace les couches diffusantes par un bloc de couches de convolutions de sorte à ce que l'on récupère en sortie de ce bloc des données ayant les mêmes dimensions que les données que l'on obtenait en sortie des couches diffusantes sur le réseau hybride. On recommence alors un entraînement dans les mêmes conditions d'entraînement que pour l'étape 1 afin de faire converger les poids des nouvelles couches de convolution.

L'article [33] dont s'inspire cette expérience met en évidence la possibilité d'obtenir le même score de précision avec le réseau de convolution construit à partir du réseau

hybride qu'avec le réseau hybride original. On parvient effectivement à reproduire ce résultat avec 80% de précision pour chacun des deux réseaux.

On en conclut donc que les filtres de convolution peuvent parfaitement imiter les filtres d'ondelettes utilisés par la transformée diffusante pour peu que l'on dispose de suffisamment de données pour entraîner les filtres.

5.2.2.5 Comparaison des attributs extraits

Dans cette section, on cherche à afficher et à comparer les features que les CNN classique et les CNN construits à partir d'un réseau hybride extraient des images sur lesquelles ils s'entraînent. L'idée est de vérifier si les filtres des premières couches de ces réseaux ne tendraient pas naturellement vers des ondelettes 2D. Les tests suivants sont réalisés sur le data set MNIST déjà présenté précédemment et l'extraction des features se fait avec l'implémentation de l'algorithme Deep Dream de la Deep Learning Toolbox de Matlab.

L'algorithme Deep Dream L'algorithme Deep Dream - initialement développé par Google - permet de voir quelle image d'entrée d'un réseau convolutif active le plus fortement un filtre, on peut ainsi voir quel type de feature ce filtre détecte.

L'idée est en fait d'utiliser un réseau déjà entraîné et de définir une fonction de coût à la sortie du filtre sélectionné et qui sera d'autant plus faible que la sortie de ce filtre sera grande. On considère donc simplement la fonction de coût valant l'opposé de la moyenne des coefficients obtenus par la convolution avec le filtre.

Une fois cette fonction de coût définie, on peut alors procéder à l'optimisation de l'image en entrée du réseau. Il faut bien comprendre que les poids du réseau sont verrouillés et que c'est l'image en entrée que l'on optimise.

Voici un exemple avec un réseau très classique souvent donné en exemple pour les problèmes de classification portant sur le data set MNIST.

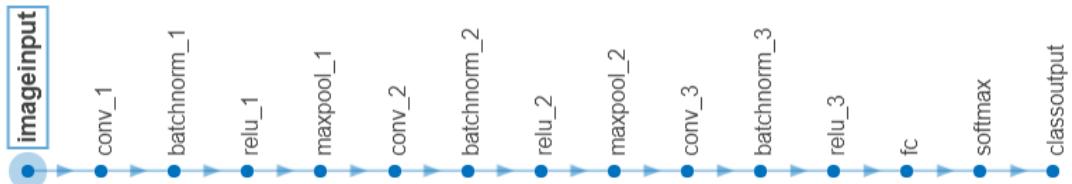


FIGURE 5.24 – Architecture du réseau de convolution.

Les couches de convolution contiennent respectivement 16, 32 et 64 filtres 3×3 avec un padding et un stride de 1 qui conservent la taille des images. Les max pooling ont un stride de 2. Une fois entraîné, le réseau atteint une précision d'environ 92%.

On peut alors essayer de s'intéresser aux features apprises par les différents filtres avec Deep Dream.

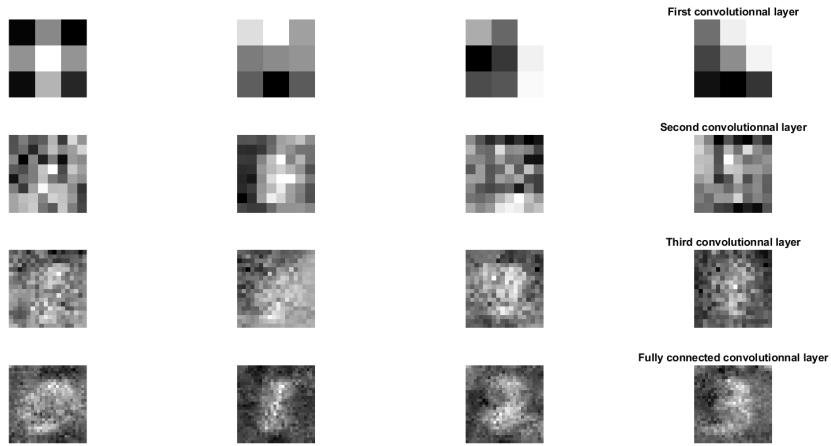


FIGURE 5.25 – De haut en bas : 4 images qui activent fortement 4 filtres parmi 16 pris au hasard dans la première couche de convolution, 4 images qui activent fortement 4 filtres parmi 32 pris au hasard dans la première couche de convolution, 4 images qui activent fortement 4 filtres parmi 64 pris au hasard dans la première couche de convolution, 4 images qui activent fortement les neurones du fully connected layer correspondant aux chiffres 0, 1, 2 et 3.

On remarque que plus on considère une couche basse, plus la feature apprise est élémentaire (un angle, une ligne, un point isolé) et qu'à mesure que l'on se rapproche du fully connected layer, on voit apparaître des features de plus haut niveau jusqu'à pouvoir distinguer des chiffres.

Expérience sur un set d'images classiques Intéressons-nous maintenant au data set CIFAR10 pour mener notre expérience et essayer de mettre en évidence des particularités sur les features apprises dans les premières couches d'un CNN et d'un CNN construit à partir d'un réseau hybride. Les réseaux utilisés sont ceux entraînés dans la section précédente.

