UCLouvain

École polytechnique de Louvain

Indicateurs de généralisation au sein des couches d'un réseau de neurones convolutifs

Auteurs: **Mathias Novak, Antoine Van Hoof**
Promoteur: **Christophe De Vleeschouwer**
Lecteurs: **Laurent Jacques, Simon Carbonnelle**
Année académique 2018–2019
Master [120] : ingénieur civil en informatique
Master [120] : ingénieur civil en mathématiques appliquées

"Ce qui est vrai se dit simplement."

"Il ne faut pas simplifier inutilement, on perd alors une partie de la vérité."

Table des matières

1 Remerciements

Nos premiers remerciements vont bien évidemment à notre promoteur, Christophe De Vleeschouwer. En plus de son savoir transmis et ses critiques importantes durant notre travail, M. De Vleeschouwer nous a vraiment consacré beaucoup de temps. C'est principalement grâce à ces précieuses heures que nous avons pu avancer à bon rythme et dans la bonne direction. Que ce soit par de nombreux mails ou un nombre important de réunions, nous avons beaucoup reçu de sa part. Sans lui, tout cela n'aurait pas été possible, voilà pourquoi nous tenons donc à lui exprimer notre sincère gratitude.

Nous souhaitons également remercier une autre personne importante, Simon Carbonnelle. En plus de son précédent travail, il nous a donné un avis critique et éclairé sur bien des points pour nous remettre sur la bonne voie.

C'est donc l'occasion d'aussi remercier toute la communauté scientifique pour l'information disponible à ce sujet.

Merci à Brigitte Dupont pour son soutien technique, ainsi qu'à l'UCLouvain et tout ce qui a été mis en œuvre pour l'accomplissement de ce travail tel que l'accès à Cassiopée, un puissant ordinateur doté de cartes graphiques adaptées à nos besoins.

Un dernier hommage pour nos familles et amis, qui nous ont soutenus malgré la charge de travail importante, le stress et tous les aléas comportementaux qu'un travail d'une telle ampleur peut amener.

Mille mercis à vous tous.

2 Résumé

Le domaine de l'*I*ntelligence *A*rtificielle (*IA*) est actuellement en plein essor ; les réseaux de neurones sont de plus en plus présents, notamment les réseaux de neurones convolutifs (*convolutional neural networks* ou *CNN*) pour leurs performances parfois remarquables dans la classification et la reconnaissance d'images.

Cependant, malgré l'engouement pour la capacité de généralisation des *CNN* profonds, cette capacité à classifier correctement des images jamais vues auparavant, on ne connaît pas encore leur fonctionnement de manière assez précise. Souvent considérés comme des "boites noires", ces réseaux sont plus ou moins bons dans leurs applications, et cela pour des raisons encore floues. En d'autres termes, la raison d'une bonne généralisation d'un *CNN* n'est pas encore bien connue.

Dans ce travail, nous avons donc comparé des réseaux/modèles entraînés par des algorithmes d'optimisations différents ; via *sgd*, via *sgd* avec *weight decay* et via *sgd* utilisant l'extension *layca* [CV18b]. L'entraînement à l'aide d'algorithmes d'optimisation différents nous a permis d'obtenir des modèles offrant des propriétés observables différentes en termes de généralisation.

En continuation des travaux [CV18b], nous affirmons que la variation de la *cosinus distance* entre le nouveau vecteur de poids généré lors de l'apprentissage et le vecteur de poids initial n'est pas directement liée à la généralisation, mais elle en représente une expression visible. Nous avons alors analysé la façon dont sont générés ces nouveaux vecteurs de poids lors de l'apprentissage et avons conclu qu'une grande cosinus distance est signe d'une aptitude du réseau à mettre à zéro les liens inutiles, améliorant ainsi la robustesse du réseau à l'élagage (*pruning*) de ces poids les plus petits.

On se penche maintenant sur la redondance d'information des activations. Dans les premières couches, on remarque qu'un réseau qui généralise mieux aura un moins grand coefficient de silhouette (*CS*) et une plus petite information mutuelle entre ses neurones (*IM*). Le comportement inverse se produira dans les dernières couches.

L'information mutuelle des neurones avec les classes (*IMC*) montre un comportement similaire, plus important et plus tôt dans le réseau. Elle serait un symptôme direct d'un meilleur apprentissage au sein même de chaque neurone. De plus, il semble que les comportements de l'*IM* et de la *CS* soient conséquents de cette première.

Le meilleur apprentissage de chaque neurone amène à un meilleur choix de sous-classes, directement liées avec la classification finale voulue. On peut donc émettre l'hypothèse qu'un réseau qui généralise mieux est constitué de neurones qui classent et transforment plus efficacement leurs entrées par rapport à l'ensemble des données reçues (images et classes).

L'ensemble de ces expériences nous amène à mieux comprendre le fonctionnement de *sgd*, *weight decay* et *layca*. Effectivement, avec ces deux dernières extensions, on amène le réseau à adopter une structure plus nette ainsi qu'une plus grande diversité dans les biais observés. C'est cette diversité dans l'observation des biais par le réseau qui lui permet de prendre sa décision de classification sur plusieurs caractéristiques, et donc être plus apte à une meilleure généralisation.

On confirme l'hypothèse affirmant qu'un réseau peu profond aura tendance à ne distinguer que les biais superficiels des données. Les performances de généralisation, apportées par la détection des biais désirés, seront ainsi diminuées. Plus un réseau sera profond, plus il y aura de paramètres à ajuster et plus on pourra dégager des sous-classes complexes adéquates qui cernent les bons biais des données servant à la classification voulue. Cette complexité amène une classification plus conceptuelle et intelligente, c'est-à-dire une meilleure généralisation.

Finalement, on observe que les réseaux dégagent plusieurs biais dans les données. Ces biais sont créés de façon à différencier correctement les entrées dans la base de données. De plus, si un réseau observe plusieurs biais en lien avec la classification, cela lui apportera généralement de meilleures performances car il sera alors plus robuste. C'est ce qu'essaie d'apporter l'usage de *dropout*, *weight decay* et *layca* dans l'apprentissage, et c'est ce qui est mis en avant par le *prunning*.

Pour faire simple, l'explication d'une bonne généralisation vient du fait que le réseau observe un meilleur choix de biais.

3 Introduction

"*Plus j'apprends, plus je m'aperçois que je ne sais pas.*"

Galilée

De nos jours, les réseaux convolutifs atteignent des performances de généralisation jamais vues auparavant. On citera entre autres leurs contributions dans le populaire jeu de Go, les voitures autonomes, la découverte de nouveaux médicaments, etc.

Cependant, malgré nos connaissances actuelles sur la grande puissance d'apprentissage des réseaux de neurones convolutifs ainsi que leur efficacité grandissante dans leur tâche, il est encore assez difficile d'expliquer les phénomènes qui vont amener un réseau à bien généraliser.

Pour introduire le phénomène de généralisation, utilisons un exemple. Lorsqu'un réseau de neurones apprend à reconnaître un chat sur une base de données d'images, le réseau va modifier ses paramètres (ses poids) pour assimiler certaines caractéristiques propres aux images représentant un chat. C'est grâce à cet apprentissage que lorsque le réseau fera face à une nouvelle image jamais présentée, il va pouvoir affirmer si c'est une image de chat ou non avec un certain degré de précision. La généralisation dans les *CNN* est donc le phénomène de pouvoir reconnaître un concept (un objet, une personne, un animal, etc) sur une image jamais vue, mais uniquement grâce à l'entraînement d'images similaires vues précédemment. Après entraînement, un réseau qui généralise mieux qu'un autre aura donc un degré de classification plus élevé sur une base de données d'images inconnues.

Dans cette étude, nous nous efforcerons d'identifier les caractéristiques d'un réseau révélant/induisant une bonne généralisation. Notre analyse se fondera sur la comparaison de modèles offrant des qualités de généralisation différentes mais qui ont tous mémorisé [1] les données d'entraînements. Nous regarderons ce qu'il se passe au niveau des poids des couches de ces

[1] Mémoriser dans le sens d'une *training error* nulle et non de l'*overfitting*.

réseaux lors de l'apprentissage ainsi qu'au niveau de l'activation des neurones dans ces couches. Pour résumer, voici donc une question qui nous tiendra en haleine tout au long de ce travail :

Que se passe-t-il au sein même du réseau, au niveau de ses couches, dans son comportement et lors de l'apprentissage qui le mène à bien généraliser ?

La réponse à cette question pourrait être intéressante pour la conception et l'entraînement de futurs réseaux. Voilà pourquoi, afin de mieux comprendre ce phénomène, il faut expérimenter et faire des analyses rigoureuses qui nous sortent de la simple intuition. C'est ce que nous avons essayé de faire concrètement dans ce travail.

Structure de la thèse

Dans la suite de cette étude, nous commencerons par brièvement mettre à jour les concepts de base dans le **Chapitre 4**. Il vous sera ainsi plus aisé de comprendre nos raisonnements futurs.

Nous vous présenterons l'état de l'art dans le **Chapitre 5**. C'est à dire les derniers travaux et avancées du moment qui nous ont en même temps servi d'inspiration et de soutien durant nos recherches.

Dans le **Chapitre 6**, nous discuterons des outils que nous avons utilisés et de la façon dont nos modèles ont été entraînés.

Après cette mise en bouche, nous vous présenterons alors les deux grands champs de recherche de notre travail :

- Le premier est de regarder ce qu'il se passe au niveau des *poids* du réseau lors de l'apprentissage (**Chapitre 7**).
- Le second est de poser un regard sur ce qu'il se passe lors de l'*activation* des neurones face à des stimuli dans un réseau déjà entraîné (**Chapitre 8**).

Enfin dans le **Chapitre 9**, nous tirerons une conclusion sur l'ensemble de nos résultats.

4 Préliminaires

Dans ce chapitre, nous introduirons des concepts pertinents pour comprendre le reste de la dissertation.

Sommaire

4.1 Un bref historique

Avant d'introduire formellement le concept de réseau de neurone, retournons un peu en arrière dans le temps et penchons nous sur les premiers essais visant à simuler le comportement du cerveau humain.

Les premiers essais de modélisation du cerveau humain sont anciens, même plus vieux que l'âge des ordinateurs. Ce n'est qu'assez récemment, en 1943, que le neurophysiologiste McCulloch et le logicien Pitts proposent le premier concept de neurone formel, une représentation mathématique/informatique du neurone biologique. La FIGURE 4.1 montre une représentation schématique du neurone tel qu'imaginé par McCulloch et Pitts.

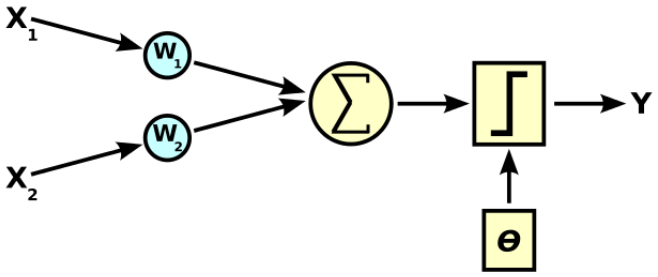


FIGURE 4.1 – Premier neurone formel

Dans ce modèle, un neurone est une fonction qui reçoit des entrées binaires $X_1, X_2, ... \in \{0, 1\}$ (les signaux électriques des neurones précédents) et qui émet une valeur également binaire $Y \in \{0, 1\}$ représentant l'activation du neurone.
Pour calculer cette sortie Y, le neurone effectue une somme $\sum$ pondérée w_i de ses entrées X_i (qui, en tant que sorties d'autres neurones formels, valent aussi 0 ou 1) puis applique une fonction d'activation $\int$ à seuil : si la somme pondérée dépasse une certaine valeur θ, la sortie du neurone vaut 1 (neurone activé), sinon elle vaut 0 (neurone non-activé).

Malgré la simplicité de cette modélisation, ou plus justement grâce à elle, le neurone formel dit de McCulloch et Pitts reste aujourd'hui un élément de base des réseaux de neurones artificiels. De nombreuses variantes ont été proposées, biologiquement plausibles, mais s'appuyant généralement sur les concepts inventés par les deux auteurs. On sait néanmoins aujourd'hui que ce modèle n'est qu'une approximation des fonctions remplies par le neurone réel et, qu'en aucune façon, il ne peut servir pour une compréhension profonde du système nerveux. [Wik19e]

L'idée serait alors d'assembler ces neurones formels sous forme de couches. C'est l'origine du *perceptron*. Ce concept fut utilisé par Rosenblatt en 1959 pour simuler la fonction rétinienne et reconnaître des formes. C'est le premier système artificiel capable d'apprendre par expérience. Nommée 'connectioniste', cette approche a atteint ses limites technologiques mais aussi théoriques durant le début des années 70. [Wik19g]

De nos jours, les réseaux de neurones sont à nouveau en plein développement grâce à des bases de données plus conséquentes et à la puissance des ordinateurs actuels, et cette technologie est en constante évolution.

4.2 Le réseau de neurones

Un réseau de neurones, en anglais *Neural Network (**NN**)*, est constitué de neurones artificiels qui vont être liés ensemble par des liens pondérés. Commençons par regarder d'un peu plus près comment fonctionne actuellement un neurone artificiel.

4.2.1 Le perceptron

Le neurone artificiel, aussi appelé perceptron, est inspiré des neurones biologiques. Comme celui-ci, il va recevoir de l'information sous forme de potentiel électrique en entrée, et suivant la grandeur de ce potentiel, le neurone va transmettre une impulsion électrique plus ou moins grande au neurone suivant [Sin17][Tas16].

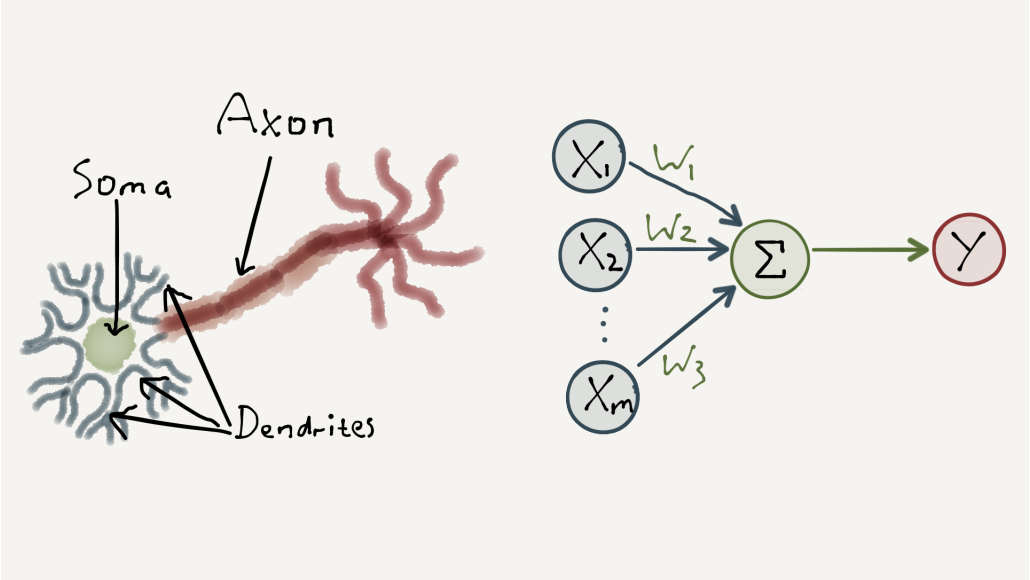


FIGURE 4.2 – Analogie entre neurone biologique et neurone artificiel. Les signaux électriques, provenant des autres neurones, sont reçus au niveau des *dendrites*, composés par le *soma* et le tout retransmis par *l'axone*.

L'analogie peut être vue sur la FIGURE 4.2, où l'on voit les dendrites d'un neurone biologique représentées par les entrées X_i du neurone artificiel, le corps de la cellule représente la somme pondérée de ces entrées, et l'axone représente la sortie.

De manière un peu plus précise, les entrées sont pondérées et sommées, pour ensuite passer par une fonction d'activation dont le résultat est envoyé en sortie. On voit sur la FIGURE 4.3 l'agencement plus rigoureux d'un neurone artificiel.

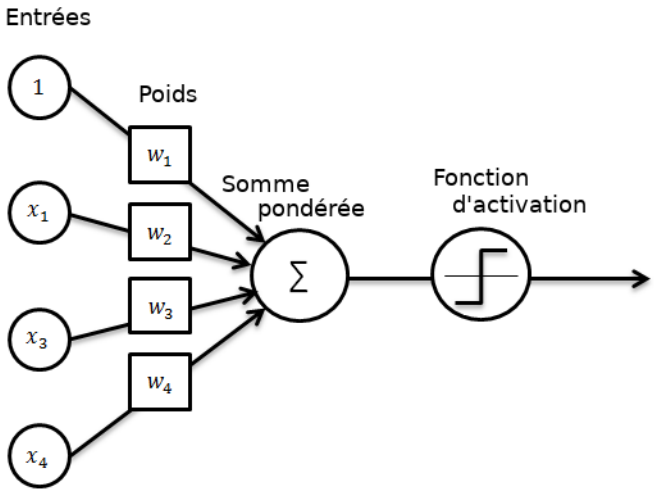


FIGURE 4.3 – Neurone artificiel

Le neurone artificiel va donc ajuster ses poids pour donner plus ou moins d'importance à certaines entrées. Lorsque la somme de ses entrées est suffisamment élevée, l'impulsion est alors assez grande pour que la sortie soit non nulle après passage par la fonction d'activation.

De manière plus mathématique, on considère le cas général d'un neurone formel à m entrées, auquel on doit donc soumettre les m grandeurs numériques (ou signaux, ou encore stimuli) notées x_1 à x_m. Un modèle de neurone formel est une règle de calcul qui permet d'associer une sortie à m entrées : c'est donc une fonction à m variables et à valeurs réelles.

A chaque entrée est associé un poids synaptique, c'est-à-dire une valeur numérique notée de w_1 pour l'entrée 1 jusqu'à w_m pour l'entrée m. La première opération réalisée par le neurone consiste en une somme des grandeurs reçues en entrées, pondérées par les coefficients synaptiques, c'est-à-dire la somme

$$w_1 x_1 + \ldots + w_m x_m = \sum_{j=1}^{m} w_j x_j. \tag{4.1}$$

À cette grandeur s'ajoute un seuil w_0. Le résultat est alors transformé par une fonction d'activation non linéaire (parfois appelée fonction de sortie), φ. La sortie associée aux entrées x_1 à x_m est ainsi donnée par

$$\varphi\left(w_0 + \sum_{j=1}^{m} w_j x_j\right) \tag{4.2}$$

qu'on peut écrire plus simplement :

$$\varphi\left(\sum_{j=0}^{m} w_j x_j\right), \tag{4.3}$$

en ajoutant au neurone une entrée fictive x_0 fixée à la valeur 1.

Le neurone effectue donc un produit entre 2 vecteurs, les entrées et les poids, ce qui est facilement calculable par un ordinateur (voir équation 4.4, sous forme matricielle).

$$S = W^T X \tag{4.4}$$

La somme S passe alors par une fonction de non-linéaire nommée *fonction d'activation* ou fonction de transfert afin de renvoyer en sortie une valeur possible comprise généralement dans les intervalles [0 ;+1] ou [-1 ;+1], selon la fonction d'activation. Les différents types de neurones se distinguent par la nature de cette fonction.

Les principales fonctions d'activation sont :

$$\begin{aligned}
\text{linéaire} \quad & g(x) = x \\
\text{sigmoide} \quad & g(x) = 1/(1+e^x) \\
\text{softmax} \quad & g(x) = \frac{e^x}{\sum^k e^{x_k}} \\
\text{ReLU} \quad & g(x) = \max(0, x) \\
\text{seuil} \quad & g(x) = 1_{[0,+\infty[}(x) \\
\text{stochastique} \quad & g(x) = \begin{cases} 1 & \text{avec un probabilité de } 1/(1+e^{-x/H}) \\ 0 & \text{sinon} \end{cases}
\end{aligned}$$

Les modèles linéaires, sigmoidales, softmax et ReLU sont adaptés aux algorithmes utilisant la rétro-propagation du gradient grâce à leur fonction d'activation différentiable ; ils sont les plus utilisés. Nous utiliserons d'ailleurs les modèles ReLU dans le réseau, ainsi que des modèles softmax plus adaptés pour différencier les classes à la fin du réseau.