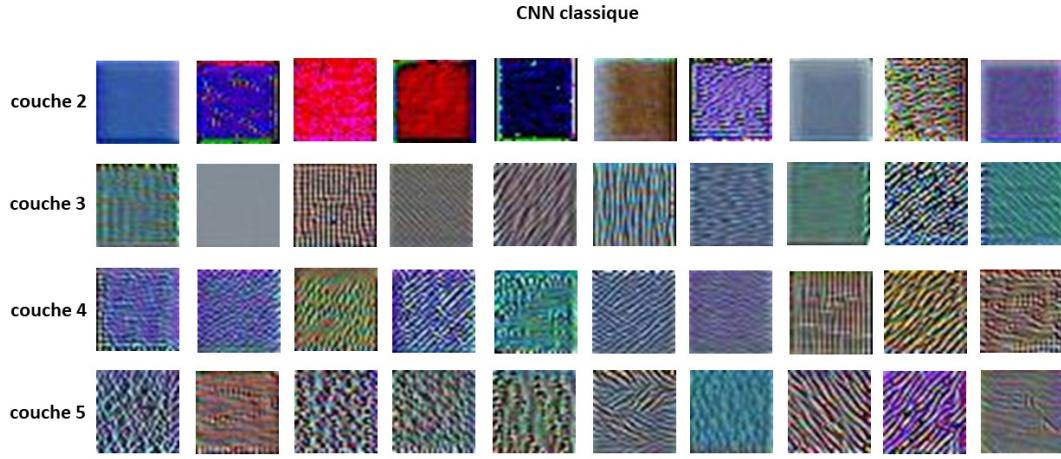


FIGURE 5.26 – Pour chacune des cinq premières couches : un exemple d'image obtenue par optimisation de l'entrée du réseau en fonction de la sortie d'un filtre pour le CNN classique. Les dix filtres affichés sont sélectionnés aléatoirement parmi les filtres de la couche considérée.

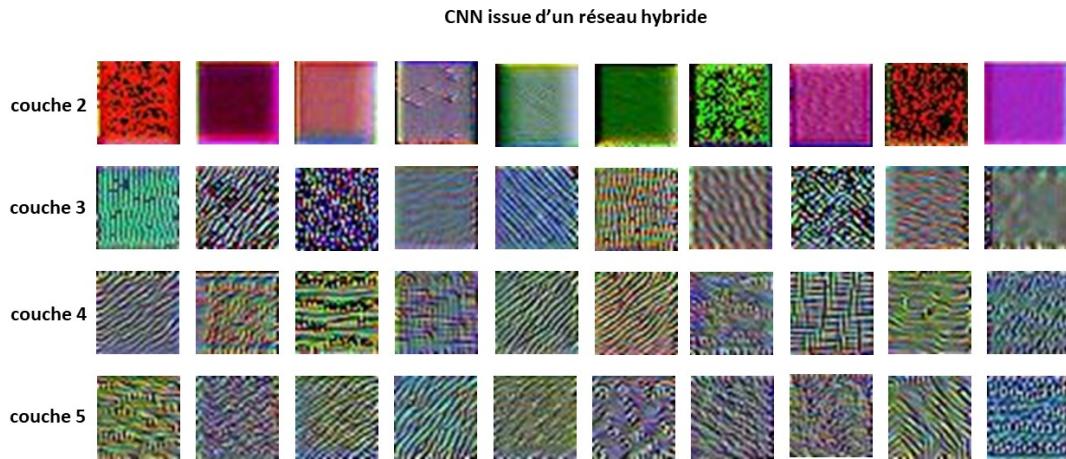


FIGURE 5.27 – Pour chacune des cinq premières couches : un exemple d'image obtenue par optimisation de l'entrée du réseau en fonction de la sortie d'un filtre pour le CNN issu d'un réseau hybride. Les dix filtres affichés sont sélectionnés aléatoirement parmi les filtres de la couche considérée.

Les résultats sont assez décevants, il serait un peu osé d'affirmer que l'on puisse voir quelque chose qui ressemblerait à une ondelette. On peut néanmoins faire quelques

remarques.

D'abord, on observe une nouvelle fois que la complexité des features apprises va en grandissant au fur et à mesure de la profondeur du réseau. Ainsi, on détecte par exemple une simple couleur sur la première couche du CNN classique.

Ensuite, il semble que la méthode d'obtention du CNN n'a pas eu beaucoup d'influence sur les features apprises. En effet, on ne note pas de différences majeures dans la nature des features apprises par les deux CNN pour ces exemples.

Les remarques faites ici sont illustrées par dix filtres par couche tandis que les couches possèdent en réalité de 24 à 256 filtres. Ces remarques ont néanmoins été faites en essayant de considérer le plus possible l'ensemble des filtres de chaque couche mais la tâche est particulièrement compliquée lorsqu'il s'agit de comparer deux groupes de 256 filtres.

Expérience sur un set de textures Des tests similaires effectués avec un data set de textures (UIUC) donnent des résultats complètement différents.

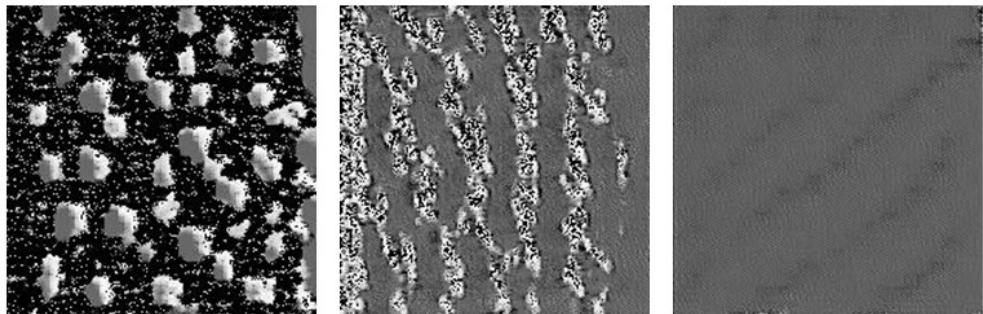


FIGURE 5.28 – Exemples d'images obtenues par optimisation en fonction de la sortie de filtres de la dernière couche de convolution de la partie remplaçant la transformée diffusante (couche 5).

Les motifs sont moins variés quand on observe l'ensemble des images obtenues sur tous les filtres de la cinquième couche mais semblent se rapprocher un peu plus de la forme d'une ondelette.

D'autres architectures de remplacement des couches diffusantes ont été testées pour tenter d'obtenir de meilleures conclusions sur la visualisation des features, sans succès.

5.2.2.6 Robustesse aux « adversarial examples »

Les « adversarial examples » [34] sont des données qui sont suggérées aux systèmes d'intelligence artificielle de manière à les induire en erreur. Du fait que les réseaux de neurones convolutifs ne voient pas les images de la même manière que les humains, il est possible de placer un « overlay » (invisible à l'œil nu) sur une image cible, avec pour

conséquence que le réseau de neurones va reconnaître un objet qui n'a rien avoir avec l'objet présent dans l'image.

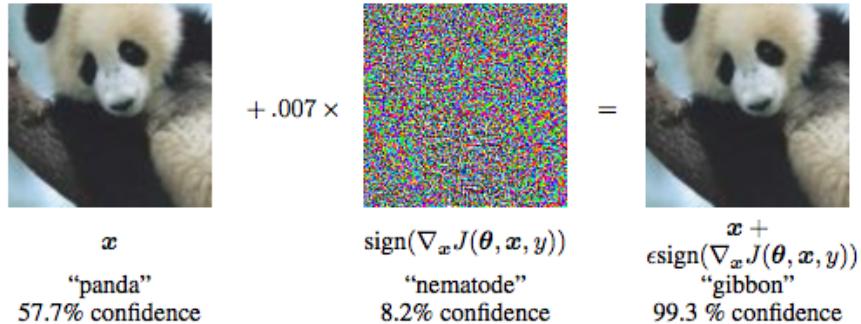


FIGURE 5.29 – Exemple d'aversarial attack [34].

Nous avons donc testé les adversarial examples sur trois différents réseaux :

- réseau purement convolutif
- réseau hybride convolutif (transformée diffusante puis réseau convolutif)
- réseau hybride dense (transformée diffusante puis réseau de neurones simples)

On génère un bruit b superposable à l'image de base en optimisant la fonction de coût suivante où y_target est la classe ciblée :

$$L = \text{MSE}(0, b) + \text{cross_entropy}(y_target, y_pred)$$

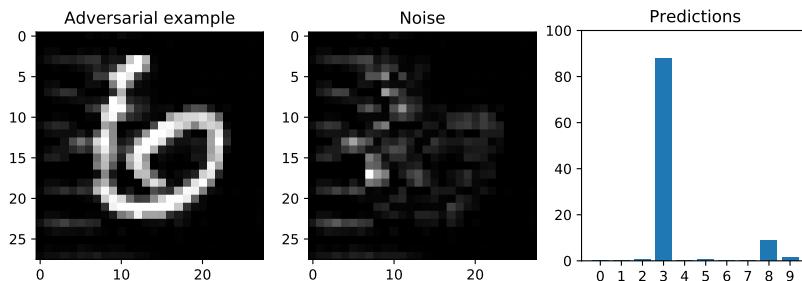


FIGURE 5.30 – Exemple d'« adversarial attack ».

De manière à tester la robustesse de chacun des réseaux, 10 images provenant du jeu de données MNIST sont choisies aléatoirement puis on génère un bruit pour chacune d'entre elles et chacune des architectures avec une classe cible préalablement définie. On teste ensuite les images bruitées sur les réseaux et on note la probabilité prédite pour la vraie classe de l'image (avant bruitage).

	CNN	Scattering + CNN	Scattering + Dense
Médiane	1.6 %	48.6 %	47.8 %
Moyenne	3.0 ± 4.0 %	40.6 ± 30.9 %	52.8 ± 31.3 %

TABLE 5.5 – Précisions sur 10 images du set MNIST.

On constate tout d'abord que les réseaux classiques CNNs sont très peu robustes à ce type d'attaque. Les réseaux hybrides semblent être en moyenne plus robustes mais pas infaillibles, car les résultats sont assez variables selon l'image et la classe cible choisies.

En effet, les réseaux convolutifs étant instables et la transformée diffusante stable, il n'y a pas de raison que l'ensemble « Scattering + CNN » soit stable. Les résultats de cette étude corroborent les conclusions de [35].

5.2.2.7 Propriétés d'invariance

Invariance par translation Lorsqu'on cherche à détecter un objet dans une image, nous souhaiterions que la probabilité prédicta par le réseau pour la classe correspondante soit la même peu importe la position de l'objet dans l'image : c'est l'invariance par translation.

Illustrons l'invariance par translation avec un exemple. Sur la figure ci-dessous, un réseau composé de deux couches de convolutions avec pooling et de deux couches « fully connected ».

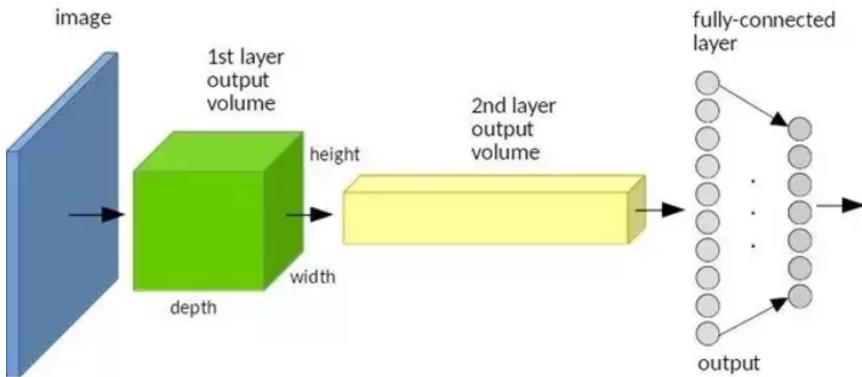


FIGURE 5.31 – Réseau considéré.

Prenons comme hypothèse que les filtres de la première couche (qui génèrent le volume vert) détectent les yeux, le nez et d'autres formes de base (dans les vrais CNNs, les filtres de la première couche détectent des features beaucoup plus abstraites). Les filtres de la deuxième couche (qui génèrent le volume jaune) détectent les visages, les jambes et les autres objets qui sont des agrégations des filtres de la première couche (qui permettent donc de détecter des objets plus complexes).