Le modèle à seuil est probablement plus enclin avec la réalité biologique mais pose des problèmes d'apprentissage car la fonction qu'il représente n'est pas différentiable, une qualité

nécessaire pour l'apprentissage.

Finalement, le modèle stochastique est utilisé pour les problèmes d'optimisation globale de fonctions perturbées ou pour des analogies avec des systèmes de particules (machine Boltzmann).

En voilà quelques uns des plus connus, mais il en existe bien d'autres...

Pour reprendre un peu de recul et résumer la finalité, un neurone artificiel est un algorithme qui permet de faire une classification sous un *apprentissage supervisé*. Cette classification est dite géométrique, et plus précisément linéaire (schéma sur la figure 4.4).

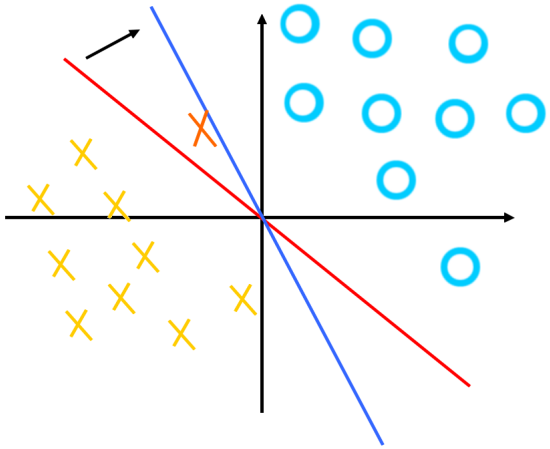


FIGURE 4.4 – Visualisation de l'apprentissage linéaire d'un neurone pour la classification entre les X et les O, par exemple les entrées activées et non-activées.

L'***apprentissage supervisé*** (supervised learning en anglais) est une tâche d'apprentissage automatique consistant à apprendre une fonction de prédiction à partir d'exemples annotés, au contraire de l'apprentissage non supervisé [Wik19a].

4.2.2 L'apprentissage

Nous verrons ici comment un neurone fait pour apprendre. Tout d'abord, il doit connaître son erreur. C'est grâce à la *fonction d'erreur* entre sa sortie et la sortie voulue qu'il va alors adapter ses paramètres. Il va alors propager l'erreur et l'apprentissage se fera en réaction en chaîne grâce à la *rétro-propagation*.

Fonction objectif

La fonction objectif, aussi appelée fonction d'erreur ou *loss function* en anglais, est utilisée en optimisation mathématique pour désigner une fonction qui sert de critère pour déterminer la meilleure solution à un problème d'optimisation. Concrètement, elle associe une valeur à une instance d'un problème d'optimisation. Le but du problème d'optimisation est alors de minimiser ou de maximiser cette fonction jusqu'à l'optimum grâce à un optimiseur choisi.

Cette fonction objectif va être définie via notre sortie désirée y et la valeur prédite par notre réseau $\hat{y}$.

Dans notre cas, nous aurons des images montrées au réseau qui ne représente qu'un seul objet. Ces images ne doivent donc rentrer que dans une seule classe chacune. La fonction objectif catégorielle d'entropie croisée sera donc utilisée dans ce travail.

En effet, cette fonction objectif est adaptée à des problèmes de classification à label unique, c'est à dire quand chaque entrée ne peut être classée que dans une seule catégorie. En d'autres mots, quand chaque image n'appartient qu'à une seule classe [Ano19b][Tic17]. Voici sa formule mathématique 4.5.

$$\begin{aligned} L(y|\hat{y}) &= CCE \\ &= -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{J} y_j \cdot log(\hat{y}_j) + (1-y_j)\cdot log(1-\hat{y}_j) \end{aligned} \tag{4.5}$$

où $\hat{y}$ est la valeur prédite, y est la vraie valeur, N est le nombre d'exemple utilisé et J est le nombre de catégories.

Notons ici sa dérivé 4.6, qui sera utile pour comprendre l'entraînement du réseau.

$$\begin{aligned} \frac{dL(y|\hat{y})}{d\hat{y}} &= \frac{1}{N}\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}} \\ &= \frac{1}{N}\frac{y(1-\hat{y}) - (1-y)\hat{t}}{\hat{y}(1-\hat{y})} \\ &= \frac{1}{N}\frac{y - y\hat{y} - \hat{y} + y\hat{y}}{\hat{y}(1-\hat{y})} \\ &= \frac{1}{N}.\frac{y-\hat{y}}{\hat{y}.(1-\hat{y})} \end{aligned} \tag{4.6}$$

Il faut cependant savoir que bien d'autres fonctions objectifs existent telles que l'erreur quadratique moyenne ($MSE = \frac{1}{N}\sum_{i=0}^{N}(\hat{y}_i - y_i)^2$), l'entropie croisée binaire (BCE), de vraisemblance,...

La rétro-propagation

L'apprentissage profond est une branche du machine learning où les algorithmes apprennent d'une immense masse d'information de manière indépendante. Tout comme les être humains, ces algorithmes deviennent plus intelligents avec l'expérience en rassemblant et en traitant de plus en plus de données. Concrètement, on dit qu'un réseau apprend lorsqu'il change ses poids de manière à diminuer son erreur de classification. Mais comment fait un réseau de neurones pour ajuster autant de poids de manière efficace ?

Les neurones du réseau vont apprendre par un mécanisme de rétro-propagation du gradient ou *back-propagation*. Cet algorithme d'apprentissage permet à tous les neurones du réseau d'apprendre en même temps, et cela de manière assez efficace en terme de temps de calcul sur les ordinateurs actuellement utilisés.

Le principe de base est que la sortie est une fonction linéaire des poids et des entrées. Quand on connaît l'erreur sur cette sortie, on peut alors donner un degré de responsabilité à chacune des entrées pondérées, et ainsi changer les poids pour améliorer les résultats lors d'une prochaine fois (On peut voir ce principe de rétro-propagation sur la figure 4.5). On applique alors cette méthode sur tous les neurones d'une couche, et puis de manière itérative des dernières couches jusqu'aux premières couches.

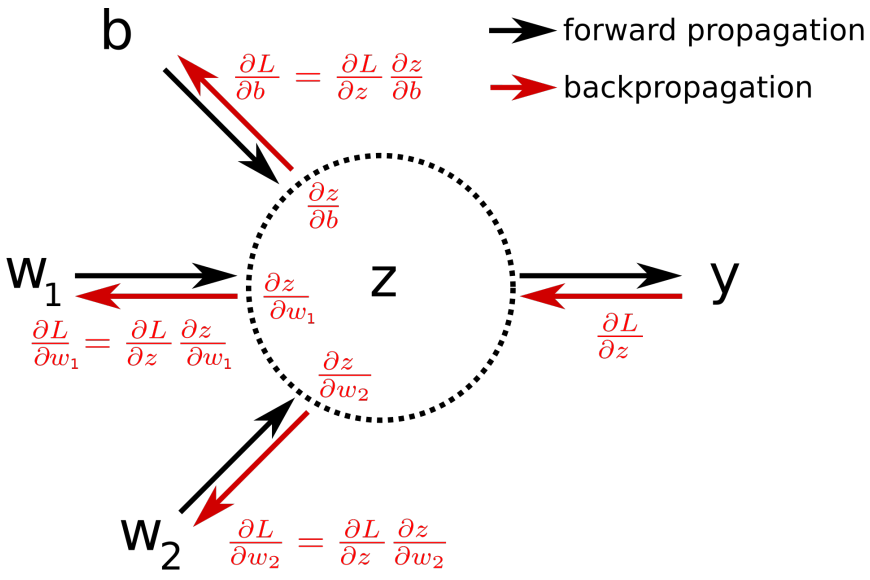


FIGURE 4.5 – Back-propagation dans le neurone : On voit ici que le gradient sur la sortie y est propagée sur le biais b, ainsi que sur chacun des poids w_i.

Ce principe permet d'ajuster tous les poids du réseau dans un même temps. Le réseau va alors finir par apprendre et donc par converger vers une solution globale plus ou moins bonne, c'est à dire qui généralise plus ou moins bien.

Notons que l'erreur rétro-propagée est en réalité plus précisément la moyenne des erreurs d'un échantillon d'entrée appelé *batch*. La taille de ce batch aura une importance significative sur l'apprentissage du réseau.

On a donc vu l'apprentissage pour le réseau entier de chacun des poids de ses neurones. Dans une prochaine section, nous verrons plus formellement comment l'optimiseur *SGD* procède pour appliquer cela couche après couche.

Métrique

La métrique est une fonction qui est utilisée pour juger les performances d'un modèle. Une fonction métrique est similaire à une fonction objectif, à l'exception que les résultats de l'évaluation d'une métrique ne sont pas utilisées pour entraîner le modèle. Effectivement, après avoir entraîné le modèle, on veut mesurer si cela s'est bien passé. Dans ce cas on peut utiliser cette métrique, qui mesure le pourcentage de bon résultats que le modèle à fourni sur un échantillon d'images tests.

4.2.3 Le réseau

On vient de voir comment fonctionne un perceptron, mais qu'en est-il quand on en met plusieurs ensemble pour former un réseau ?

En fait, notre intérêt pour ce perceptron explose quand on voit les applications possibles lorsqu'on en assemble plusieurs de manière à former un réseau.

La première couche constitue la couche d'entrée, et la dernière la couche de sortie. Entre ces 2 couches se trouve un nombre plus ou moins grand de couches dites cachées (*hidden layers* en anglais). On voit sur la FIGURE 4.6 comment sont agencés les neurones en couches dans un réseau de neurones ne comprenant qu'une couche cachée, une couche d'entrée et une couche de sortie [Val17].

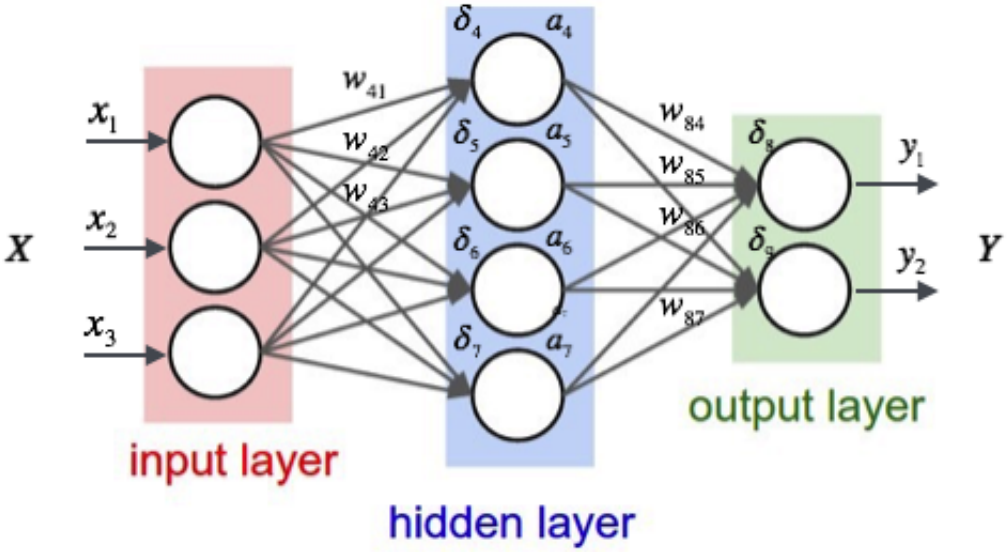


FIGURE 4.6 – Visualisation des couches d'un réseau de neurones ne possédant qu'une couche cachée

Mis sous forme de couche l'une à la suite de l'autre, il est possible de faire de la classification non-linéaire dès 2 couches, et d'approcher n'importe quelle fonction à partir de 3 couches (illustration sur la figure 4.7) [Ano19c].

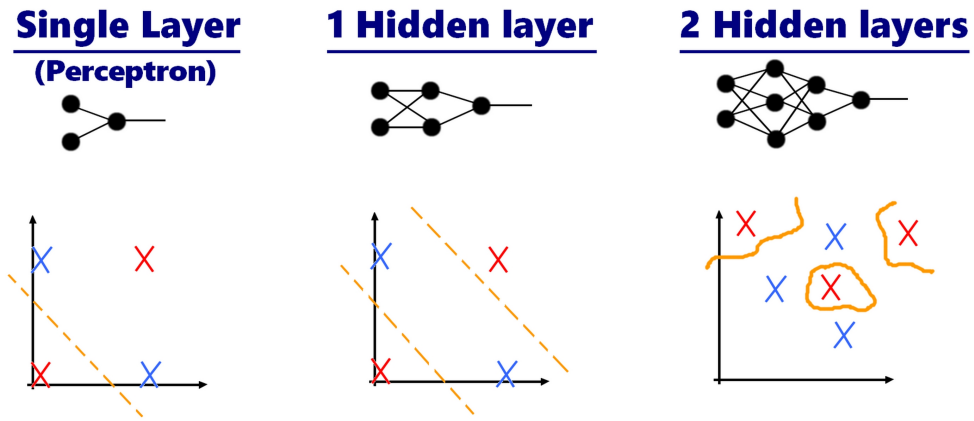


FIGURE 4.7 – Un neurone pour une classification linéaire, 1 couche cachée pour une classification non-linéaire, et 2 couches cachées pour approcher n'importe quelle fonction

4.2.4 Les méta-paramètres du réseau

On peut mettre plus de couches cachées, choisir le nombre de neurones que comporte chaque couche ainsi que leur fonction d'activation respective, choisir le taux d'apprentissage, la taille du batch, etc. Tout cela constitue les *hyper-paramètres* du réseau de neurones.

Un ***hyper-paramètre***, est un paramètre qui est assigné avant que le processus d'apprentissage commence. Par opposition aux valeurs des autres paramètres qui vont changer via l'entraînement.

Et “bien que la conception des couches d’entrée et de sortie d’un réseau de neurones soit souvent simple, il peut s’avérer être tout un art de concevoir les couches cachées.” [Nie17]

Plus le réseau aura de couche cachées, plus il sera *profond* (illustration sur la FIGURE 4.8).

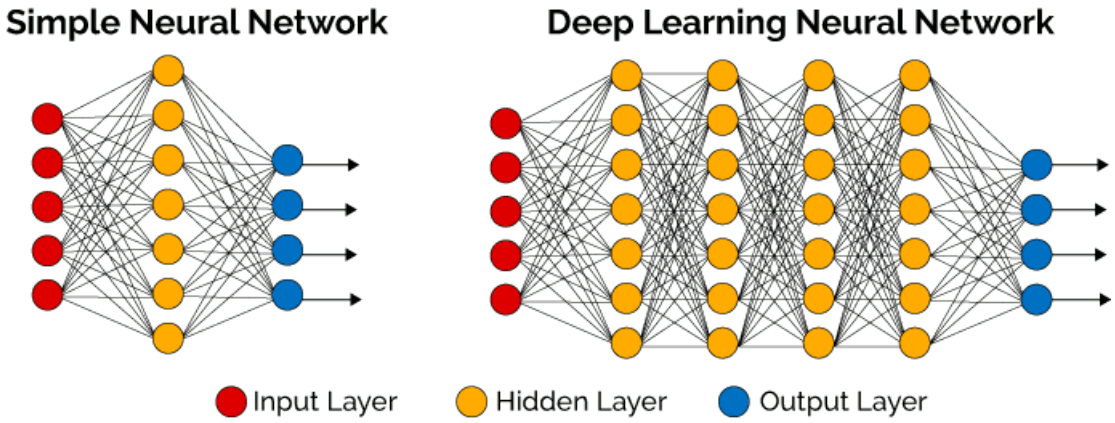


FIGURE 4.8 – Réseau de neurones **profond**, ou ***Deep*** *Neural network* en anglais

Plus le réseau sera profond, plus il y aura de poids et donc de paramètres à ajuster. Et ce qu’il y a de "magique", c’est que cette énorme liberté de paramètres à ajuster dans les réseaux profonds permettent de découvrir des motifs et des structures derrière d’immenses masses de données en entrées lors de l’entraînement.

Notons que plus les applications deviennent complexes et imposantes comparé à un simple perceptron, plus l’entraînement sera difficile et prendra en général plus de temps.

4.2.5 Optimiseur

Un point important est l’optimiseur. Celui-ci, avec la fonction objectif, utilise la rétro-propagation vue précédemment pour permettre l’apprentissage du réseau grâce aux données entrées durant l’entraînement, dont on connaît la sortie désirée (voir *apprentissage supervisé*).

L’optimiseur le plus connu, car il peut donner d’excellents résultats, est celui que nous utiliserons : *SGD*.

SGD, ou *Gradient de Descente Stochastique*, est une optimisation par approximation stochastique du gradient de descente. Il est appelé stochastique car des échantillons d’images sont sélectionnés de manière aléatoire (ou mélangés) pour former le batch utilisé lors de l’apprentissage. Contrairement au gradient de descente standard où l’on apprend par un seul batch de données, ou par des plus petits batchs dans l’ordre dans lequel les données d’entraînement sont données. [Wik19j] [Cod19]

Amenons maintenant un peu plus de rigueur mathématique. Dans les équations suivantes, nous utiliserons la représentation matricielle.

L’équation 4.7 représentent les updates des poids W.

$$\begin{aligned} V_{k+1} &= -\eta.\nabla L(W|X,y) \\ W_{k+1} &= W_k + V_{k+1} \end{aligned} \tag{4.7}$$

Où η est la taille du pas (ou *learning rate)*, et $\nabla L(W|X,y)$ est le gradient de la fonction objectif en fonction des entrées X et de la sortie y.

Le gradient de la fonction objectif peut être résolu en utilisant la rétro-propagation de la dérivée de la fonction objectif par les poids. Voici l'équation 4.8 pour la couche de sortie.

$$\begin{aligned} \nabla L(W_{ij}) &= \frac{dL(W_{ij})}{dW_{ij}} \\ &= \frac{dL(y|\hat{y})}{d\hat{y}}.\frac{d\hat{y}}{dZ}.\frac{dZ}{dW_{ij}} \\ &= \frac{1}{N}.\frac{(y-\hat{y})}{\hat{y}.(1-\hat{y})}.\hat{y}.(1-\hat{y}).X_{ij} \\ &= \frac{1}{N}.(y-\hat{y})X_{ij} \end{aligned} \tag{4.8}$$

Et pour les prochaines couches cachées l'équation 4.9.

$$\begin{aligned} \nabla L(W_{ij}) &= \frac{dL(W_{ij})}{dW_{ij}} \\ &= \frac{dL(y|\hat{y}_{ij})}{d\hat{y}_{ij}}.\frac{d\hat{y}_{ij}}{dZ_{ij}}.\frac{dZ_{ij}}{dW_{ij}} \\ &= \frac{1}{N}.\frac{(y-\hat{y})}{y.(1-\hat{y})}.\hat{y}.(1-\hat{y}).W_{ij}.\sigma(x)'.X_{ij} \\ &= \frac{1}{N}.(y-\hat{y}).W_{ij}.\sigma(x)'.X_{ij} \end{aligned} \tag{4.9}$$

où intervient $\sigma(x)$, la fonction d'activation.

C'est grâce à ces équations que l'on peut adapter les poids de l'entièreté du réseau à chaque entraînement de celui-ci. Sachez cependant qu'SGD n'est pas le seul optimiseur, et qu'il en existe d'autres plus ou moins performants tels que RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam, etc.