Dans la figure 5.32, on place en bas à gauche de l'image un visage, représenté par trois points (deux yeux et un nez). Les deux yeux sont détectés par le premier filtre et représentent donc deux activations à la première tranche du volume vert. Il en va de même pour le nez, sauf qu'il est détecté par le deuxième filtre et qu'il apparaît à la deuxième tranche du volume vert. Ensuite, le filtre qui détecte les visages constatera qu'il y a deux yeux et un nez l'un à côté de l'autre dans le volume vert, et génère donc une activation dans le volume jaune (dans la même région du visage à l'entrée de l'image). Enfin, la couche entièrement connectée détecte qu'il y a un visage (et peut-être

une jambe et un bras détectés par d'autres filtres) et elle active donc le neurone de sortie qui indique qu'un humain a été détecté.

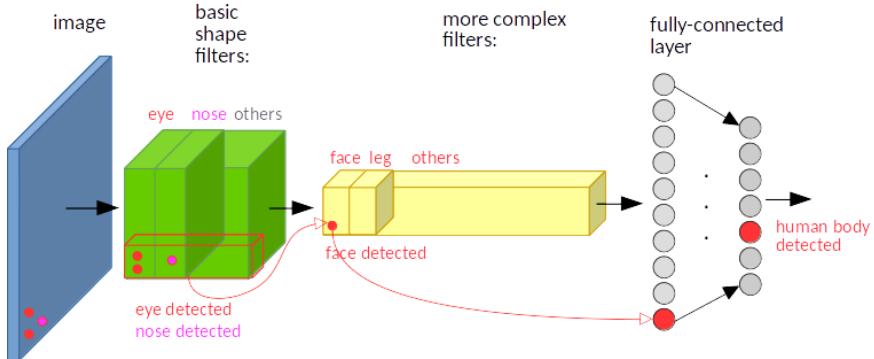


FIGURE 5.32 – Illustration. [Source](#).

Prenons maintenant le même cas mais cette fois-ci les trois points sont situés en haut à gauche de l'image 5.33. Le processus de détection est le même que précédemment, mais les activations se produisent dans une région différente des volumes vert et jaune. Par conséquent, tout point d'activation à la première tranche du volume jaune signifie qu'un visage a été détecté, indépendamment de l'emplacement du visage. Cependant, le chemin d'activation à l'intérieur de la couche entièrement connectée est désormais différent, ce qui signifie que la couche « fully-connected » doit apprendre qu'un visage peut éventuellement apparaître aux deux endroits différents comme dans notre exemple.

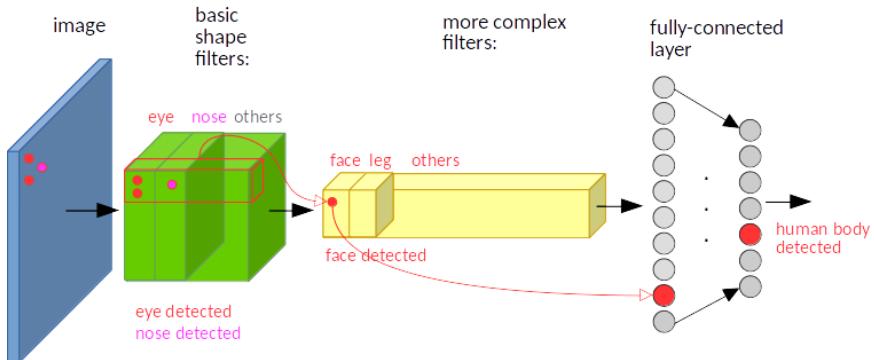


FIGURE 5.33 – Illustration. [Source](#).

Si jamais notre jeu de données est biaisé, c'est à dire que seulement des visages sont présents dans une partie de l'image (au centre par exemple), la garantie d'invariance par translation ne pourra pas être respectée (si un visage est placé dans les coins d'une image par exemple).

L'exemple précédent a permis de montrer que les couches de neurones entièrement connectées semblent être un obstacle à l'invariance par translation. Cependant, les architectures récentes de deep learning n'utilisent plus ces couches mais plutôt des couches de « Global Average Pooling », qui consistent à faire une moyenne par tranche de « feature map ».

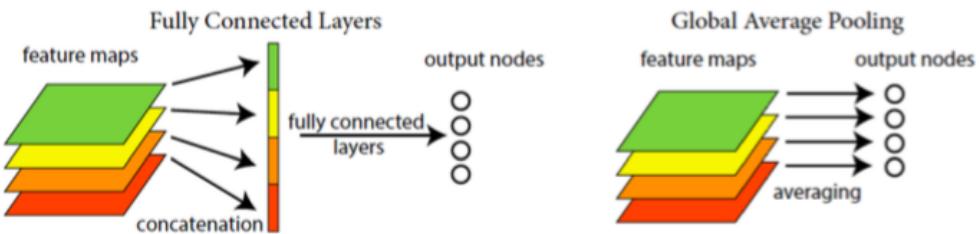


FIGURE 5.34 – Fully-connected et Global Average Pooling. Source.

Émettons désormais l'hypothèse que les tranches verte jaune et rouge de la figure 5.34 permettent de détecter respectivement un nez, des yeux et une bouche. Le neurone de la dernière couche correspondant à la classe « visage » sera donc fortement connecté à ces trois « feature maps ». Le problème soulevé précédemment n'est donc plus présent, car peu importe la position des features dans chacune des feature maps l'opération de moyennage permet de ne garder que le « what » sans prendre en considération le « where ».

On teste désormais un réseau de classification d'image ResNet18 [10], composé de couches de convolutions, de max-pooling, et d'une couche de GAP avant la couche entièrement connectée qui contient autant de neurones qu'il y a de classes. Le réseau est entraîné sur des images de taille 32×32 du dataset STL-10 sans utiliser d'augmentation d'image, puis on choisit ensuite une image du set de test sur laquelle on applique des translations horizontales. On reporte sur les figures ci-dessous la probabilité que l'image contienne bien un objet de la classe correspondante en fonction du décalage horizontal.



FIGURE 5.35 – Probabilité que l'image contienne la classe « horse » en fonction du décalage horizontal.



FIGURE 5.36 – Probabilité que l'image contienne la classe « car » en fonction du décalage horizontal.