Dans le but d'avoir de meilleures performances, nous verrons plus loin que l'on peut complexifier cela avec de la *quantité de mouvement*, des extensions tel ***Layca*** ou encore des processus de *régularisation* comme ***Weight Decay***.

4.2.6 Quantité de mouvement, ou *Momentum*

Le problème d'SGD est qu'il y a beaucoup trop d'oscillations dans son optimisation. Voilà pourquoi le terme de quantité de mouvement a été ajouté. Le terme de quantité de mouvement, ou *Momentum* en anglais, est utilisé pour adoucir les oscillations et accélérer la convergence. On peut voir son effet dans l'équation 4.10 où en général $\gamma = 0.9$.

$$\begin{aligned} V_{k+1} &= \gamma V_k - \eta . \nabla L(W_k | X, y) \\ W_{k+1} &= W_k + V_{k+1} \end{aligned} \tag{4.10}$$

On l'utilisera avec le **Nesterov momentum**, dont le principe est de permettre de prédire la position d'un objet en temps T si l'on connaît la vitesse et la direction. Une visualisation du principe se trouve sur la figure 4.9.

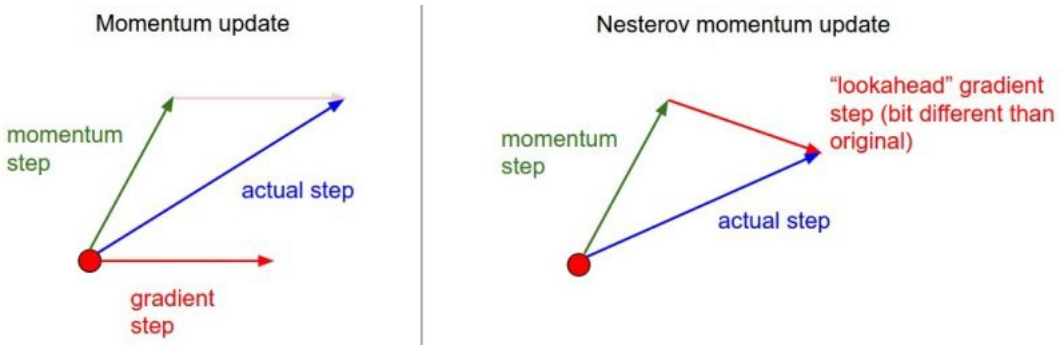


FIGURE 4.9 – Nesterov momentum représentation

Les nouvelles équations des poids du Nesterov momentum sont visibles en 4.11 [Bus17].

$$\begin{aligned} V_{k+1} &= \gamma V_k - \eta . \nabla L(W_k + \gamma V_k | X, y) \\ W_{k+1} &= W_k + V_{k+1} \end{aligned} \tag{4.11}$$

Le Nesterov momentum est un lointain cousin du momentum normal, mais il est très populaire car il permet d'atteindre de manière cohérente des minimas à une vitesse suffisamment rapide.

4.2.7 Régularisation

Les processus de régularisation permettent d'éviter le sur-apprentissage, ou *overfitting* en anglais. Pour le comprendre visuellement, regardons la figure 4.10 [Jai18]. On voit que le comportement des points oranges décrit une courbe que l'on essaie d'approcher au mieux via notre modèle en bleu (image 1 et 2). Lorsqu'il y a sur-apprentissage (image 3), on voit que le modèle colle trop bien aux données, mais non plus à l'allure de cette courbe des données. Notre modèle perd donc en performance.

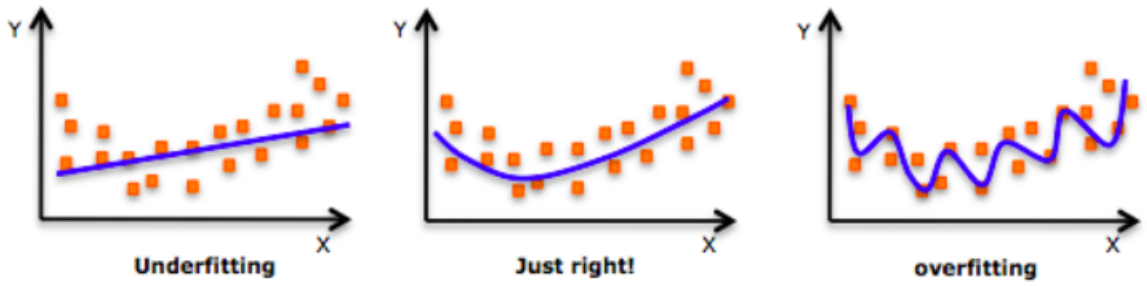


FIGURE 4.10 – Compréhension visuelle du sur-apprentissage

Ce processus est du au fait que l'on apprend trop des données d'entraînement. Les paramètres s'adaptent trop bien aux données, et du coup on perd la capacité à donner de bons résultats sur des données inconnues. C'est pourtant ce dernier cas qui est souvent recherché et qui est le but de l'apprentissage des réseaux de neurones, aussi appelé **généralisation**.

On peut voir les résultats d'erreur du modèle sur les données d'entraînement et sur les données de test quand il y a sur-apprentissage sur la figure 4.11. On voit que même si le modèle apprend mieux des données d'entraînement, il fait cependant plus d'erreurs sur des nouvelles données de tests.

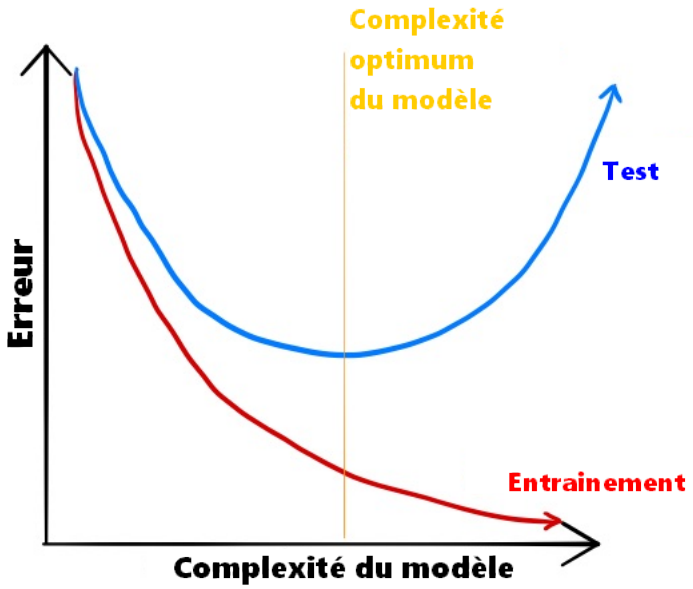


FIGURE 4.11 – Erreur sur les données d'entraînement et sur les données de tests

C'est afin de s'arrêter d'apprendre au bon moment que la technique du *early stopping* a été inventée.

Early stopping

Le early stopping est une méthode de régularisation pour prévenir le sur-apprentissage des données sur un processus itératif tel que la descente de gradient. [Wik19c]

Le principe est que, habituellement, le réseau va apprendre itérativement de manière intéressante jusqu'à un certain moment. A partir de ce moment, le modèle va continuer à apprendre des données mais en sur-apprennant. Il y a donc eu des méthodes mises au point afin d'arrêter l'apprentissage à ce moment où l'apprentissage ne devient plus bénéfique pour le modèle.

De manière générale, ces méthodes utilisent une base de données de validation pour savoir quand s'arrêter. On peut voir le moment d'early stopping sur la figure 4.12.

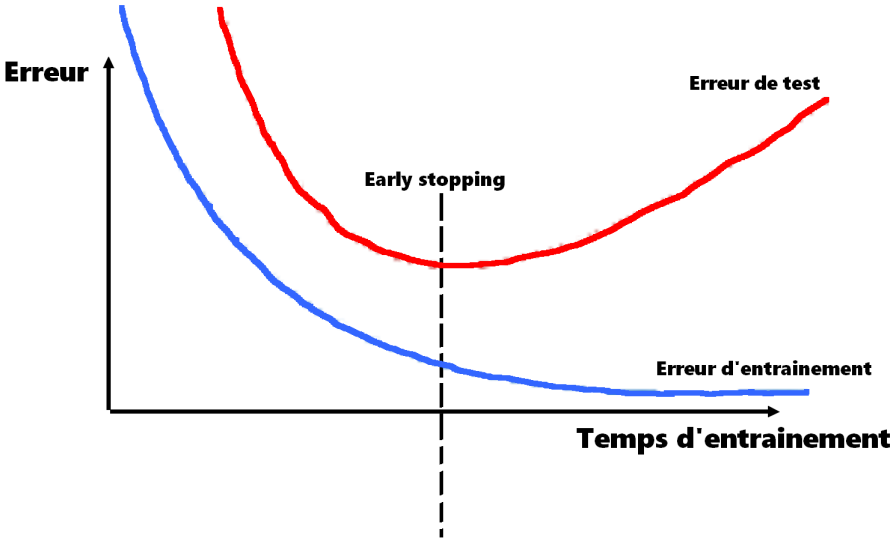


FIGURE 4.12 – Early Stopping point

Weight Decay and L2

La méthode de dégradation des pondérations, ou **Weight Decay** en anglais, est une autre technique de régularisation utilisée pour éviter le sur-apprentissage dans un réseau de neurones. Elle consiste à ajouter une pénalité à la fonction d'erreur (Cost function) qui dépend de la magnitude des poids qui relient les neurones entre eux. Il a été mathématiquement démontré [J+91] (Geman, Bienenstock & Doursat 1992 ;Krogh & Hertz 1992) qu'un réseau utilisant des connexions avec des pondérations de forte magnitude avait plus de peine pour généraliser. [Wik19k]

Pour palier à cela, on utilise donc la pénalité suivante à la fonction d'erreur pour une dégradation dite L^1 :

$$-\lambda \sum_i |w_i|$$

Ou la pénalité suivante pour une dégradation dite L^2 :

$$-\lambda \sum_i w_i^2$$

où w_i est le i^e poids dans le réseau et λ est un coefficient positif qui donne plus ou moins d'importance à la pénalité. Le paramètre λ est en général très petit (0.005, voir moins), et peut tendre vers zéro. [Nag17]

Les régularisations L^1 et L^2 ont une interprétation probabiliste légèrement différente. Dans tous les cas, cela sera équivalent à ajouter une priorité sur la distribution des poids de la matrice W ; une priorité Gaussienne dans le cas L^2, et une priorité Laplacienne pour le cas L^1. On transforme en fait le problème d'optimisation d'une estimation de maximum de vraisemblance (MLE, Maximum Likelihood estimation) en une estimation d'un maximum à posteriori (MAP) ; c'est à dire qu'on passe d'une inférence fréquentielle à une inférence Bayesienne.[Pal18] [Bab18] Plus d'informations sur cette interprétation peuvent être fournie dans une publication de Brian Keng [Ken16].

Voyons à présent l'impact de cette pénalité sur le changement des poids. Dans notre cas, nous utiliserons L^2 qui nous offre toujours une solution calculable car possédant une solution analytique. Donc pour la régularisation L^2, en ajoutant le terme de régularisation à la fonction objectif, on obtient l'équation 4.12.

$$\begin{aligned} V_{k+1} &= V_k - \eta.\nabla L(W_k + \gamma V_k|X, y) - \lambda||W_k|| \\ W_{k+1} &= W_k + V_{k+1} \end{aligned} \tag{4.12}$$

où $||.||$ est la norme L^2.

En fait, cette régularisation L^2 est presque équivalente à la régularisation dite Weight Decay, que l'on peut voir dans l'équation 4.13, pour l'utilisation d'SGD dans les conditions standards. (Attention, cela n'est pas spécialement vrai pour tous les algorithmes basés sur le gradient. Par exemple, sur les algorithmes de gradients adaptatifs comme Adam.)

$$\begin{aligned} V_{k+1} &= V_k - \eta.\nabla L(W_k + \gamma V_k|X, y) \\ W_{k+1} &= W_k + V_{k+1} - \lambda W_k \end{aligned} \tag{4.13}$$

On remarque en effet qu'il y a présence d'un facteur multiplicatif 2 entre le paramètre λ de Weight Decay et celui L^2. Cela vient du fait que la dérivée de $-\mathbf{0.5}\lambda W^2$ vaut $-\lambda W$, et donc pour la régularisation L^2 on a :

$$W_{k+1} = W_k - 2\lambda W_k - \eta.\nabla L(W + \gamma V_k|X, y)$$

.

Il faut donc simplement diviser λ par 2 quand on passe de WD à la régularisation L^2 pour avoir le même comportement.

Remarquons aussi intuitivement que l'utilisation de Weight Decay revient à diminuer les poids lors de chaque update d'une certaine fraction proportionnelle à l'amplitude de ceux-ci.

Voici finalement un exemple en 4.14 de notre cas avec la règle d'update des poids SGD de pas η, de Nesterov momentum $\gamma = 0.9$ et de paramètre Weight Decay $\lambda = 0.0005$.

$$\begin{aligned} V_{k+1} &= 0.9V_k - 0.0005\eta W_k - \eta\nabla L(W_k - 0.9V_k|X, y) \\ W_{k+1} &= W_k + V_{k+1} \end{aligned} \tag{4.14}$$

Dropout

La technique de régularisation dropout est couramment utilisée en apprentissage de réseaux profonds car elle produit de très bons résultats de prévention contre le sur-apprentissage. Elle a été approchée dans plusieurs articles [Gom+17][Sri+14].

Cette technique consiste à ignorer/retirer des neurones de manière aléatoire durant l'apprentissage. Le réseau va donc apprendre sans les liens qui le liaient à ces neurones.

Lors d'une itération d'apprentissage suivante, d'autres neurones seront sélectionnés pour être ignorés et l'apprentissage se fera sans eux. Pour aider à visualiser dropout, une illustration se trouve sur la figure 4.13. On peut y voir que dropout peut s'appliquer aux couches cachées, mais aussi aux couches d'entrées.

Un ensemble de modèles fonctionne habituellement mieux qu'un seul isolément car cet ensemble peut capturer plus de fluctuations aléatoires des données. De manière similaire, un réseau utilisant dropout fonctionne lui aussi mieux qu'un réseau de neurones normal.

Notons ici que la proportion de neurones qui doivent être retiré lors d'une itération de dropout est un hyper-paramètre et doit donc être choisi au mieux avant l'apprentissage du réseau.

Malgré le fait que cette technique est particulièrement efficace sur les réseaux profonds, par souci de simplification et afin de cerner au mieux le comportement des réseaux qui offrent une meilleure génération, nous n'utiliserons pas dropout dans nos réseaux.

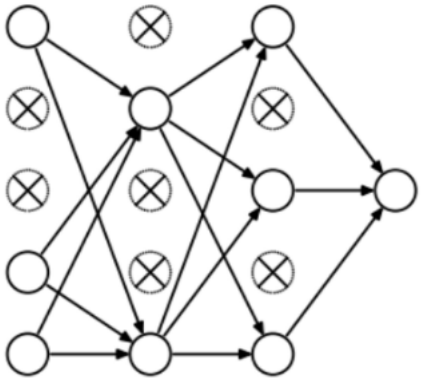


FIGURE 4.13 – Dropout

Batch Normalization (BN)

La *Batch Normalization (BN)* est une technique pour améliorer la vitesse, les performances et la stabilité des réseaux de neurones artificiels. [Wik19b][Col17][al15] [IS15] [Jan18] [Dom18]

Il est recommandé d'utiliser BN avant l'activation ReLu et les dropout (que nous n'utiliserons donc pas).

L'idée est que, au lieu de simplement normaliser les entrées du réseau, nous les normalisons aux niveaux des entrées des couches du réseau. C'est ce qu'on appelle la normalisation «batch» car lors de l'entraînement, nous normalisons les activations de la couche précédente, c'est-à-dire que nous appliquons une transformation qui maintient l'activation moyenne proche de 0 et l'écart type d'activation proche de 1. De cette façon, chaque couche ne doit pas faire beaucoup d'effort pour normaliser les sorties de la couche précédente, et peut se consacrer à son propre apprentissage.

On peut voir sur la figure 4.14 un exemple où les activations de la couche précédente sont normalisées grâce à une batch normalisation afin de se retrouver dans la zone intéressante de l'activation d'une sigmoide.

Data Augmentation

Si on a plus de données d'entraînement, notre réseau a moins de possibilités de sur-apprendre celles-ci. En utilisant la technique de régularisation d'augmentation des données pour l'entraînement, notre réseau gardera donc une meilleure capacité à généraliser.

Afin d'avoir plus de données, il est possible de pratiquer différentes sortes de transformations sur les images : les tourner d'un certain angle, les retourner comme un miroir, les réduire ou les agrandir, les décaler dans n'importe quelle direction,... et bien plus encore.

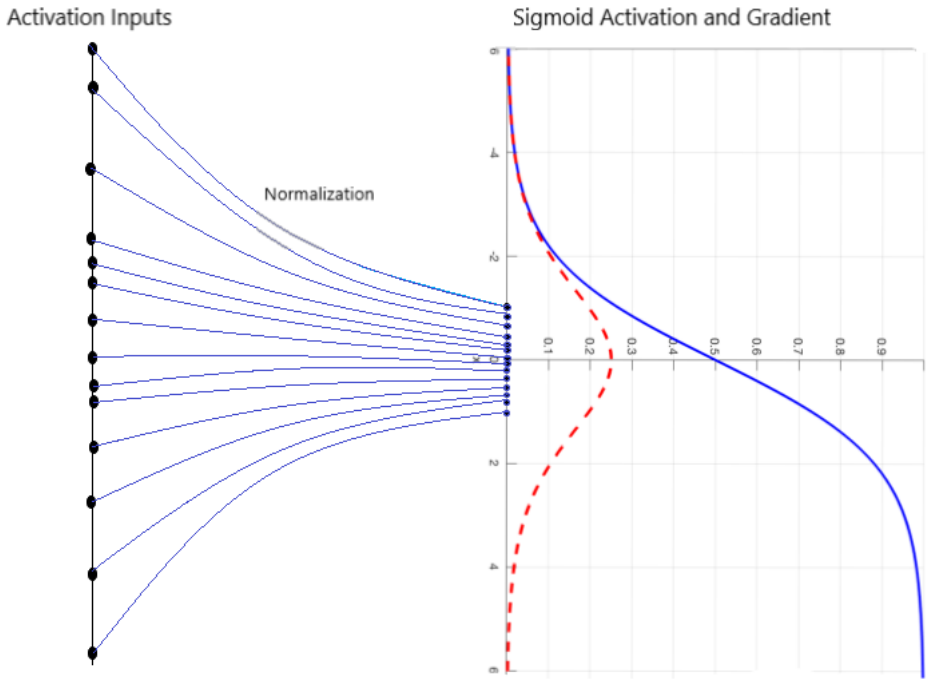


FIGURE 4.14 – Exemple d'utilisation de batch normalisation avec une fonction d'activation sigmoidale.

4.3 Convolutional Neural Networks (CNNs)

Afin de réaliser des tâches plus complexes, on peut complexifier le réseau de neurones en lui ajoutant des couches de convolution et de pooling. Ces 2 techniques ont prouvé leur efficacité dans de nombreux domaines de reconnaissance d'images et de classification. [Lab13][Ham17][Dan+11][Ale13b][DUJ12]

Dans les réseaux de neurones convolutifs (visible sur la figure 4.15), on retrouve la même base que pour les réseaux de neurones artificiels vue précédemment. La différence notable est due à la présence de couches convolutives et de max-pooling. Celles-ci ont permis grâce à la puissance des ordinateurs actuels de faire apprendre des réseaux très grands et très profonds. Effectivement, sans ce mécanisme convolutif il y aurait eu beaucoup trop de paramètres dans des architectures de telle ampleur.

On voit sur la figure 4.16 l'utilisation générale de convolution et de pooling.

Le concept de *convolution* consiste à passer un filtre sur une partie des données et à faire la somme des éléments de cette partie. Le concept de *pooling* fonctionne de la même manière, mais sur une plus grande zone. Alors que la convolution agit plus comme un filtre, le pooling va plutôt rechercher des comportements dans les entrées [Sah18].