On constate sur les deux exemples ci-dessus que la propriété d'invariance par translation n'est absolument pas respectée, et ce même pour des décalages de quelques pixels. L'incapacité des CNNs à généraliser correctement sur des images translatées est donc surprenante. Intuitivement, il semblerait que si toutes les couches d'un réseau sont convolutives, la représentation devrait simplement se translater d'autant que l'image a été translatée. Si les caractéristiques finales pour la classification sont obtenues par une opération de GAP, alors ces caractéristiques devraient être invariantes par translation. Où cette intuition échoue-t-elle ?

Cette intuition ignore en fait l'opération de sous-échantillonnage « max-pooling » qui prévaut dans les CNNs modernes, également connue sous le nom de « stride ».

Prenons un exemple de réseau simple où l'on aurait seulement deux couches de max-pooling (fenêtre de 4 cases et pas de décalage de 2) suivies d'une couche de GAP. On illustre dans la figure 5.37, 4 cas où une image quelconque est translatée et on calcule la sortie.

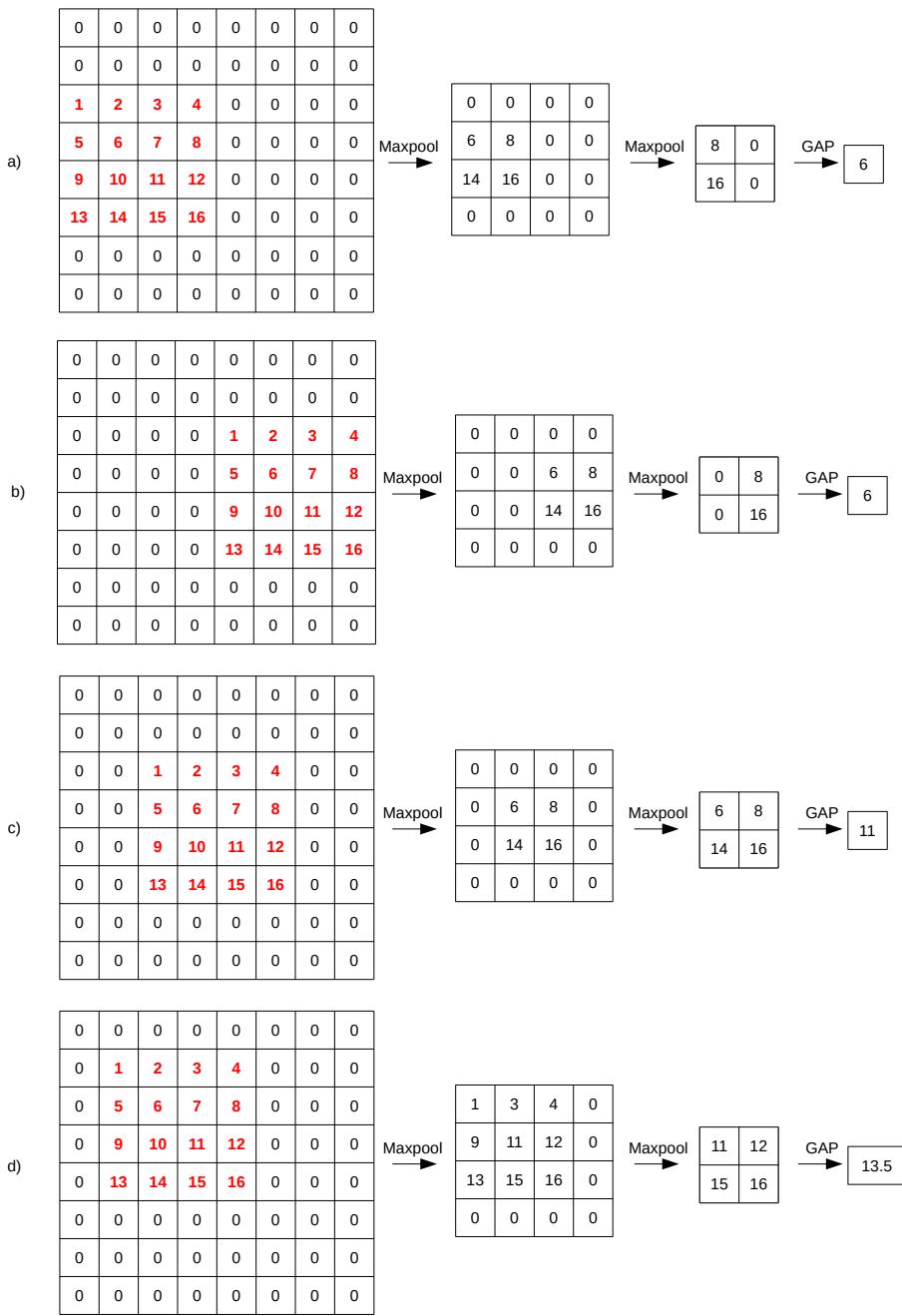


FIGURE 5.37 – 4 cas d'une image translatée de différentes manières. L'image est représentée en rouge.

On constate que la sortie du réseau n'est pas la même dans les quatre cas. [36] explique en fait que la convolution avec max-pooling, qui est en fait une opération de sous-échantillonnage, ne respecte pas le théorème classique de l'échantillonnage, ou théorème de Shannon. Aussi, [37] affirme que l'on ne peut pas s'attendre à une invariance par translation dans un système basé sur la convolution et le sous-échantillonnage, car en pratique l'invariance ne peut exister uniquement que si la translation est un multiple

de chacun des facteurs de sous-échantillonnage dans le système, ce qui est vérifiable dans la figure 5.37 sur les cas a) et b).

Or dans une architecture de deep learning classique, il n'est pas forcément possible de supprimer les couches de max-pooling car celles-ci permettent de diminuer la taille de la feature map, et donc de pouvoir ajouter plus de filtres de convolutions dans un GPU à mémoire limitée. Dans le cas où l'on supprimerait ces couches de sous-échantillonnage, l'ensemble formé par les couches de convolution et la couche de GAP permettent effectivement d'obtenir une invariance par translation parfaite [36], au prix d'un réseau moins profond, et donc moins performant.

Pour pallier partiellement ce problème, il est possible d'utiliser de l'augmentation d'image de manière à réduire le biais présent dans le set de données.