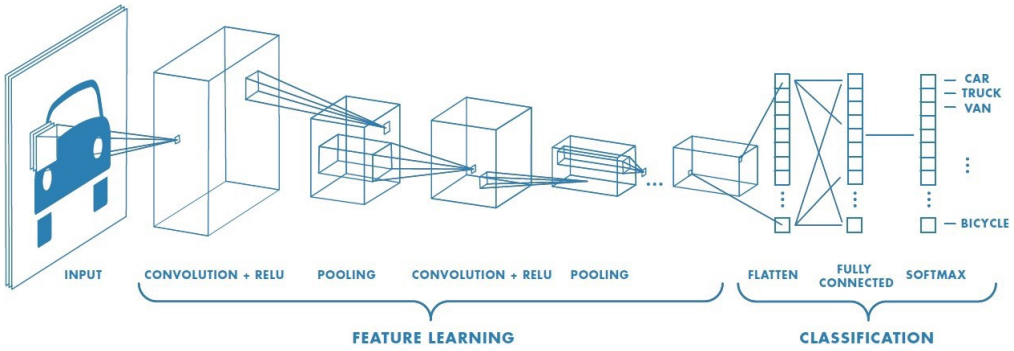


FIGURE 4.15 – Fonctionnement d'un CNN

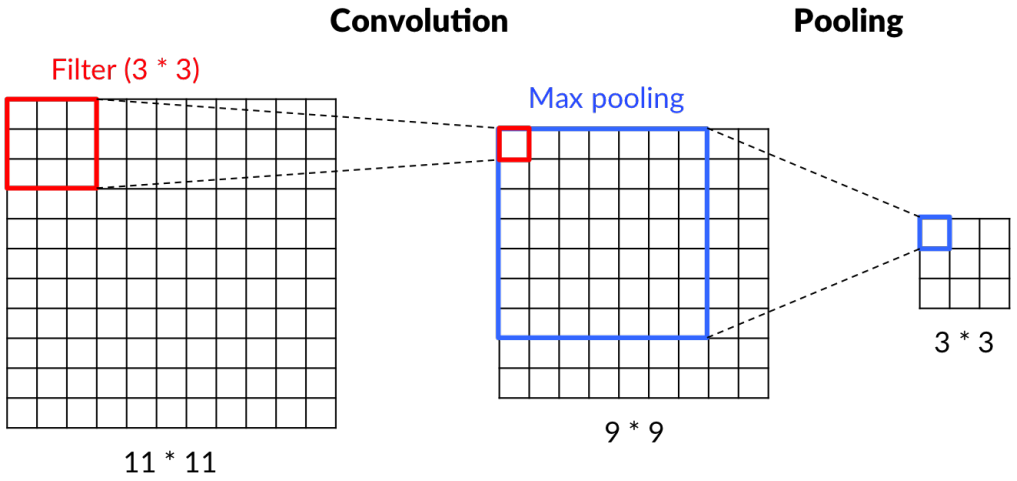


FIGURE 4.16 – Convolution and pooling

4.3.1 Convolution 2D

La convolution d'un signal à une dimension est appelée convolution 1D, ou tout simplement convolution. Si la convolution est appliquée en 2 dimensions (comme dans nos CNN), elle sera appelée convolution 2D. Ce concept peut être étendu à des convolutions multidimensionnelles pour des signaux à dimensions multiples. [HL18]

Dans le domaine digital, la convolution est calculée par la multiplication et l'accumulation des valeurs instantanées du chevauchement des échantillons de 2 signaux d'entrées, dont l'un a été retourné. La définition de la convolution 1D est aussi applicable pour la convolution 2D à l'exception que, dans cet dernier cas, l'une des entrées est retournée 2 fois.

On va en général utiliser la convolution sur une matrice 2D représentant l'image par une plus petite matrice appelée noyau 2D. Sur la figure 4.17, on peut voir un exemple de convolution 2D. On calcule l'image de sortie y produite par la convolution de l'image d'entrée x de taille 5x5 par le noyau h de taille 3x3.

C'est ce même principe qui prévaut dans les couches convolutionnelles 2D dans les réseaux

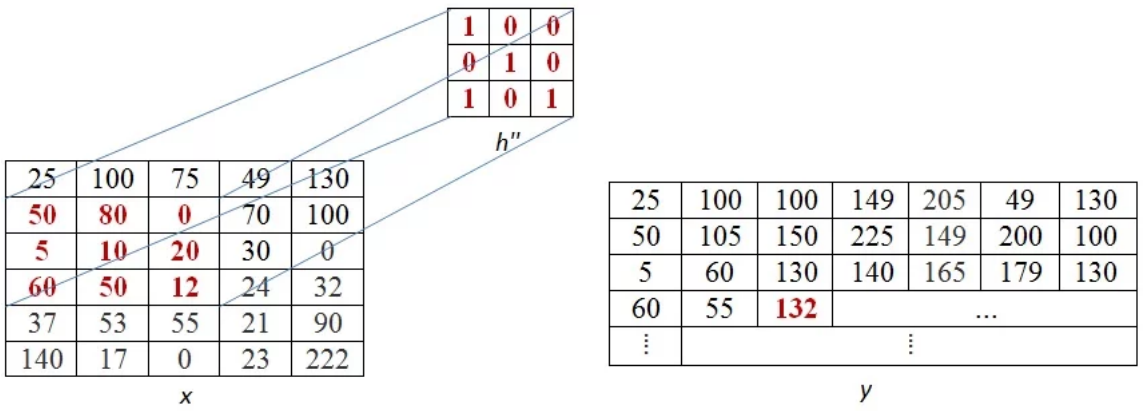


FIGURE 4.17 – Exemple d'une convolution 2D

de neurones, comme on peut le voir sur la figure 4.18 [Cav18].

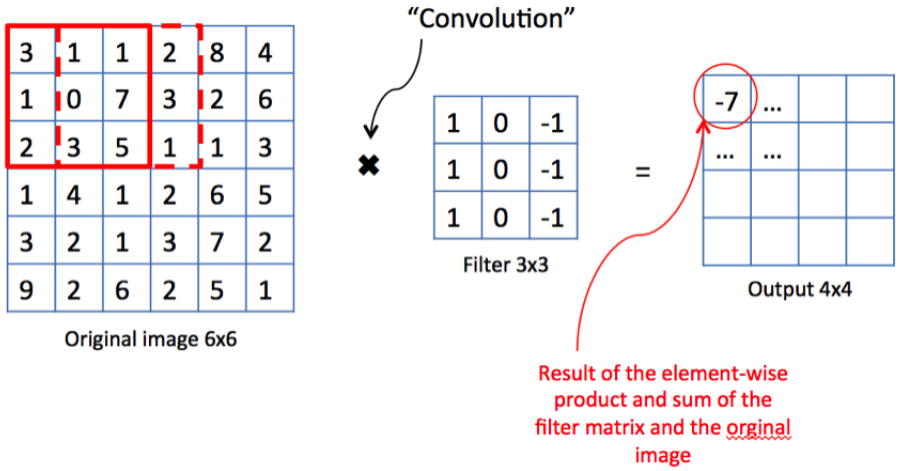


FIGURE 4.18 – Autre exemple de fonctionnement de convolution

La formulation mathématique d'une convolution 2-D est donnée par

$$y[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m,n].x[i-m, j-n] \tag{4.15}$$

Notons qu'une couche convolution est constituée de *plusieurs* convolutions avec des noyaux 2D de paramètres différents. On peut voir ces multiples convolutions différentes pour une même couche convolutive sur la figure 4.19.

On voit sur la figure 4.16 qu'il y a une réduction de dimension, les sorties ont des dimensions plus petites que les entrées. Cependant, on peut ne pas avoir de perte de dimension grâce au *padding*.

Le **padding** consiste à imaginer que l'image d'entrée est entourée par des valeurs, souvent mises à zéro, et à appliquer la convolution par dessus.

4.3.2 Pooling 2D

Remarquons sur cette même figure 4.19, la présence de couches de regroupement, ou *max-pooling* en anglais, plus petites que les couches de convolutions.

En fait, les réseaux de neurones convolutifs peuvent inclurent des couches de max-pooling locales ou globales. Les couches de pooling réduisent les dimensions des données en combinant les sorties des groupes de neurones d'une couche en un seul neurone dans la couche suivante.

Pour le pooling local, on fait un pooling en combinant des petits groupes de neurones, généralement 2 x 2. Le pooling global agit quant à lui sur tous les neurones de la couche convolutive précédente.

Le pooling peut s'effectuer pour un maximum ou une moyenne. Le pooling maximale utilise la valeur maximale de chaque groupe de neurones de la couche précédente. Le pooling moyen utilise la valeur moyenne de chaque groupe de neurones de la couche précédente.

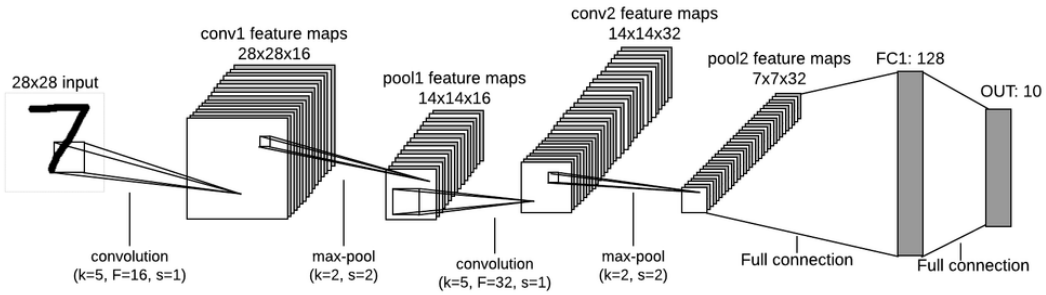


FIGURE 4.19 – Structure d'un réseau convolutif utilisant du pooling

4.3.3 Initialisation des poids des couches

La façon dont les poids sont initiés influence les conditions initiales. D'autres valeurs peuvent amener le modèle à des résultats complètement différents de performances. Il y a donc tout un répertoire de méthodes pour initialiser les poids des neurones artificiels.

Dans notre cas, les *couches denses* sont initialisées de manière **uniforme** aléatoire, avec des biais mis à zéro. Les *couches convolutives* sont initialisées de manière *glorot uniform*, avec des biais aussi mis à zéro.

Glorot uniform est une distribution uniforme dont les limites positives et négatives sont $\sqrt{\frac{6}{fan_{in}+fan_{out}}}$, où fan_{in} est le nombre de neurones d'entrées (input units) dans le tenseur des poids, et fan_{out} est le nombre de neurones de sorties (output units) dans le tenseur des poids.

Ce sont les paramètres de base d'initialisation des poids les plus couramment utilisés. Ils offrent une large gamme de poids choisis de manière arbitraire, sans tendance, et ils se prêtent parfaitement bien pour nos expériences.

5 État de l'art

"Si j'ai vu plus loin, c'est que parce que je j'étais sur les épaules de géants."

Isaac Newton

Sommaire

5.1 Introduction

Que nous apprend la littérature sur la question de généralisation ? Le présent chapitre constituera une brève réponse. Nous ferons ici un petit aperçu de la littérature et des travaux traitant de la

généralisation des réseaux profonds. Ces articles et autres publications nous ont tout d'abord servi d'inspiration et nous ont ensuite permis de calibrer nos recherches.

5.2 L'importance des données d'entraînement

Grâce aux avancées actuelles, il est aisé pour les réseaux de converger et d'atteindre 0% d'erreur sur les données d'entraînement. Cependant, cela ne garantit pas une bonne généralisation. En manipulant les données d'entraînement (ex : changement aléatoire des labels), [Zha+17] ont montré que des réseaux qui atteignent aisément 0% d'erreur montrent pourtant des performances médiocres en terme de généralisation. Il y aurait donc une différence notable entre la facilité d'optimisation des modèles, et les vraies causes de la généralisation.

Ceci étant, les réseaux profonds montrent généralement une aptitude surprenante à généraliser même si les modèles sur-paramétrisent, c.à.d. si les réseaux possèdent beaucoup plus de paramètres (poids) que de données d'entraînement. Ceci est surprenant car la *théorie de l'apprentissage statistique* attribue généralement les qualités de généralisation d'un modèle a sa plus faible complexité ([Zha+17]). Curieusement donc, un réseau profond, bien que riche et complexe, ne mémorisera pas simplement les données d'entraînement mais essayera de trouver une relation qui les caractérisent ([Kru+17] [Dev+17]). Plusieurs publications mettent en avant ce phénomène de différentes manières. En voici quelques uns :

Le document [KN19] fait la distinction entre les exemples faciles et difficiles à apprendre pour le réseau. Les auteurs ont quelques conclusions intéressantes :

1. les exemples difficiles contribuent plus à la généralisation que les exemples faciles
2. retirer beaucoup d'exemples faciles mène à une pauvre généralisation
3. les exemples mal classés dans le set de validation sont pour la plupart des exemples difficiles
4. tous les réseaux convolutifs, quelle que soit leur architecture, commencent à apprendre des mêmes exemples

Pour expliquer ces résultats, ils émettent l'hypothèse suivante : les réseaux convolutifs commencent par apprendre des motifs fréquents qui ne se contredisent pas à travers les classes. En effet par définition, un *pattern* fréquent dans les données apparaîtra souvent dans les *batchs* lors d'un *update*. Ces *patterns* définissent la direction principale du gradient, donnant la priorité au réseau de les apprendre en premier. [1]

[1] On pense que c'est pour cela que des batchs plus petits pour l'entraînement de *SGD* permettent de donner plus de poids aux exemples difficiles, et donc d'accentuer une meilleure généralisation en un nombre d'*updates* moins

L'excellent travail [Céc18] nous fait prendre conscience de l'importance des données utilisées, principalement de leur contenu. Il nous indique que les images avec une grande cohérence demandent moins de temps d'apprentissage par le réseau, mais ce dernier généralise mal sur des images inconnues. Au contraire, les images avec une faible cohérence, à priori plus faciles à classifier, nécessitent un plus long apprentissage, mais le réseau généralise alors très bien. On remarquera la similitude avec les résultats émis par [KN19].

[Tre+18] nous parle de l'entraînement de réseau de neurones profond pour la reconnaissance d'images artificielles. En ajoutant seulement du bruit artificiel dans l'apprentissage, les données d'entraînement sont moins biaisées par d'autres biais de bruits et ainsi les performances en terme de généralisation sont meilleures. Cette façon d'apprendre de manière synthétique peut dépasser l'apprentissage sur images réelles en terme de performances et offre donc des possibilités d'avenir intéressantes.

5.3 La dynamique des poids

5.3.1 La vitesse de rotation des couches

[CV18b] a montré que la rotation des couches durant l'entraînement — l'évolution de la *cosinus distance* entre le vecteur de poids initial et final d'une couche — est un bon indicateur de la généralisation. Leurs résultats suggèrent que si cette *cosinus distance* évolue à une vitesse similaire et atteint une valeur finale la plus haute possible (proche de 1) à travers les couches, alors le réseau aura tendance à bien généraliser. Ces résultats sont appuyés par un nouvel algorithme d'optimisation baptisé *layca* qui force les couches du réseau à évoluer uniformément. Ils démontrent également que *SGD* n'atteint généralement pas de telles caractéristiques mais que l'ajout du régularisateur *weight decay* engendre automatiquement les couches du réseau à évoluer plus uniformément jusqu'à une valeur de *cosinus distance* proche de 1, d'une manière similaire à *layca*. Ceci expliquerait la raison pour laquelle l'ajout de *weight decay* amènerait généralement à une meilleure généralisation. Cette étude nous a motivés à analyser de plus près ce lien entre l'évolution des poids dans le réseau et sa capacité à généraliser (**Chapitre 7**).

important. Effectivement, selon nous, la structure primaire d'un réseau qui généralise bien est principalement amenée par les exemples faciles. Alors que ce sont les exemples difficiles qui amènent le réseau à se perfectionner et corriger les anomalies, et donc à mieux cerner la généralisation voulue. Nous pensons qu'il y a une grande importance à accorder aux données, que leur diversité et leur hétérogénéité va apporter beaucoup en terme de généralisation lors l'entraînement ; tout comme le cerveau humain, il faut qu'on conceptualise l'objet, et ensuite on peut le retrouver dans des situations dont il est plus difficilement repérable. Cette approche reste possible à la suite de nos résultats.

5.3.2 L'élagage (*pruning*) des liens

La récente publication [Zho+19] montre qu'on peut avoir des performances supérieures en ré-entraînant un réseau dans lequel on a mis une bonne partie des poids à 0, seulement si on redémarre l'entraînement avec les mêmes poids initiaux. En fait, il y a des performances similaires quand on fait un *pruning* de 90 à 99.5 %, et des performances supérieures quand on fait un *pruning* entre 50 et 90 %. Il est donc montré que l'initialisation des poids n'est pas anodine. En particulier, l'important dans l'initialisation des poids est le signe, pas l'amplitude.

Ces expériences sur le *pruning* sont en accord avec les résultats de [AS19], qui démontrent qu'une représentation du modèle moins dense et plus élaguée peut nous apporter plus de robustesse aux bruits et interférences.

On citera aussi le point de vue des travaux sur le *targeted dropout* [Gom+17], qui est une méthode qui rend robuste le réseau au *pruning*, sans pour autant l'appliquer.

Ces études nous ont motivés à étudier la robustesse de nos modèles face au *pruning* (**Chapitre 7**).

5.4 La dynamique des activations

5.4.1 Les caractéristiques apprises par un réseau convolutif

[ZF14] ont montré que les caractéristiques (*features*) apprises par un réseau convolutif sont loin d'être aléatoires et inintelligibles. Au contraire, elles dévoilent beaucoup de propriétés désirables comme la composition, l'augmentation de la discernabilité des classes en aval des couches. En d'autres mots, elles collent avec l'intuition que l'on a de l'apprentissage des couches : les premières couches apprennent les caractéristiques très basiques de l'image comme les bords et les couches suivantes composent ces caractéristiques pour créer des objets plus abstraits, comme les oreilles d'un chat par exemple. On voit également que dans leur analyse des activations, il y a une importance à distinguer *plus* de *features complètement différentes* dans les premières couches. Ceci nous motivera à étudier la redondance des *features* apprises dans les activations des couches de nos modèles (**Chapitre 8**).

On citera aussi la publication [PY17] qui nous aide à mieux comprendre comment un réseau est activé. Effectivement, les auteurs ont développé une méthode de détection de saillance, qui permet de mettre précisément en avant et de sélectionner l'objet détecté d'une image. Certaines régions dans l'image apprises ne seront pas utiles, et d'autres régions le seront bien davantage pour la reconnaissance. Par exemple, pour reconnaître un chat, une petite partie de son oreille peut jouer le rôle déterminant dans la classification. Le masque est créé rétroactivement, et possède de

très bonnes performances comparé à ce qu'il se fait dans le domaine pour le moment. On retiendra que l'influence du placement des données sur l'image pour la classification est loin d'être anodine.

5.4.2 L'apprentissage vu par un neurone

Aussi complexe que peut être un réseau, il est finalement constitué de neurones. L'algorithme d'apprentissage *SGD* n'est pas basé sur une répétition d'un algorithme d'entraînement au niveau neuronal, mais bien d'en mécanisme global d'optimisation du réseau, qui est traité comme une boite noire. Mais d'après les recherches menées par [CV18a], les neurones cachés auraient tendance à se comporter comme des classifieurs binaires, divisant les entrées en 2 catégories de tailles quasi égales, même quand il n'y a pas de fonction d'activation utilisée. Cela va nous aider à motiver la réalisation des expériences des activations sur le *clustering* (**Chapitre 8**), en analysant mieux la dynamique encore trop peu explorée d'apprentissage du réseau au niveau même du neurone.