Le théorème d'échantillonnage suggère également que flouter les feature maps serait un moyen d'imposer cette invariance par translation. Cependant, de telles représentations floues peuvent entraîner une diminution des performances. Alternative, on pourrait utiliser des architectures de réseaux spécialement conçues dans lesquelles l'invariance par translation serait garantie jusqu'à une certaine limite, comme par exemple les réseaux diffusants.

Les réseaux diffusants sont en effet, de par leur construction, invariants par translation jusqu'à une limite de 2^J [30]. Dans le cas où on calculerait la transformée diffusante avec l'échelle maximale $2^J = N$, où N est la taille d'un côté de l'image, on obtiendrait une représentation parfaitement invariante par translation [38]. Cependant, cette représentation éliminerait toute l'information de l'image au dessus de la fréquence 2^{-J} , ce qui résulterait en une perte d'information trop importante pour pouvoir classifier des images complexes.

De manière à avoir une idée un peu plus quantitative de cette invariance, on réalise une expérience assez simple qui consiste à faire translater horizontalement un bloc blanc dans une image. On mesure l'erreur quadratique moyenne entre les coefficients diffusants de l'image référence et ceux de l'image avec le bloc translaté.

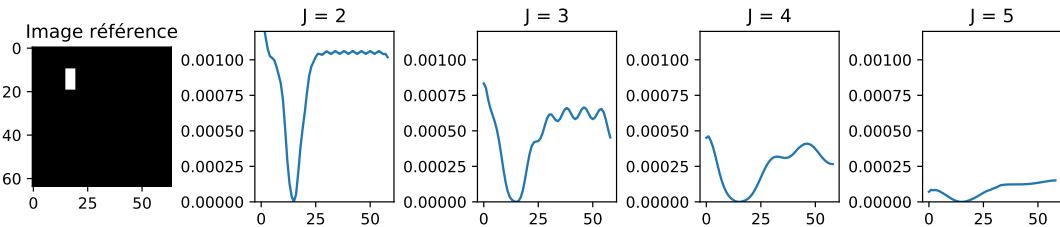


FIGURE 5.38 – Image référence à gauche. Erreurs quadratiques moyennes en fonction du décalage du bloc blanc pour différents J .

Comme prévu, pour des petits J l'invariance par translation est garantie autour d'un intervalle assez faible. Lorsque J augmente, l'invariance par translation est beaucoup plus forte. Cette expérience ne permet cependant pas de montrer que si J augmente on perd en capacité de discrimination.

On ré-itère la même expérience mais cette fois-ci le bloc « référence » est tourné, alors que le bloc translaté **reste vertical**.

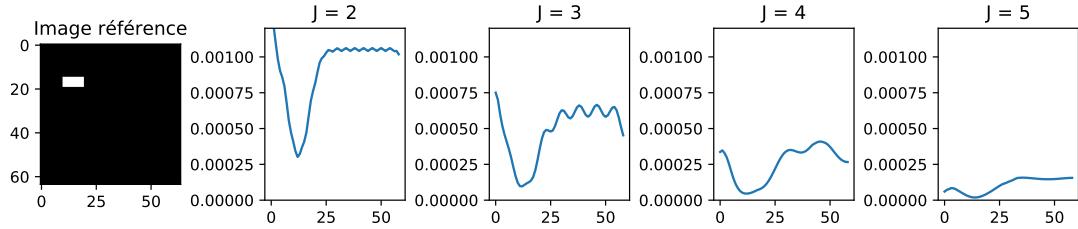


FIGURE 5.39 – Image référence à gauche. Erreurs quadratiques moyennes en fonction du décalage du bloc blanc pour différents J .

Pour $J = 2$, l'erreur quadratique moyenne n'est jamais nulle, il reste donc encore une petite capacité de discrimination même lorsque les blocs (un horizontal, un vertical) sont superposés. Pour $J = 5$, plus aucune capacité de discriminer les blocs car l'erreur moyenne quadratique est proche de 0.

Invariance par rotation L'invariance par rotation peut-être importante dans certains domaines où les objets n'ont pas de sens prédéfinis (milieu biomédical, segmentation de cartes, etc.).

L'approche principale pour apprendre l'invariance par rotation est d'utiliser l'augmentation de données pour fournir au CNN plus d'exemples des mêmes objets avec une orientation différente. Cependant, les CNNs apprendront souvent plusieurs copies du même filtre dans des orientations différentes, ce qui est assez inefficace. D'autres approches [39] tentent de réduire cette redondance, afin de diminuer le nombre de paramètres et donc de réduire le risque de sur-apprentissage.

En ce qui concerne les réseaux diffusants, [40] propose une version améliorée de ces réseaux, appelés « Roto-Translation Scattering Network », et qui sont insensibles aux rotations ainsi qu'aux changements locaux de translation.

L'idée principale est que pour un réseau diffusant à deux couches, la première couche calcule une transformée diffusante sur la variable spatiale $u = (x, y)$ pour réaliser l'invariance par translation locale avec une ondelette 2D classique :

$$\psi_{j,\theta}(u) = 2^{-2j} \psi(2^{-j} r_\theta u)$$

Pour la deuxième couche du réseau diffusant, une ondelette 3D est choisie :

$$\psi_{j,\beta,k}(u, \theta) = \psi_{j,\beta}(u) \bar{\psi}_k(\theta)$$

Les coefficients diffusants ont alors une invariance par translation locale et par rotation grâce à l'introduction d'une dimension θ supplémentaire.

On peut illustrer cette invariance par rotation par une expérience utilisant l'implémentation de la « Roto-Translation Scattering Transform » proposée par la bibliothèque ScatNet en Matlab que l'on appellera RTST dans la suite.

On considère une image exemple pour laquelle on calcule la RTST. On fait ensuite pivoter de $\pi/2$ cette image et on calcule une nouvelle RTST sur cette image pivotée. Enfin on refait pivoter dans le sens contraire toutes les imagettes obtenues par RTST

pour cette deuxième image. L'erreur quadratique moyenne obtenue entre les deux RTST est de l'ordre de 10^{-7} .

On réalise la même suite d'opérations sur la même image avec une transformée diffusante classique. L'erreur quadratique moyenne obtenue entre les deux scattering transforms est de l'ordre de 10^{-1} !

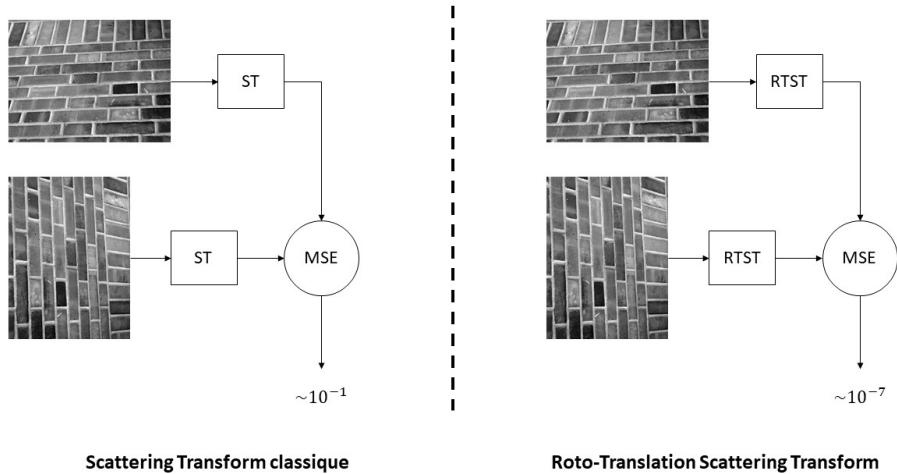


FIGURE 5.40 – Comparaison de la transformée diffusante classique et de la RTST.

La différence est flagrante et met bien en exergue l'invariance par rotation de la RTST.

On peut d'ailleurs profiter de cette propriété pour vérifier que l'on a alors un gain de précision sur la classification du set de textures UIUC qui contient de nombreuses images qui sont parfois quasiment identiques à une rotation près.

On calcule les coefficients de la RTST sur 1000 images du data set, et on obtient après une recherche par grid-search identique aux recherches précédentes un score maximum de 93.5% avec une SVM soit 2.5% de mieux qu'avec une scattering transform classique.

L'utilisation de la RTST comme outil d'extraction de features est donc meilleure que la transformée diffusante classique dans le cas où l'invariance par rotation est nécessaire (comme sur les textures).

Invariance d'échelle L'invariance d'échelle est une propriété assez importante où l'on souhaiterait qu'un objet dans une image soit détecté peu importe sa taille. Dans le domaine de la détection d'objets, l'invariance d'échelle est un challenge essentiel que beaucoup d'architectures ont essayé de résoudre².

Les CNNs, composés de couches de convolutions, de pooling et de GAP ne possèdent donc intrinsèquement aucune propriété permettant une certaine invariance d'échelle,

2. What do we learn from single shot object detectors (SSD, YOLOv3), FPN and Focal loss (RetinaNet)?

même si selon [41] les couches de pooling introduiraient un certain niveau d'invariance aux petites distorsions.

L'approche principale pour apprendre l'invariance d'échelle est d'utiliser de l'augmentation de données pour fournir au CNN plus d'exemples des mêmes objets avec des échelles différentes. Il est également possible de former plusieurs CNNs spécialisés dans chaque échelle et de combiner leurs prédictions. Enfin, d'autres approches consistent à intégrer l'invariance d'échelle dans l'architecture elle-même, comme dans l'architecture Inception [42] ou dans la détection d'objets avec les FPNs [43]. On pourrait aussi pallier ce problème en utilisant des convolutions déformées ou dilatées par exemple.

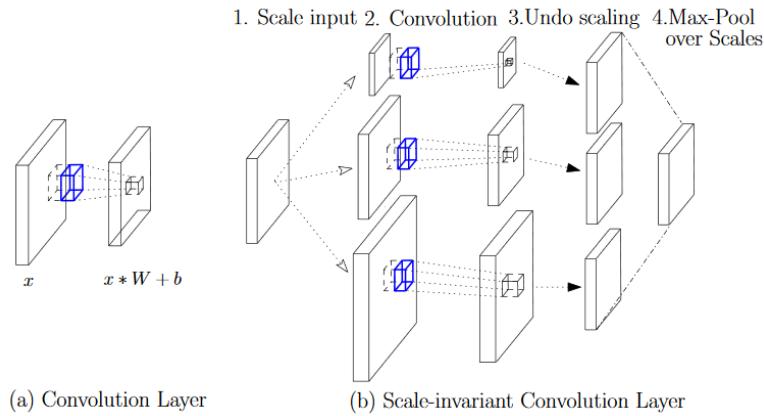


FIGURE 5.41 – Exemple de convolution invariante par échelle [44].

En ce qui concerne les réseaux diffusants, l'invariance d'échelle est démontrée théoriquement pour des petites déformations [30], sans avoir pour autant savoir jusqu'à quel point l'invariance est garantie.

Chapitre 6

Conclusion

Au cours de la dernière décennie, l'apprentissage profond a eu un impact considérable sur la vision par ordinateur en atteignant des performances jamais atteintes auparavant sur de nombreuses tâches telles que la classification d'images, la détection d'objets ou encore la segmentation d'images.

Ces progrès ont notamment été rendus possibles grâce à la démocratisation des GPUs, à la création de librairies telles que TensorFlow, Theano, Keras, Torch ou Caffe, à l'augmentation des données annotées disponibles et à la participation de la communauté aux codes source ouverts et au partage des modèles.

Cependant, les CNNs donnent toujours cette impression de « boîtes noires ». D'un point de vue théorique, les réseaux neuronaux profonds ne sont pas bien compris en raison de leur propriété non convexe. Malgré de nombreux efforts, il n'a jamais été trouvé de preuve de convergence vers de bons minimums locaux. Ainsi, la plupart des recherches effectuées dans ce domaine sont expérimentales et empiriques [45]. D'un point de vue pratique, les CNNs nécessitent un grand nombre d'échantillons d'entraînement ce qui ne leur permet pas de généraliser lorsqu'ils sont entraînés sur de petits sets de données.

À l'opposé, la transformée diffusante a une bonne interprétabilité et un solide fondement théorique. On sait notamment qu'elle construit des invariances aux variabilités géométriques, telles que les rotations et les translations locales.