5.5 Les hyperparamètres influençant la généralisation

5.5.1 Profondeur

Les recherches menées par [Zha+17] nous informe entre autres qu'un réseau de profondeur 2 (*hidden layer + output layer*) peut approximer n'importe quelle fonction du moment que le nombre de ses paramètres (poids) dépasse la taille du *training set*. Cela met en avant la capacité de mémorisation du réseau.

5.5.2 Fonction d'activation

[ALL18] nous informe également qu'un réseau de neurones avec des activations $ReLU$ peut avoir les mêmes performances qu'un plus petit réseau avec des activations continues $\mathcal{C}^1$. Selon [Ale13a], la fonction d'activation $ReLU$ accélère également l'apprentissage par *SGD*. Pour cette raison, nous l'utilisons exclusivement dans ce travail.

5.5.3 *Learning rate*

Une multitude de sources dont notamment [Lau17] ont montré que décroître le *learning rate* pendant l'apprentissage amenait à une meilleure généralisation. C'est ce que nous avons fait aussi dans notre étude.

5.5.4 *Batch size*

Selon [Kes+16], la taille du *batch size* aurait un impact sur la capacité du réseau à généraliser. Plus la taille du *batch size* augmente, plus le réseau a de chances de se retrouver dans un minimum local étroit, ce qui a tendance à diminuer ses performances. À titre d'information, [EID17] démontre que le problème n'est pas directement lié au *batch size* mais au nombre d'*updates* de l'apprentissage.

Si on utilise une multitude de petits batchs, on peut quand même, après un certain nombre d'itération, mieux cerner les biais voulus [Dom18]. C'est d'ailleurs souvent cette dernière option qui est choisie car plus efficace après un nombre suffisent d'itération.

Pourtant les auteurs ont mis au point un algorithme d'optimisation permettant de travailler avec des *batch size* de plus grande taille tout en préservant les performances des modèles. Nous n'utilisons cependant pas cet algorithme dans notre travail, préférant travailler avec un *batch size* relativement petit (128).

5.6 Conclusion

Voici les principaux travaux qui nous intéresseront et qui nous stimuleront dans notre recherche. Nous vous invitons vivement d'y jeter un oeil afin de mieux comprendre notre démarche, et de voir les liens évidents tissés avec ces travaux de par la position de notre travail.

6 Méthodes

Le but de ce chapitre est de donner :

- une vue d'ensemble sur les méthodes expérimentales utilisées
- les moyens au lecteur pour qu'il puisse répliquer les résultats de nos expériences

Sommaire

6.1 Méthodologie

Afin de cerner les caractéristiques qui amènent un réseau à bien généraliser, notre analyse se fonde sur la comparaison de modèles qui ont une bonne et une mauvaise capacité à généraliser. Puisque notre travail s'inscrit en grande partie dans la lignée des recherches publiées par [CV18b], nous avons décidé d'utiliser les trois optimiseurs *SGD*, *weight decay* et *layca* afin de créer nos modèles.

6.1.1 Modèles entraînés

Nos modèles sont au nombre de neuf, chacun ayant subit un entraînement différent. Ils sont repris dans le tableau 6.1.

Nom du modèle	**Optimiseur**	**Validation accuracy**	**Test accuracy**
vgg_sgd_0	*SGD*	0.65	0.64
vgg_sgd_1	*SGD*	0.75	0.75
vgg_sgd_2	*SGD*	0.84	0.84
vgg_wdecay_0	*SGD + weight decay*	0.68	0.66
vgg_wdecay_1	*SGD + weight decay*	0.74	0.74
vgg_wdecay_2	*SGD + weight decay*	0.87	0.86
vgg_layca_0	*SGD + layca*	0.65	0.65
vgg_layca_1	*SGD + layca*	0.76	0.74
vgg_layca_2	*SGD + layca*	0.85	0.85

TABLE 6.1 – Performances des différents modèles utilisés. Les scores ont été arrondis à la deuxième décimale.

Nous utilisons donc trois types d'optimiseurs différents. Chacun de ses optimiseurs amènera les modèles à atteindre des performances en terme de validation de l'ordre de 65%, 75% et 85% [1]. Notons que les poids initiaux utilisés seront les mêmes pour les neuf modèles.

L'intérêt de cette méthodologie est que cela permet de faire une analyse tant horizontale que verticale des modèles. On pourra par exemple se poser les questions suivantes :

- Sur base d'un critère **A**, qu'est-ce qui différencie les optimiseurs **X** et **Y** qui ont une *test accuracy* similaire ?
- Sur base d'un critère **B**, qu'est-ce qui différencie l'optimiseur **X** pour différentes valeurs de la *test accuracy* ?

[1] Si nous avions ajusté les hyperparamètres de nos modèles pour qu'ils atteignent ces scores sur le *test set*, alors ce score aurait représenté une légère surestimation de la capacité du modèle à généraliser. Cela reste cependant un détail dans ce cas-ci car les scores de validation et de test sont assez proches.

6.1.2 Architecture du réseau

Nos réseaux convolutifs sont calqués sur l'architecture **VGG-16** visible sur la FIGURE 6.1, que nous avons adapté pour notre recherche :

1. en changeant la taille de la couche de sortie à 10 (puisque nous avons 10 classes)
2. en rajoutant des couches de *batch normalization* avant chaque activation afin d'accélérer l'apprentissage

L'architecture complète est disponible sur notre *github* dans le fichier MODELS.PY.

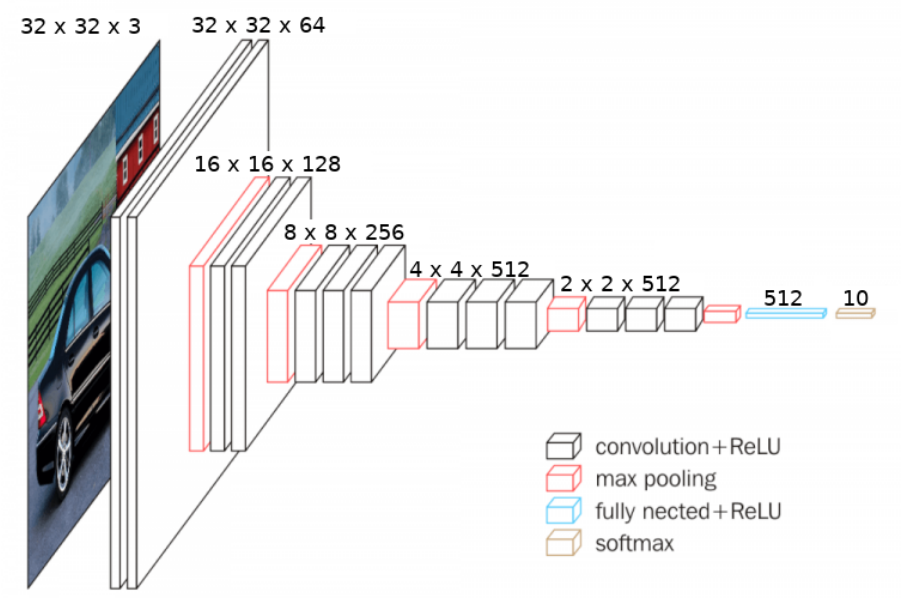


FIGURE 6.1 – Architecture VGG-16 utilisée

On peut alors se faire une idée de la disposition des couches de notre réseau sur la FIGURE 6.2.

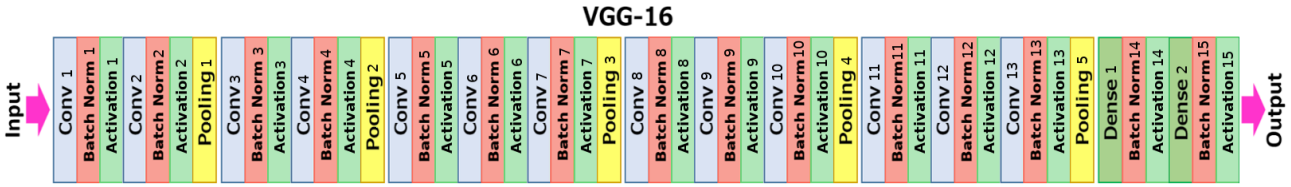


FIGURE 6.2 – Disposition des couches de nos réseaux

Le tableau 6.2 contient de plus amples informations sur l'architecture comme le nombre de neurones par couche, ainsi que le nombre de paramètres ajustables.

Couche	# Neurones	# Paramètres	Couche	# Paramètres	Couche	# Paramètres
Conv 1	65536	1792	BN 1	128	Acti 1	0
Conv 2	65536	36928	BN 2	128	Acti 2	0
			Pooling 1	0		
Conv 3	32768	73856	BN 3	256	Acti 3	0
Conv 4	32768	147584	BN 4	256	Acti 4	0
			Pooling 2	0		
Conv 5	16384	295168	BN 5	512	Acti 5	0
Conv 6	16384	590080	BN 6	512	Acti 6	0
Conv 7	16384	590080	BN 7	512	Acti 7	0
			Pooling 3	0		
Conv 8	8192	1180160	BN 8	1024	Acti 8	0
Conv 9	8192	2359808	BN 9	1024	Acti 9	0
Conv 10	8192	2359808	BN 10	1024	Acti 10	0
			Pooling 4	0		
Conv 11	2048	2359808	BN 11	1024	Acti 11	0
Conv 12	2048	2359808	BN 12	1024	Acti 12	0
Conv 13	2048	2359808	BN 13	1024	Acti 13	0
			Pooling 5	0		
Dense 1	512	262656	BN 14	1024	Acti 14	0
Dense 2	10	5130	BN 15	20	Acti 15	0

TABLE 6.2 – Information relative à chaque couche de notre architecture

6.1.3 Analyse des résultats

Les expériences ont toutes été menées sur les neufs modèles. Pour chaque *epoch* de l'apprentissage, nous avons sauvegardé un *snapshot* du modèle. Ceci nous a permis d'observer la dynamique d'apprentissage de plus près, en plus de la comparaison des modèles finis entre eux.

La majorité des résultats obtenus ont été étudiés graphiquement. Pour chaque résultat obtenu, nous avions émis des hypothèses basées sur notre connaissance du sujet. Cela permettait de les affirmer ou les infirmer. Le tout étant exécuté en veillant à garder un regard critique.

6.2 Outils

6.2.1 Base de données

Pour entraîner/tester les réseaux convolutifs que nous utilisons pour nos expériences, nous avons utilisé la base de données **CIFAR-10**. Elle contient des images d'objets et d'animaux courants telles que *voiture, avion, cheval, chien, chat, ...*

La TABLE 6.3 détaille les caractéristiques de la base de données tandis que la FIGURE 6.3 montre quelques unes de ces images.

Motivation : CIFAR-10 est largement utilisée pour l'entraînement d'algorithmes en machine learning et computer vision. Cela nous permet d'avoir suffisamment de documentation et de travaux portant sur le même matériel et donc de pouvoir comparer nos résultats plus facilement

Caractéristiques	Valeur
nombre de classes	10
nombre d'images (train/test)	60000 (50000/10000)
résolution	32x32x3 (RGB)

TABLE 6.3 – Descriptif de la base de donnée *CIFAR-10*

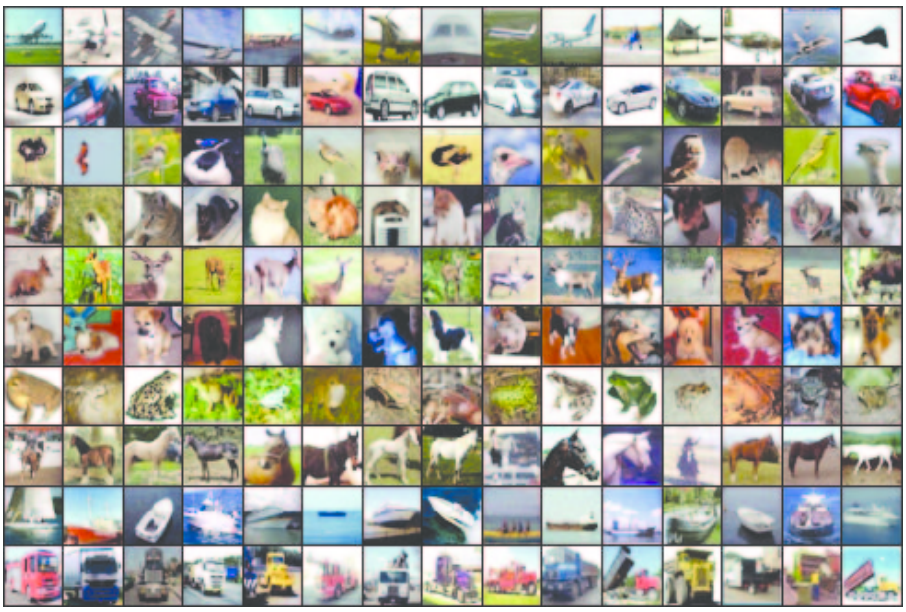


FIGURE 6.3 – Quelques exemples d'images provenant de la base de données CIFAR-10. Les 10 classes représentées dans les images sont : des camions, des bateaux, des chevaux, des grenouilles, des chiens, des chats, des avions, des voitures, des oiseaux et des cerfs

les leurs.

Notre utilisation : Notez que nous avons d'avantage départagé le set d'entraînement (*training set*) en deux pour avoir un set de validation de 10000 images. En somme, nous avons 40000, 10000 et 10000 images respectivement pour le *training*, *validation* et le *test set*. Ce set de validation sert à tester la précision de nos modèles en jouant sur leurs hyper-paramètres.

6.2.2 Langage de programmation, librairies et matériel

Le langage de programmation utilisé est **python**. Nous utilisons la librairie **Keras** pour construire et entraîner nos modèles. Pour de plus amples informations, la documentation de la librairie est disponible à l'adresse `https://keras.io/`.

Pour ce qui est du matériel, nous avons reçu l'accès au serveur *cassiopée* de l'*UCL*. Ce serveur possède quatre cartes graphiques (GPU) *Nvidia GTX 1080* qui peuvent être utilisées par **Keras** afin d'accélérer l'apprentissage des modèles.

6.3 Notre recette d'apprentissage

6.3.1 Initialisation

1. Importer la base de données **CIFAR-10** de telle manière à avoir respectivement 50000 et 10000 images pour le *training set* et le *test set*.

2. Charger les neufs modèles et leur assigner une même distribution de poids initial.

6.3.2 Entraînement

1. Choisir une valeur pour le *learning rate* pour chaque modèle.

2. Entraîner chaque modèle jusqu'à ce que le *training accuracy* atteigne un minimun de 99%. À chaque *epoch*, sauvegarder les poids du réseau, sa *cosinus distance* et autres données utiles.

3. Si la *validation accuracy* à la fin de l'entraînement n'est pas celle désirée, recommencer à partir de l'étape 1.

6.3.3 Commentaires

Évolution du *learning rate*

Afin d'arriver à de meilleurs performances en terme de généralisation, le *learning rate* de nos modèles évolue au fil de l'entraînement ([Lau17]). Nous utilisons la méthode *ReduceLROnPlateau* de Keras de telle manière à diviser le learning rate par 5 si la *validation accuracy* n'augmente pas significativement (d'un ordre de 1e-4) pendant 2 *epochs*.

Arrêt de l'entraînement

Nous avons observé que lorsqu'un modèle atteint 99% de *training accuracy*, sa *validation accuracy* n'augmente plus significativement ; le réseau *overfit* les données d'entraînements à ce stade. L'entraînement à tendance à stagner après cette valeur et prend parfois un nombre important d'*epochs* avant d'arriver à 100%. Dès lors, nous avons décidé d'arrêter l'entraînement à 99%, jugant ces dernières *epochs* sans importance dans notre analyse.

Choix du *batch size*

La taille du *batch size* a été maintenue à une valeur constante de 128 images. Il s'agit d'une valeur souvent utilisée en pratique. Cette valeur nous a permis d'obtenir de meilleures performances de généralisation en un nombre réduit d'*epochs* ([Kes+16] [EID17]).

6.3.4 Évolution des modèles avec les *epochs*

A présent, passons un coup de loupe sur les figures suivantes, qui montrent l'évolution au cours des *epochs* de la *validation accuracy* et de la *cosinus distance* pour chaque modèle.

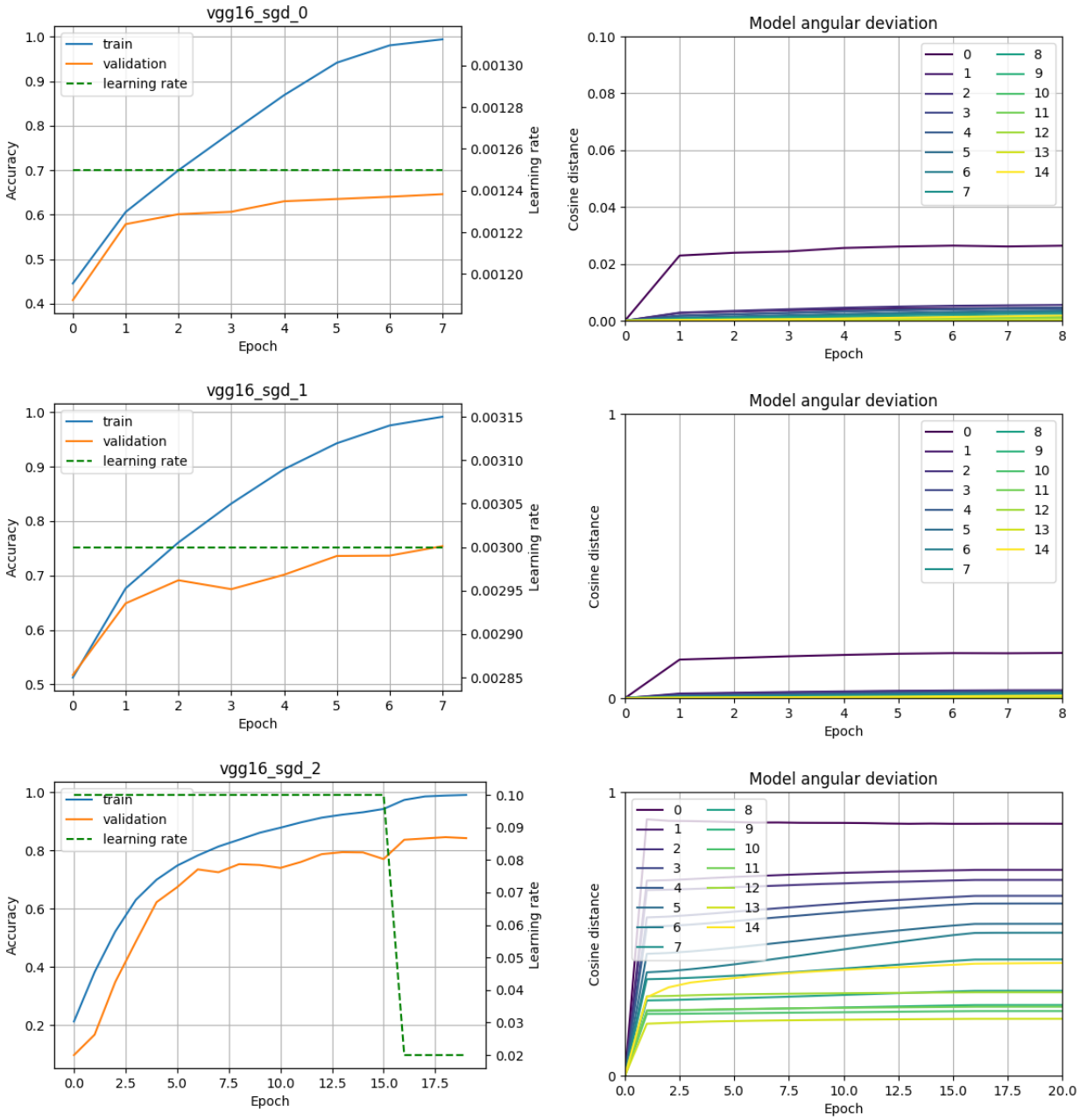


FIGURE 6.4 – **Évolution des caractéristiques des modèles entraînés par *SGD* au fil des *epochs***. Les images de gauche montrent l'évolution de la *training* (**bleu**) et *validation accuracy* (**orange**) ainsi que du *learning rate* (**vert**) du moins bon au meilleur modèle. Les images de droite montrent l'évolution de la *cosinus distance* des couches au fil des *epochs*. Les numéros des couches correspondent aux numéros de la FIGURE 6.2 moins un.