De plus, il est évident que les réseaux diffusants souffrent moins de problèmes de sur-apprentissage que les réseaux convolutifs car les filtres n'ont pas besoin d'être entraînés. Lors de l'entraînement d'un réseau hybride, la partie convulsive est donc entraînable plus facilement car les premières couches du réseau diffusant arrivent déjà à extraire des attributs pertinents.

Nous avons aussi pu montrer dans notre étude que la transformée diffusante d'ordre 1 réduit la taille du signal d'entrée et préserve la plupart des informations nécessaires pour discriminer et reconstruire une image naturelle, et ce, sans aucun apprentissage.

De plus, celle-ci généralise mieux sur les petits datasets que les réseaux convolutifs classiques, ce qui présente un avantage certain dans les domaines où l'obtention de données est difficile. Cependant, l'augmentation d'image où le « transfer learning » permettent de partiellement corriger ce problème.

Nous avons aussi pu démontrer que la transformée diffusante est remplacable par des couches de convolutions au prix de plus de paramètres à apprendre et potentiellement plus d'instabilités (adversarial examples).

Nous retenons donc que les réseaux hybrides « Scattering + CNN » sont une nouvelle classe de réseaux permettant de combiner les performances des CNNs et l'interprétabilité des réseaux diffusants.

Pour finir, nous souhaiterions remercier nos encadrants ainsi que l'équipe de ScatNet et de Kymatio pour leur travail bénévole effectué sur ces librairies, et qui nous ont permis de mener à bien ce projet.

Bibliographie

- [1] J. Bruna and S. Mallat, “Classification with scattering operators,”
- [2] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex,” *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.
- [3] H. Wang and B. Raj, “On the origin of deep learning,”
- [4] K. Fukushima and S. Miyake, “Neocognitron : A self-organizing neural network model for a mechanism of visual pattern recognition,” in *Competition and cooperation in neural nets*, pp. 267–285, Springer, 1982.
- [5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [6] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,”
- [7] S. Ioffe and C. Szegedy, “Batch normalization : Accelerating deep network training by reducing internal covariate shift,”
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.
- [9] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,”
- [10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,”
- [11] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,”
- [12] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, “Deformable convolutional networks,”
- [13] X. Zhu, H. Hu, S. Lin, and J. Dai, “Deformable convnets v2 : More deformable, better results,”
- [14] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-excitation networks,”
- [15] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, “Benchmark analysis of representative deep neural network architectures,”
- [16] J. Andén and S. Mallat, “Deep scattering spectrum,”
- [17] J. Bruna and S. Mallat, “Invariant scattering convolution networks,”
- [18] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

- [19] X. Gao and H. Xiong, “A hybrid wavelet convolution network with sparse-coding for image super-resolution,” in *Image Processing (ICIP), 2016 IEEE International Conference on*, pp. 1439–1443, IEEE, 2016.
- [20] E. Oyallon, S. Zagoruyko, G. Huang, N. Komodakis, S. Lacoste-Julien, M. B. Blaschko, and E. Belilovsky, “Scattering networks for hybrid representation learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2018.
- [21] J. Andén and S. Mallat, “Multiscale scattering for audio classification.”
- [22] T. Adel, T. Cohen, M. Caan, M. Welling, A. study group, A. D. N. Initiative, *et al.*, “3d scattering transforms for disease classification in neuroimaging,” *NeuroImage : Clinical*, vol. 14, pp. 506–517, 2017.
- [23] M. Andreux, T. Angles, G. Exarchakis, R. Leonarduzzi, G. Rochette, L. Thiry, J. Zarka, S. Mallat, J. Andén, E. Belilovsky, J. Bruna, V. Lostanlen, M. J. Hirn, E. Oyallon, S. Zhang, C. Cella, and M. Eickenberg, “Kymatio : Scattering transforms in python,”
- [24] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” 2017.
- [25] J. Andén, L. Sifre, S. Mallat, M. Kapoko, V. Lostanlen, and E. Oyallon, “Scatnet,” *Computer Software. Available : <http://www.di.ens.fr/data/software/scatnet>*, 2014.
- [26] A. Coates, A. Ng, and H. Lee, “An analysis of single-layer networks in unsupervised feature learning,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011.
- [27] E. Oyallon, E. Belilovsky, S. Zagoruyko, and M. Valko, “Compressing the input for cnns with the first-order scattering transform,”
- [28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn : Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [29] S. Lazebnik, C. Schmid, and J. Ponce, “A sparse texture representation using local affine regions,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1265–1278, 2005.
- [30] S. Mallat, “Group invariant scattering,”
- [31] S. Kornblith, J. Shlens, and Q. V. Le, “Do better imagenet models transfer better ?,”
- [32] M. Huh, P. Agrawal, and A. A. Efros, “What makes imagenet good for transfer learning ?,”
- [33] E. Oyallon, “A hybrid network : Scattering and convnet,” 2016.
- [34] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,”
- [35] K. R. Mopuri, A. Ganeshan, and R. V. Babu, “Generalizable data-free objective for crafting universal adversarial perturbations,”
- [36] A. Azulay and Y. Weiss, “Why do deep convolutional networks generalize so poorly to small image transformations ?,”

- [37] E. P. Simoncelli, W. T. Freeman, E. H. Adelson, and D. J. Heeger, “Shiftable multiscale transforms,” tech. rep., MASSACHUSETTS INST OF TECH CAMBRIDGE DEPT OF ELECTRICAL ENGINEERING AND …, 1991.
- [38] S. Mallat, “Understanding deep convolutional networks,”
- [39] S. Dieleman, J. D. Fauw, and K. Kavukcuoglu, “Exploiting cyclic symmetry in convolutional neural networks,”
- [40] E. Oyallon and S. Mallat, “Deep roto-translation scattering for object classification,”
- [41] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, “Handwritten digit recognition with a back-propagation network,” in *Advances in neural information processing systems*, pp. 396–404, 1990.
- [42] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,”
- [43] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,”
- [44] A. Kanazawa, A. Sharma, and D. Jacobs, “Locally scale-invariant convolutional neural networks,”
- [45] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,”