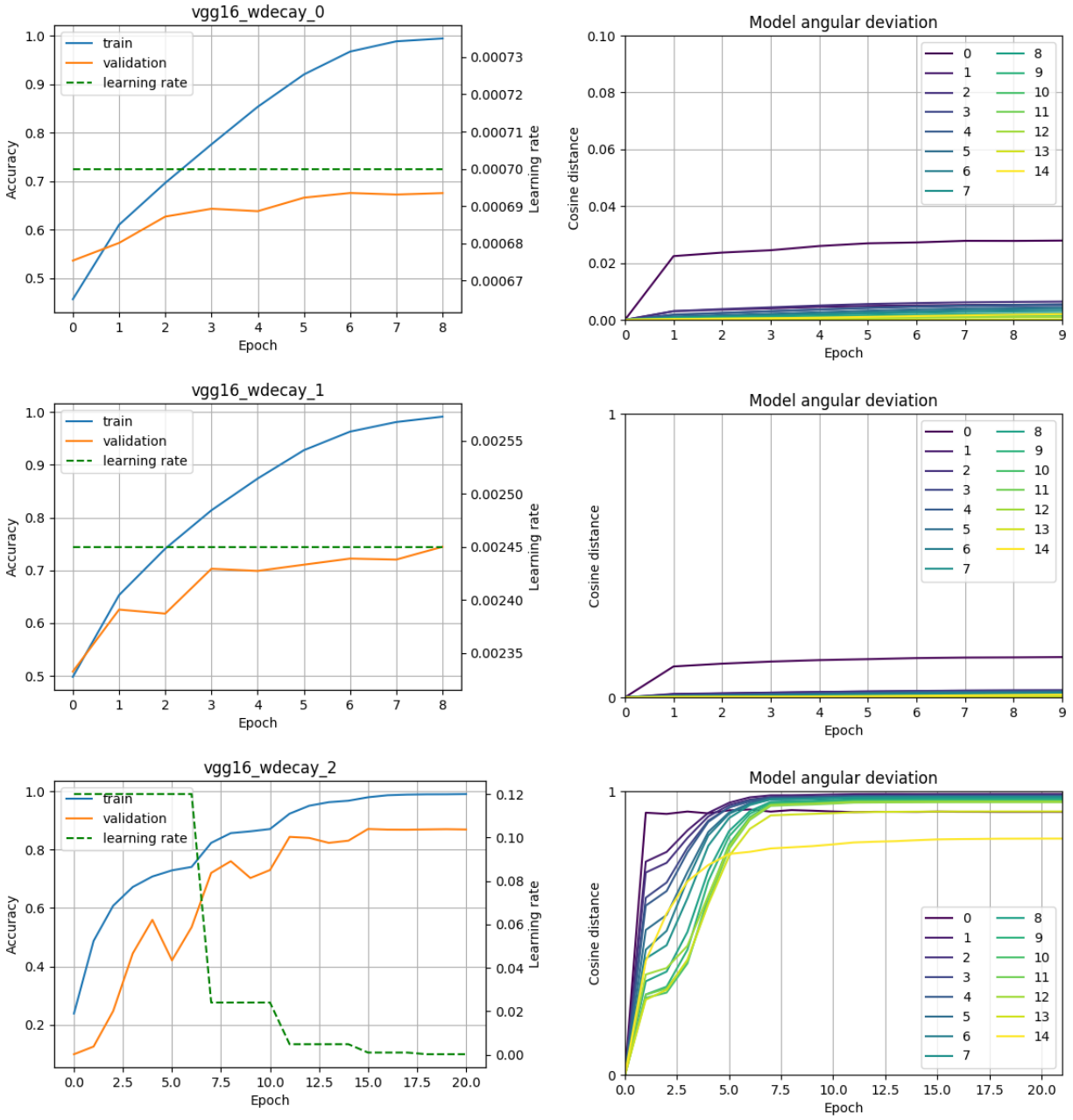


FIGURE 6.5 – **Évolution des caractéristiques des modèles entraînés par *weight decay* au fil des *epochs***. Les images de gauche montrent l'évolution de la *training* (**bleu**) et *validation accuracy* (**orange**) ainsi que du *learning rate* (**vert**) du moins bon au meilleur modèle. Les images de droite montrent l'évolution de la *cosinus distance* des couches au fil des *epochs*. Les numéros des couches correspondent aux numéros de la FIGURE 6.2 moins un.

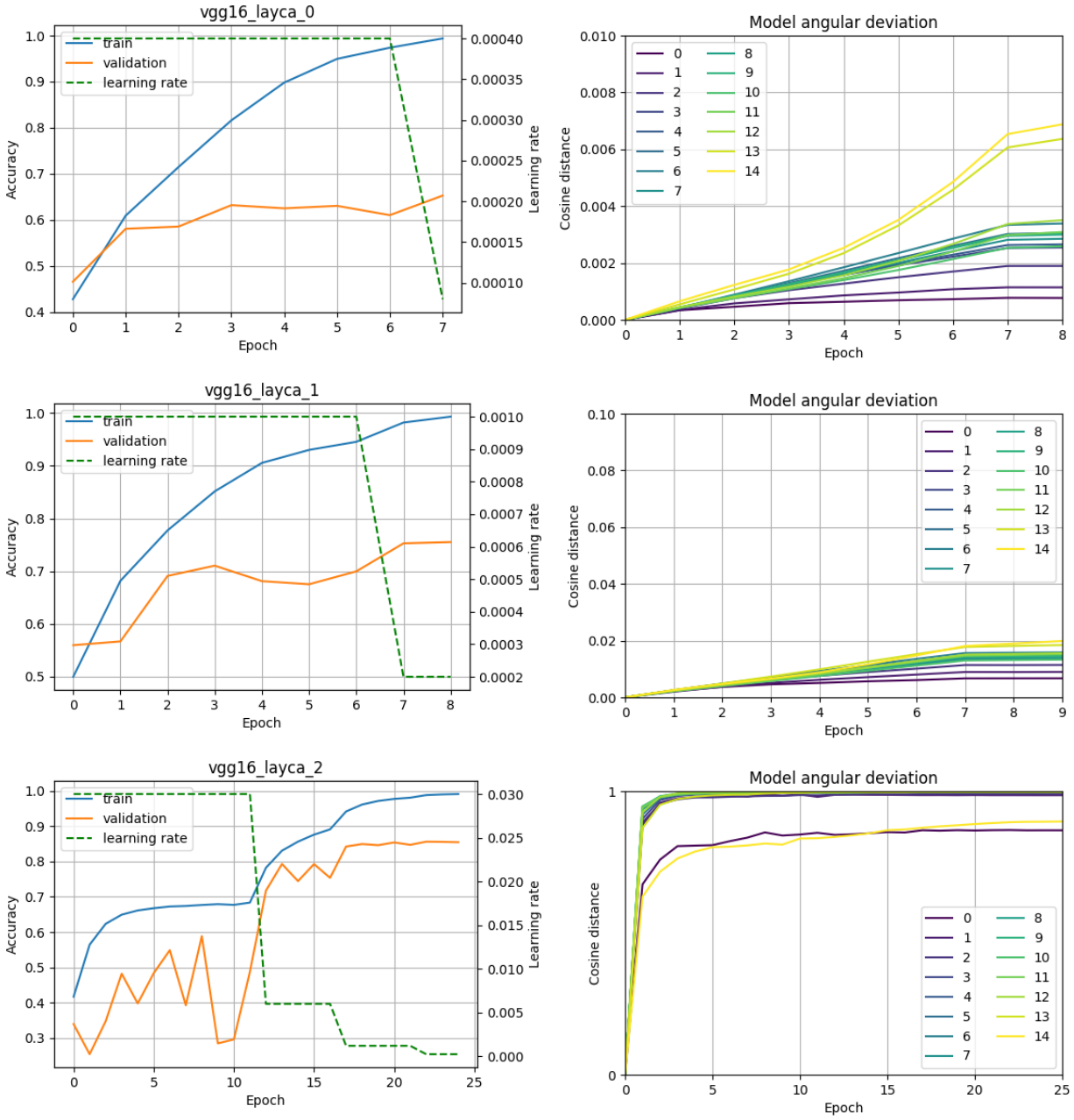


FIGURE 6.6 – **Évolution des caractéristiques des modèles entraînés par *layca* au fil des *epochs***. Les images de gauche montrent l'évolution de la *training* (**bleu**) et *validation accuracy* (**orange**) ainsi que du *learning rate* (**vert**) du moins bon au meilleur modèle. Les images de droite montrent l'évolution de la *cosinus distance* des couches au fil des *epochs*. Les numéros des couches correspondent aux numéros de la FIGURE 6.2 moins un.

Commentaires

Nous avons remarqué que pendant son entraînement, le meilleur modèle *SGD* peinait à dépasser les 84% de *validation accuracy*, alors que les deux autres optimiseurs pouvaient être ajustés de telle manière à atteindre 87%.

Notons deux dernières choses. La première que est que nous avons réussi à entraîner un

modèle *sgd* qui possède 86.7% de *validation accuracy* lorsque nous baissions le *batch size* à 64. La deuxième est que nous avons observé des courbes de cosinus distance qui n'allait pas exactement jusqu'à 1 pour un bon modèle entraîné par *layca*.

7

Une étude des poids

"*To ask the right question is harder than to answer it.*"

Georg Cantor

Sommaire

7.1 Introduction

Ce chapitre est le premier des deux chapitres qui portent sur les analyses que nous avons menées lors de cette année de recherches. Dans ce chapitre, nous étudierons nos réseaux convolutifs du point de vue de leur **poids**. Dans le chapitre suivant, nous étudierons ces réseaux via les **activations** neuronales.

7.1.1 Mesure utilisée

La cosinus distance

La cosinus distance ϕ permet de calculer la distance angulaire entre deux vecteurs. Sa formule mathématique est donnée par

$$\phi(\mathbf{a}, \mathbf{b}) = 1 - \cos(\angle \mathbf{ab})$$

où $\angle \mathbf{ab}$ est l'angle entre le vecteur $\mathbf{a}$ et $\mathbf{b}$. La mesure d'angle est ambiguë au delà de trois dimensions ; la formule s'écrit plus généralement

$$\phi(\mathbf{a}, \mathbf{b}) = 1 - \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}|| \cdot ||\mathbf{b}||}$$

où $(\cdot)$ représente le produit scalaire et $|| \cdot ||$ la norme euclidienne du vecteur.

La mesure de cosinus distance a été utilisé dans [CV18b] pour quantifier la distance angulaire entre les vecteurs de poids initiaux et finaux des couches du réseau. Nous utilisons également cette métrique de façon à pouvoir comparer les courbes de cosinus distance de nos modèles entre elles et avec celles que l'on trouve dans [CV18b].

En deux dimensions, la cosinus distance vaut 1 lorsque les deux vecteurs sont orthogonaux. Il faut cependant noter que les vecteurs des entrées des neurones de nos réseaux résident en de très grandes dimensions. L'intuition de cette distance tombe alors rapidement.

7.1.2 Motivation

Les recherches menées par [CV18b] suggèrent que l'évolution des poids d'un réseau est étroitement liée à sa capacité à généraliser ; si ses poids *changent* suffisamment lors de l'entraînement, alors le réseau aura tendance à bien généraliser. Pour être plus précis, on dira que les poids d'une couche du réseau auront beaucoup *changé* s'ils sont distants de leur état initial en terme de *cosinus distance* (c.f. plus haut). Lorsque cette cosinus distance évolue à cadence similaire sur l'ensemble des couches du réseau et que sa valeur est importante en fin d'entraînement (proche de 1), un réseau aurait tendance à bien généraliser. Comme nous pouvons le constater dans les figures du chapitre précédent, cette tendance est moins prononcée pour l'optimiseur *sgd* que pour *weight decay* et *layca*, alors qu'il atteint aussi une *test accuracy* similaire (avoisinant les 85%).

Ceci étant, si l'on effectue une comparaison des 3 modèles SGD, nous observons que la cosinus distance est plus grande sur l'ensemble des couches pour le modèle 85%. Ceci indique que cette mesure vaut la peine d'être investiguée, même si sa relation avec la capacité de généralisation est moins directe qu'on le pensait initialement.

Dans ce chapitre, nous investiguons donc ce phénomène en étudiant de plus la dynamique des poids des modèles entrainés par *sgd*, *weight decay* et *layca*, afin d'y trouver des indicateurs de bonne généralisation.

L'utilisation de ces trois optimiseurs est cruciale à notre analyse car nous pensons que le comportement d'apprentissage de *weight decay* et *layca* est similaire mais différent de *sgd*.

Cette première analyse s'appuiera surtout sur les articles [CV18b] [Zho+19] [AS19].

7.1.3 Structure du chapitre

Nous garderons en tête une question centrale pour ce chapitre :

Quelles sont, au niveau de la dynamique des poids, les mécanismes d'apprentissage qui favorisent une bonne généralisation et/ou l'émergence d'une grande cosinus distance pour l'ensemble des couches ?

Afin d'apporter des réponses à cette question fondamentale, nous formulons notre analyse sous la forme de deux questions sous-jacentes :

1. **Comment les poids du réseau évoluent-ils pendant l'entraînement ?** Cette section a pour but de mieux comprendre la façon dont chaque optimiseur joue sur ses poids au cours de l'entraînement.
2. **Que se passe-t-il si l'on ampute le réseau de certains de ces poids ?** Finalement, on regardera la façon dont nos modèles réagissent face à l'élagage (*pruning*) de certains de leurs poids.

7.2 Comment les poids évoluent-ils ?

À cause de la haute dimension des vecteurs d'entrées des neurones, la cosinus distance des poids finaux d'une couche par rapport aux poids initiaux peut augmenter relativement facilement. Effectivement, il suffit de modifier un nombre limité d'entrées pour pouvoir arriver à une cosinus distance proche de 1 et ainsi être "orthogonal" par rapport à la situation initiale. Le principal questionnement de cette section est de comprendre comment les poids d'une couche se retrouve "orthogonaux" à leurs poids initiaux.

Pour répondre à cela, nous avons différentes pistes. Est-ce qu'un petit *ensemble fini* des poids changent fortement pendant l'entraînement ? Est-ce que *tous* les poids changent légèrement ? Ou est-ce un compromis entre les deux ?

7.2.1 Distribution du changement des poids $\Delta w^{(l)}$

Afin de comprendre la façon dont a évolué chaque poids d'une couche après l'apprentissage, intéressons nous à la quantité $\Delta w^{(l)}$, représentant le changement des poids d'une couche entre les états initiaux et finaux. Formellement, soient respectivement $w_0^{(l)}$ et $w_n^{(l)}$, le vecteur poids de la couche l à l'initialisation et après n epochs. La quantité d'intérêt est

$$\Delta w^{(l)} = w_n^{(l)} - w_0^{(l)}$$

Note : pour rappel, le nombre d'epochs n est différent pour chaque modèle mais le nombre de couche est identique ; $l = 0, 1, \ldots, 14$.

Jetons maintenant à nouveau un oeil sur courbes de cosinus distance des trois optimiseurs (6.4, 6.5, 6.6). Puisque la cosinus distance de *sgd* atteint une plus petite valeur sur l'ensemble des couches par rapport à *weight decay* ou *layca*, on s'attend dès lors à ce que ses poids aient moins changé. Regardons à présent ce qu'il en est.

Les FIGURES 7.1, 7.2 et 7.3 montrent la distribution de $\Delta w^{(l)}$ pour l'ensemble des couches, et ce pour chaque modèle.

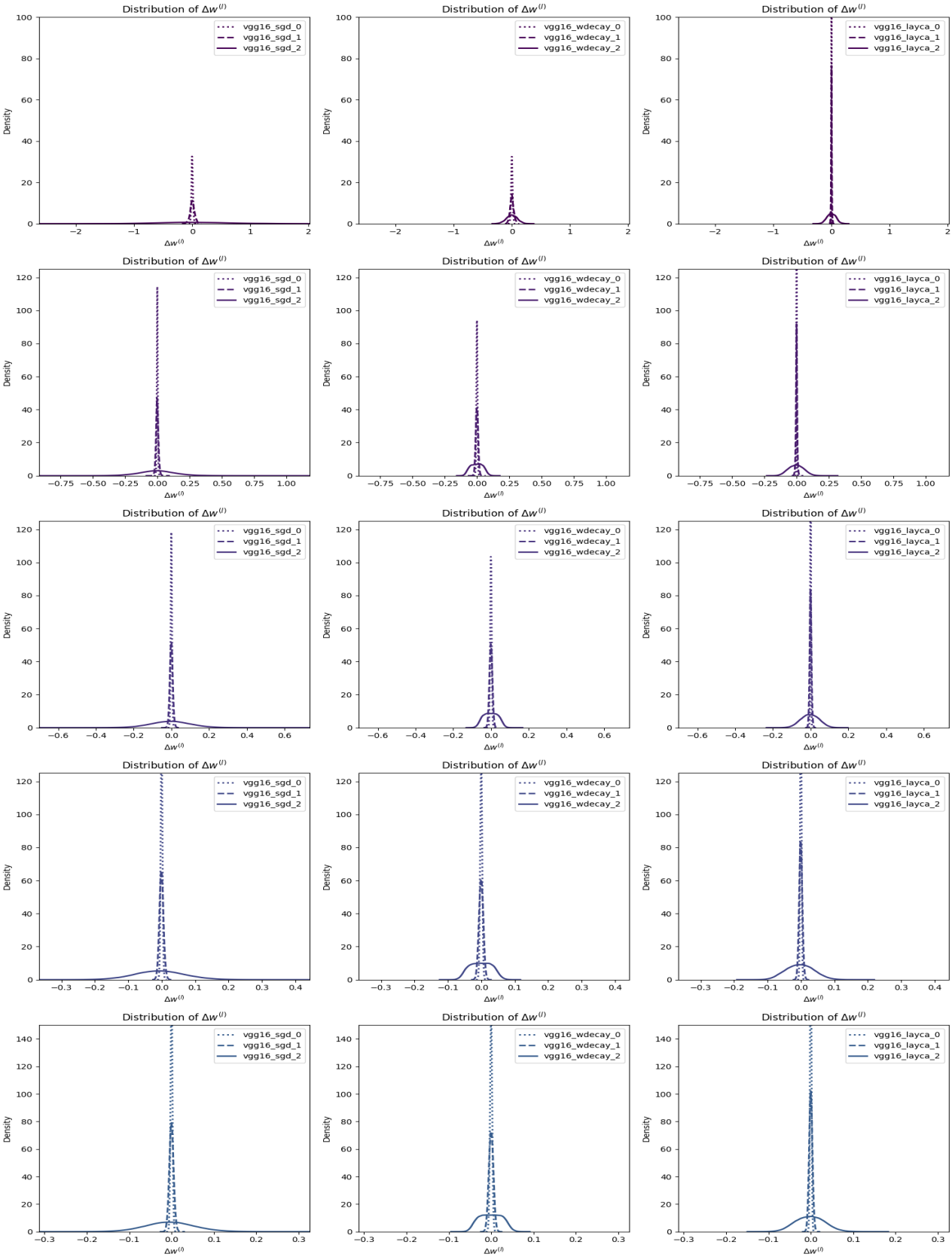


FIGURE 7.1 – **Distribution de $\Delta w^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la première couche, la deuxième ligne à la deuxième couche, etc.

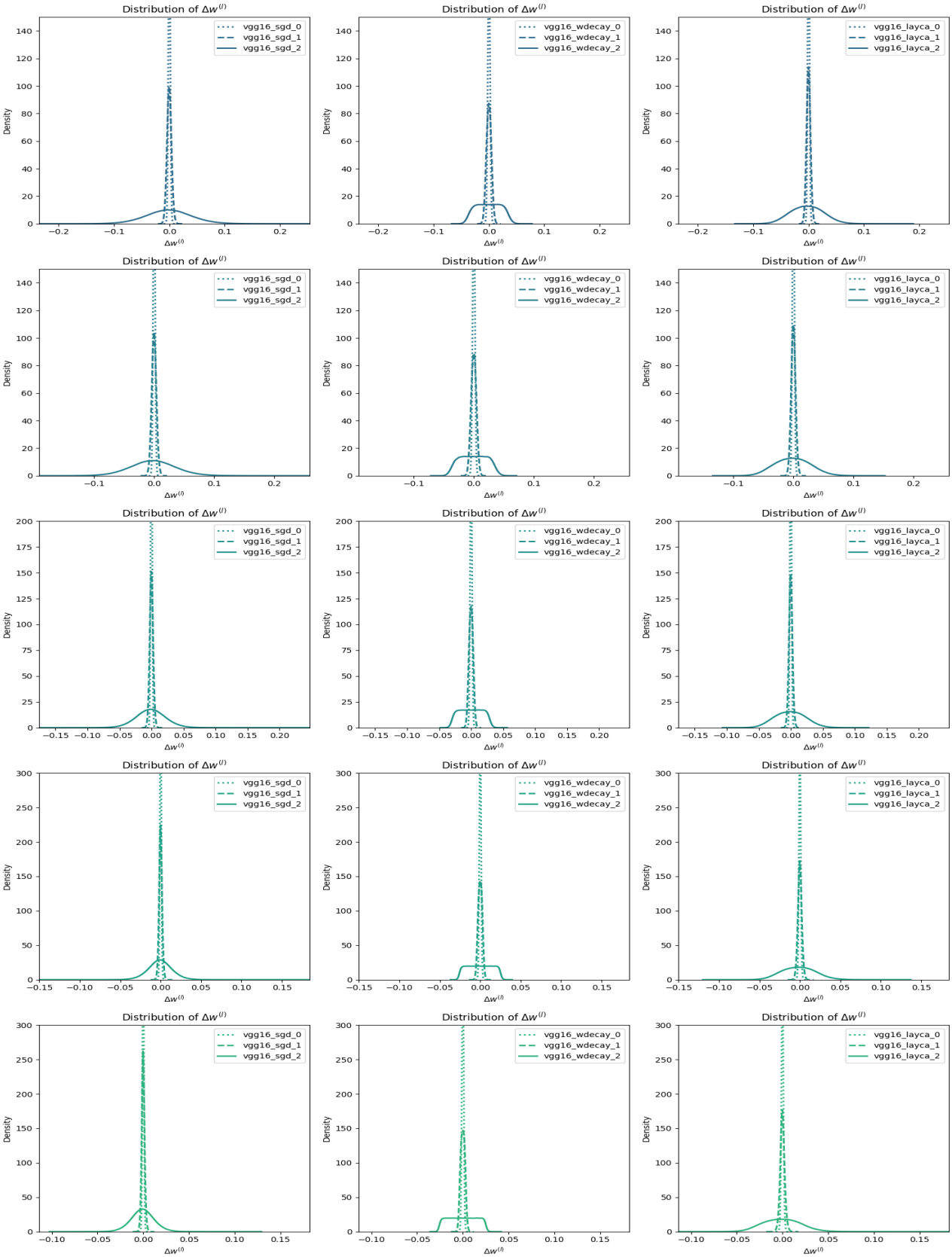


FIGURE 7.2 – **Distribution de $\Delta w^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la sixième couche, la deuxième ligne à la septième couche, etc.

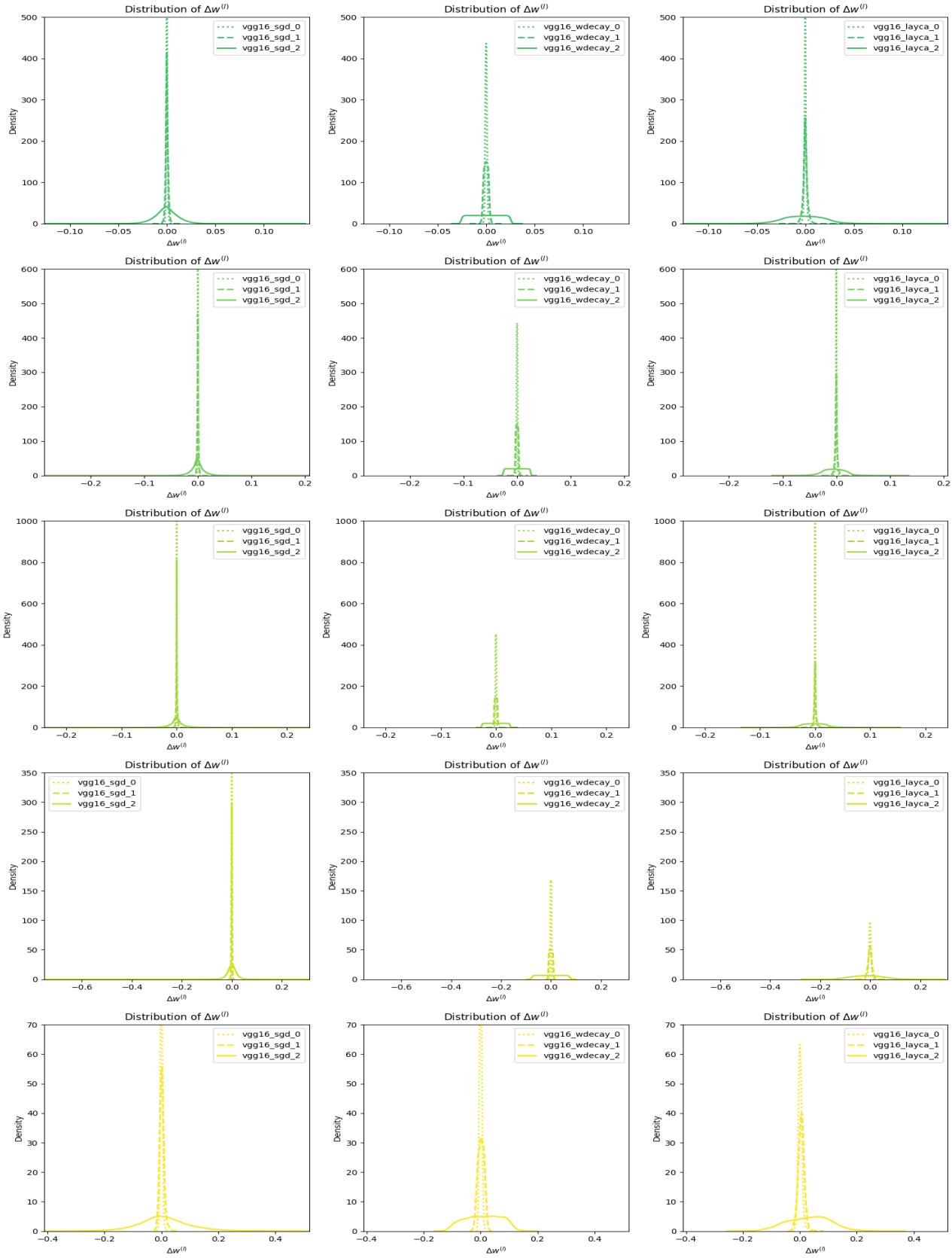


FIGURE 7.3 – **Distribution de $\Delta w^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la dixième couche, la deuxième ligne à la onzième couche, etc.

Premièrement, on remarque que la distribution du meilleur modèle *sgd* s'étale globalement de moins en moins au aval des couches (en aval veut dire dans la direction de la sortie du réseau), ce qui expliquerait pourquoi sa cosinus distance diminue aussi en aval FIGURE (voir 6.4).

On remarque aussi que l'étalement de la distribution est plus prononcé pour le meilleur modèle *sgd* que pour les moins bons modèles. Cette observation est aussi valable pour les deux autres optimiseurs. À ce stade, on pourrait imaginer qu'un réseau qui changent ses poids plus uniformément et avec une plus grande gamme de variation aurait tendance à mieux généraliser.

Si l'on fait une comparaison horizontale, on remarque que les distributions des trois optimiseurs ont une gamme de variation assez similaire. Ceci n'explique donc pas pourquoi la cosinus distance de *sgd* est moins importante que pour *weight decay* et *layca*.

On notera aussi que les distributions du meilleur modèle pour *weight decay* semblent êtres suspicieusement uniformes, davantage en aval des couches. Nous pensons que cela pourrait s'expliquer par l'aspect régularisateur de *weight decay*, empêchant de trop grandes variations de poids de se produire à long terme.

7.2.2 Distribution du changement des poids $\Delta\bar{w}^{(l)}$

Les distributions précédentes nous révèlent qu'une variation plus importante des poids serait signe d'une bonne généralisation mais n'expliquent pas les différences de cosinus distance entre les optimiseurs. D'un autre côté, un changement de poids, aussi grand soit-il, risque de n'avoir que peu d'impact sur la structure du réseau s'il est petit *relativement* à la norme des poids. Afin de se départir de la norme du poids, intéressons nous maintenant à la quantité $\Delta\bar{w}^{(l)}$ [1], représentant le vecteur variation de poids normalisé par le vecteur de poids final

$$\Delta\bar{w}^{(l)} = \frac{\Delta w^{(l)}}{||w_n^{(l)}||}$$

Les FIGURES 7.4, 7.5 et 7.6 montrent la distribution de $\Delta\bar{w}^{(l)}$ pour l'ensemble des couches, et ce pour chaque modèle.

[1] On a choisi la notation $\bar{w}$ à la place de $\hat{w}$ car cette dernière aurait pu être confondue avec la notation du vecteur unitaire.

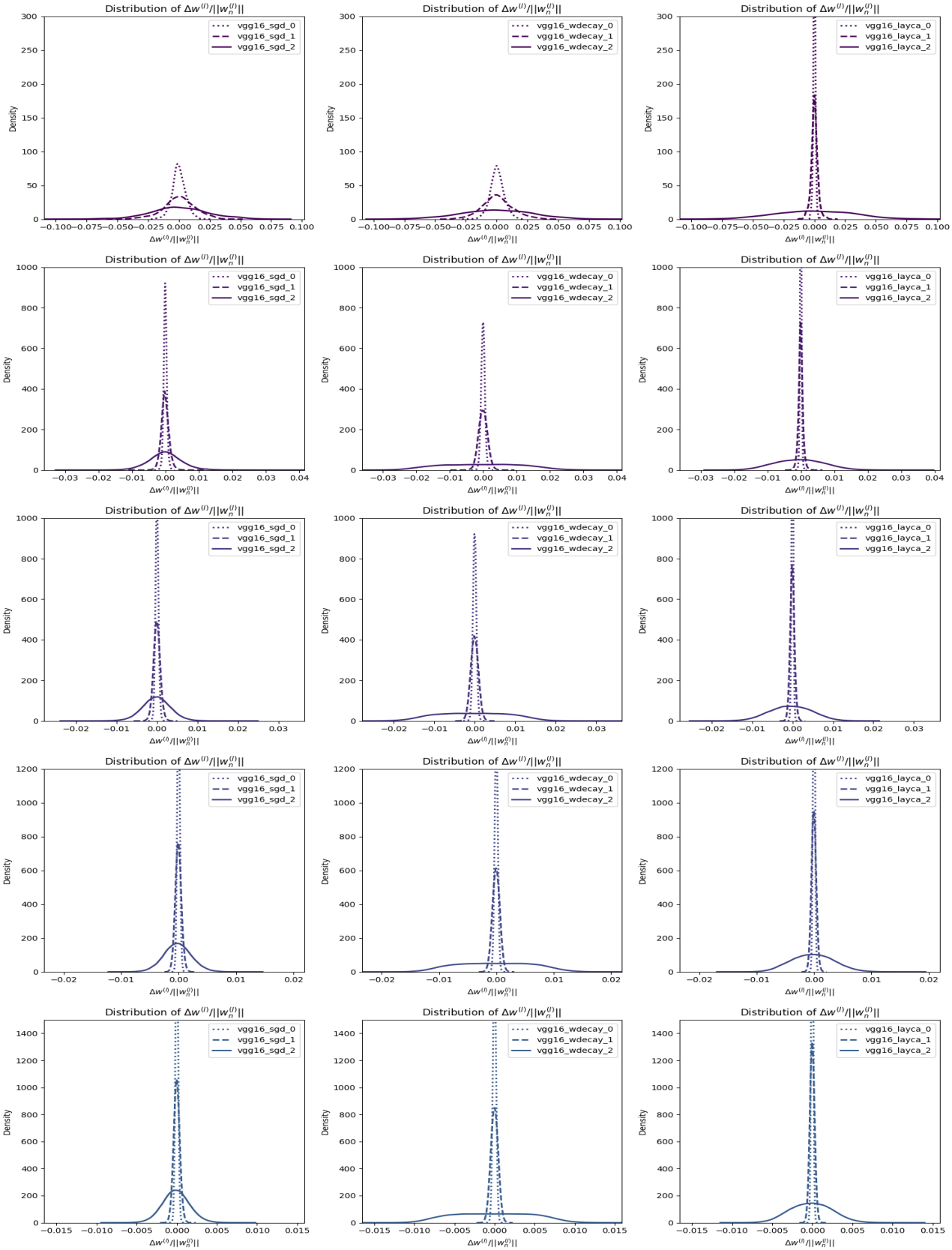


FIGURE 7.4 – **Distribution de $\Delta\bar{w}^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la première couche, la deuxième ligne à la deuxième couche, etc.

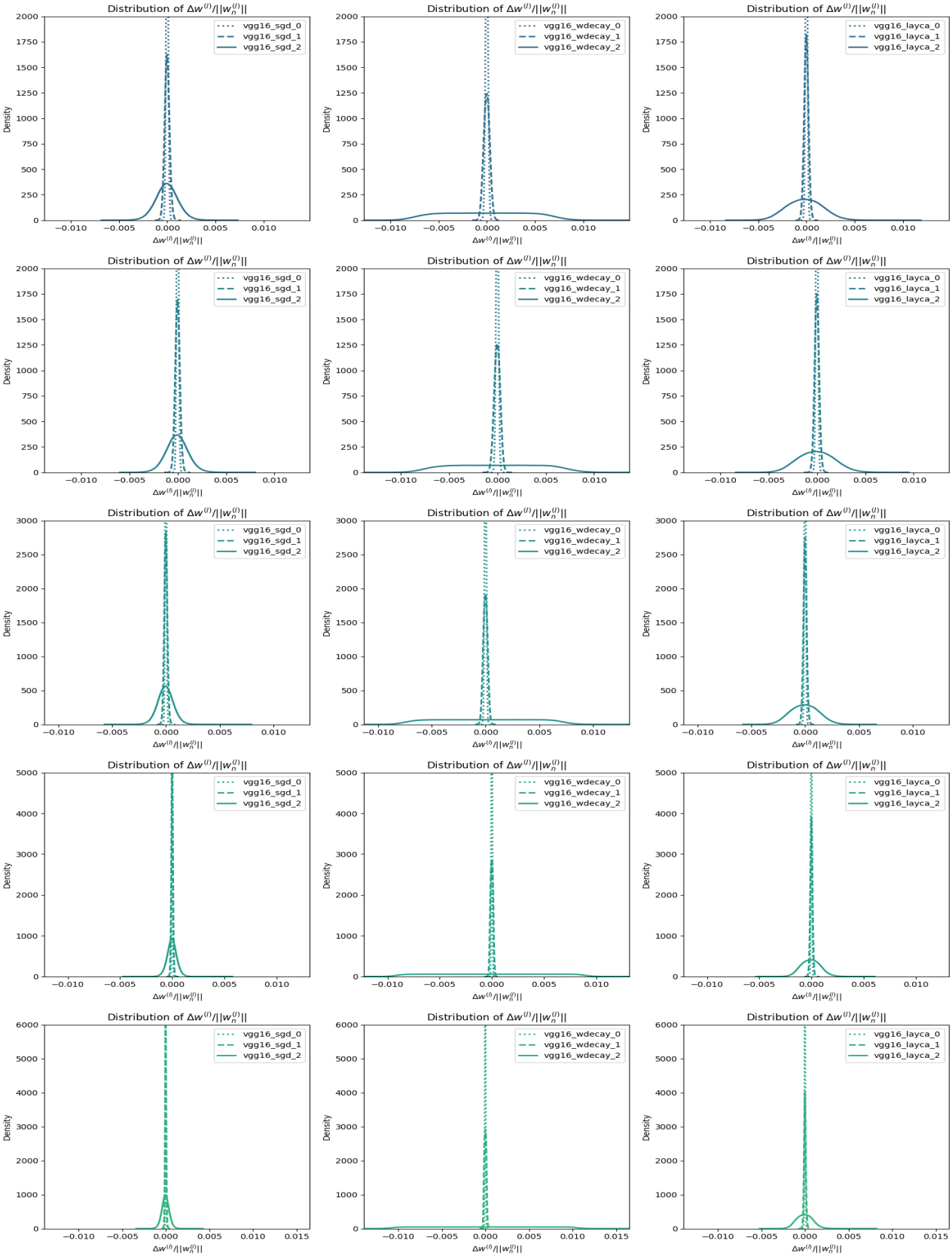


FIGURE 7.5 – **Distribution de $\Delta\bar{w}^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la sixième couche, la deuxième ligne à la septième couche, etc.

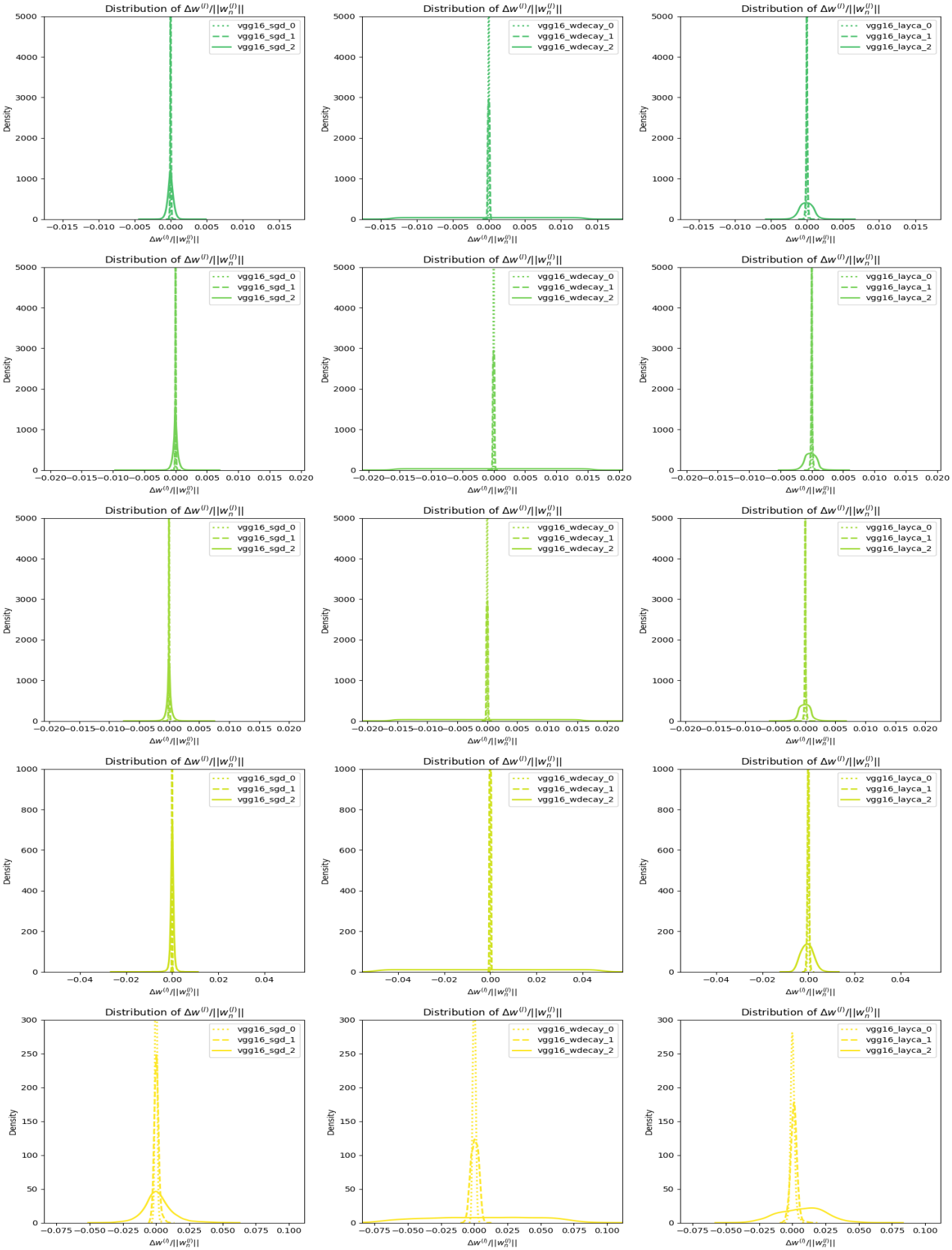


FIGURE 7.6 – **Distribution de $\Delta\bar{w}^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la dixième couche, la deuxième ligne à la onzième couche, etc.

Premièrement, on observe toujours un redressement en aval des couches de la distribution des poids pour *sgd*. Cependant, on observe aussi ce redressement pour *layca*, ce qui met à péril notre hypothèse de la section précédente comme quoi la diminution du cosinus distance au fil des couches pour *sgd* serait liée à ce redressement.

En outre, la gamme de variation des distributions *sgd* semble globalement plus étroite que pour les deux autres optimiseurs. Ceci pourrait peut-être expliquer pourquoi sa cosinus distance est moins grande.

On observe également que les distributions du meilleur modèle *weight decay* s'étalent significativement plus que pour les deux autres optimiseurs. Ceci indique que bien que l'étalement de la distribution soit signe de bonne généralisation, la grande taille de l'étalement n'est pas nécessairement liée à une meilleure généralisation. C'est à dire que même si l'étalement est beaucoup plus important, cela ne veut pas dire qu'il y aura forcément un gain de généralisation de même importance.

Enfin, on observe que les distributions des modèles 65% et 75% sont très étroites, indiquant que la majorité de leur poids ne changent pas significativement. Intuitivement, cela montre que le réseau dans son entièreté n'est pas bien exploité.

En somme, ces résultats montrent un lien intéressant entre généralisation et variation des poids, indiquant qu'en moyenne, des variations d'amplitudes plus importantes et plus uniformément distribuées sur l'ensemble des poids seraient signe de bonne généralisation. Cependant, ces observations sont incomplètes et insuffisantes pour expliquer les courbes de cosinus distance des optimiseurs.

7.2.3 La cosinus distance *relookée*

Par une certaine mise en évidence dans l'équation 7.1, on peut mieux comprendre le lien entre la cosinus distance et la variation de poids $\Delta\bar{w}^{(l)}$.

$$
\begin{aligned}
\phi^{(l)} &= 1 - \frac{w_0^{(l)} \bullet w_n^{(l)}}{||w_0^{(l)}|| \cdot ||w_n^{(l)}||} \\
&= 1 - \frac{w_0^{(l)} \bullet (w_0^{(l)} + \Delta w^{(l)})}{||w_0^{(l)}|| \cdot ||w_n^{(l)}||} \\
&= 1 - \frac{||w_0^{(l)}||^2 + w_0^{(l)} \bullet \Delta w^{(l)}}{||w_0^{(l)}|| \cdot ||w_n^{(l)}||} \\
&= 1 - \frac{||w_0^{(l)}||}{||w_n^{(l)}||} - \hat{w}_0^{(l)} \bullet \frac{\Delta w^{(l)}}{||w_n^{(l)}||} \\
&= 1 - \frac{||w_0^{(l)}||}{||w_n^{(l)}||} - \hat{w}_0^{(l)} \bullet \Delta \bar{w}^{(l)}
\end{aligned}
\tag{7.1}
$$

On remarque alors qu'il y a deux termes importants qui rentre en jeu dans l'équation de la cosinus distance [2]. On a d'abord le terme $||w_0^{(l)}||/||w_n^{(l)}||$ qui est lié à la façon dont la norme total des poids de la couche évolue au cours de l'apprentissage ; on voit donc qu'une manière d'augmenter la cosinus distance est d'augmenter la norme $||w_n^{(l)}||$. Le deuxième terme $\hat{w}_0^{(l)} \bullet \Delta \bar{w}^{(l)}$ est lié à la direction du vecteur variation de poids $\Delta \bar{w}^{(l)}$ par rapport au vecteur poids initial $\hat{w}_0^{(l)}$. Pour augmenter la cosinus distance, un optimiseur peut rendre ce terme nul ou négatif. Le produit scalaire de deux vecteurs étant la somme du produit de leurs composantes, une façon de le rendre très négatif est de par exemple, avoir une majorité de poids subissant un changement de signe opposé à leur signe de départ. Finalement, on remarquera aussi que ces deux termes ne sont pas indépendants et son liés par le terme $||w_n^{(l)}||$.

Analysons maintenant ces quantités sur l'ensemble des modèles, illustrés sur la FIGURE 7.7.

[2] On se rappellera que les termes en w_0 sont identiques pour les neuf modèles.

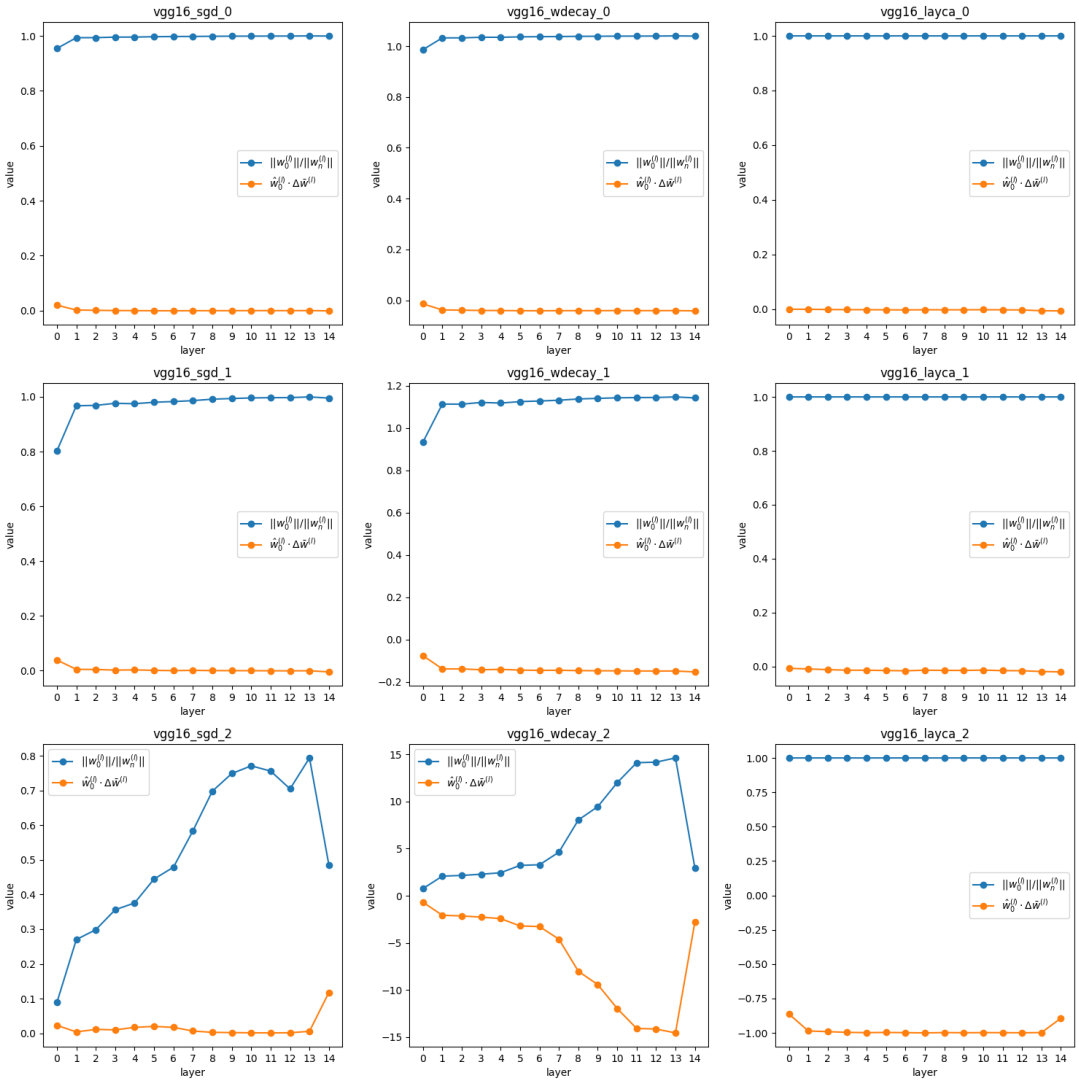


FIGURE 7.7 – **Évolution des termes $\frac{||w_0^{(l)}||}{||w_n^{(l)}||}$ et $\hat{w}_0^{(l)} \cdot \Delta\bar{w}^{(l)}$ à travers les couches, pour l'ensemble des modèles**. L'axe horizontal représente les couches du réseau.

Premièrement, on observe que les courbes entre les modèles 65% et 75% sont très similaires. Dans ces modèles, on remarque que la norme des poids de chaque couche n'a pas beaucoup changé (courbe bleu). Nous pensons que le plus faible *learning rate* pour ces modèles en serait la cause, ne donnant pas assez de poids au gradient à chaque *update* pour changer de manière significative la structure du réseau. On remarque aussi pour ces modèles que le produit $\hat{w}_0^{(l)} \cdot \Delta\bar{w}^{(l)}$ est quasi nul à travers les couches (courbe orange).

On observe aussi une augmentation générale de la norme du poids final pour le meilleur modèle *sgd* 85% (courbe bleu). L'augmentation est de moins en moins forte en aval des couches. Nous pensons que cela pourrait s'expliquer par le fait que au plus on avance dans les couches, au moins un neurone aura besoin d'évidences pour décider si oui ou non la *feature* (caractéristique) qu'il représente est activée par une image. Par exemple, dans les couches plus en aval, un neurone qui déciderait de l'activation conceptuelle d'une patte de chat (*feature* abstraite typiquement

retrouvée plus en aval) hypothétiquement présente dans l'image ne doit tenir compte que de quelques éléments de la couche précédente. Ceci implique qu'une minorité de ses poids sera importante en fin d'apprentissage. Ainsi, l'apprentissage n'aura pas augmenter une grande partie de ses poids, ce qui explique le fait que le vecteur final de poids de la couche ait augmenté mais moins que dans les premières couches.

Tout comme pour les moins bons modèles, le terme $\hat{w}_0^{(l)} \bullet \Delta\bar{w}^{(l)}$ pour le modèle 85% de *sgd* est presque nul à travers les couches (courbe orange). Ce produit scalaire peut s'annuler de plusieurs manière mais à priori, cela signifierait que les poids des couches ont subi équitablement soit des variations de même signe que leur valeur initiale, soit des variations de signe contraire. Notons que ce sont les variations de signe contraire qui augmente la cosinus distance (les termes négatifs du produit $\hat{w}_0^{(l)} \bullet \Delta\bar{w}^{(l)} = \sum_i \hat{w}_{0i}^{(l)} \cdot \Delta\bar{w}_i^{(l)}$). Si ce comportement est intrinsèque à l'optimiseur *sgd*, cela voudrait dire que la seule façon que cet optimiseur à d'obtenir une cosinus distance importante serait d'augmenter la norme du vecteur poids de ses couches. Cette hypothèse nous pousse à nous poser la question de savoir s'il est possible pour *sgd* d'arriver à des courbes de cosinus distance similaires aux autres optimiseurs en augmentant de manière uniforme cette norme à travers les couches.

Une autre observation que l'on peut faire se situe au niveau des courbes du meilleur modèle de *weight decay* : elles sont curieusement symétriques. Le premier terme $\frac{||w_0^{(l)}||}{||w_n^{(l)}||}$ augmente globalement au fil des couches, ce qui implique que la norme des poids des couches diminue en aval. Ce premier terme est quasi symétrique au terme $\hat{w}_0^{(l)} \bullet \Delta\bar{w}^{(l)}$, ce qui permet à ce modèle d'avoir une cosinus distance proche de 1 sur l'ensemble de ses couches.

Enfin pour le modèle 85% de *layca*, on voit que le terme $\frac{||w_0^{(l)}||}{||w_n^{(l)}||}$ reste à 1. Ce n'est pas surprenant vu l'implémentation de *layca* qui maintient, à chaque itération, la taille du vecteur poids de chaque couche à sa valeur initiale. En outre, le terme $\hat{w}_0^{(l)} \bullet \Delta\bar{w}^{(l)}$ vaut -1 sur toutes les couches (modulo la première et la dernière). Comme pour *weight decay*, ce dernier terme est curieusement symétrique au premier. Aussi pour ces deux modèles, les termes $\hat{w}_0^{(l)} \bullet \Delta\bar{w}^{(l)}$ sont négatifs. Cela signifierait qu'une majorité des poids ont subi des variations de signe contraire à leur signe de départ.

En résumé, les résultats cette section nous permettent de mieux comprendre les facteurs qui influencent la cosinus distance des couches du réseau, ainsi que les mécanismes d'apprentissage de ces optimiseurs.

Dans la prochaine section, nous investiguons davantage cette relation en étudiant les distributions des poids finaux des modèles.

7.2.4 Distribution des poids

On s'intéresse maintenant à la distribution du vecteur poids finaux des couches de nos modèles que l'on a représenté sur les FIGURES 7.8, 7.9 et 7.10. Le but de cette expérience est de mettre sous une autre perspective les observations faites sur la norme finale et la direction de variation des poids observés dans la section précédente.

Premièrement, on observe que les distributions des modèles 85% sont très différentes de celles des modèles 65% et 75%. Les distributions de ces derniers sont très proches de la distribution initiale des poids, indiquant un changement très faible de la structure du réseau dans son ensemble. En fait, cette observation à déjà été faite dans la première section de ce chapitre sur les FIGURE 7.1, 7.2 et 7.3.

Pour les modèles 85%, on voit que la forme de leurs distributions est assez similaire : elles sont plus bombés que celles des moins bons modèles. Il y a également une différence notable entre l'optimiseur *sgd* et les optimiseurs *weight decay* et *layca*. Pour *sgd*, la distribution s'étale par rapport à la situation initiale tandis que pour les deux autres, elle se redresse. Ce redressement est de plus en plus important pour *wdecay* en aval, confirmant la diminution de l'amplitude de ses poids au fil des couches observée sur la FIGURE 7.7. Pour ces deux derniers optimiseurs seulement, on a donc une augmentation du nombre de poids proche de zéro.

Cette observation peut également être mise en lien avec le fait que le terme $\hat{w}_0^{(l)} \cdot \Delta\bar{w}^{(l)}$ soit nul pour *sgd* et négatif pour *weight decay* et *layca*. Comment discuté dans la sous-section précédente, un produit scalaire négatif signifie qu'une majorité des poids ont changés avec une variation de signe contraire à leur signe de départ.

On met donc bien en avant le fait que *weight decay* et *layca* sont des mécanismes appliqués à *sgd* qui mettent à zéro certains poids. Ces poids auront alors effectivement subis une variation de signe contraire à leur signe initiale et d'amplitude égale à leur valeur initiale, ce qui rendra le terme $\hat{w}_0^{(l)} \cdot \Delta\bar{w}^{(l)}$ négatif.

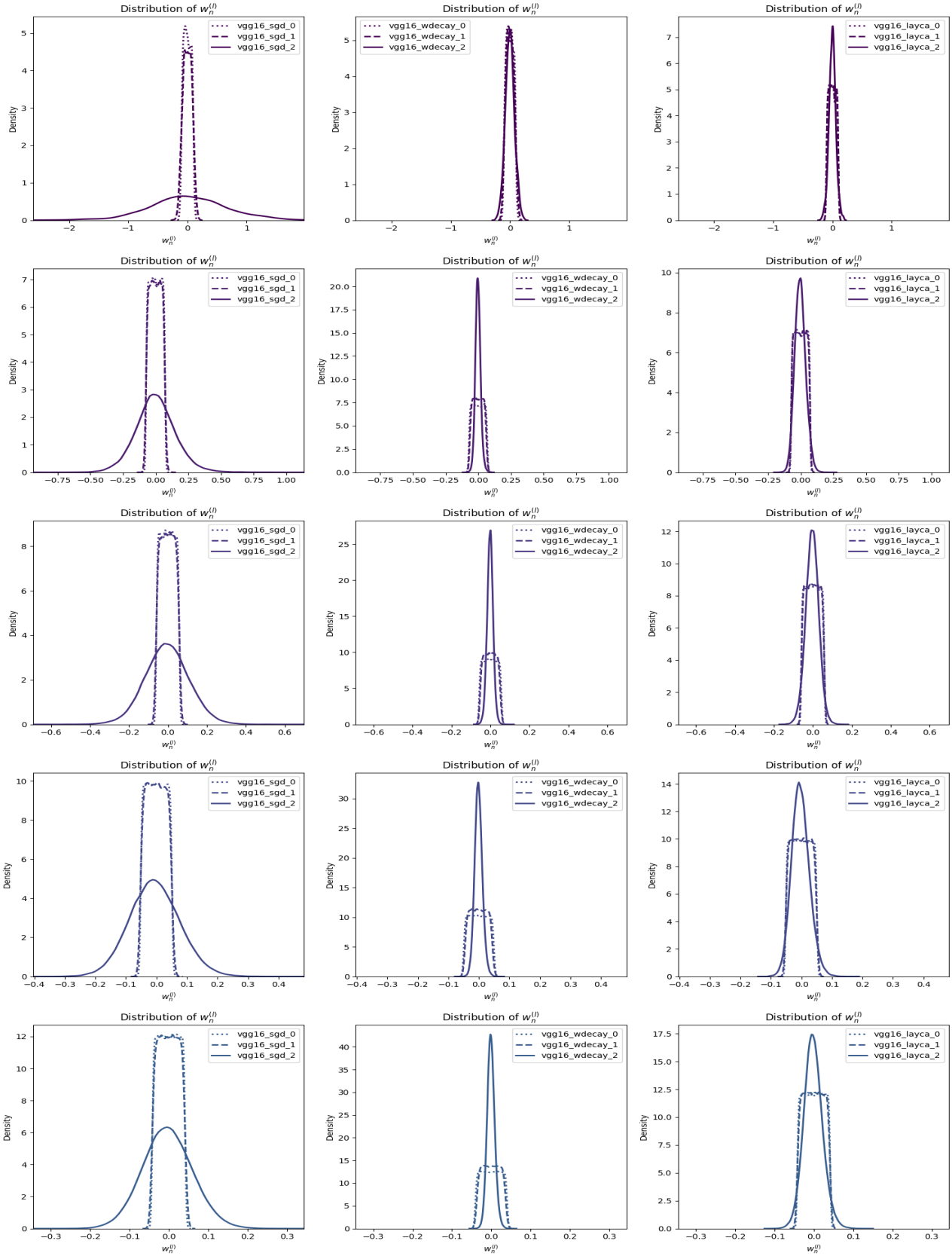


FIGURE 7.8 – **Distribution de $w_n^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la première couche, la deuxième ligne à la deuxième couche, etc.

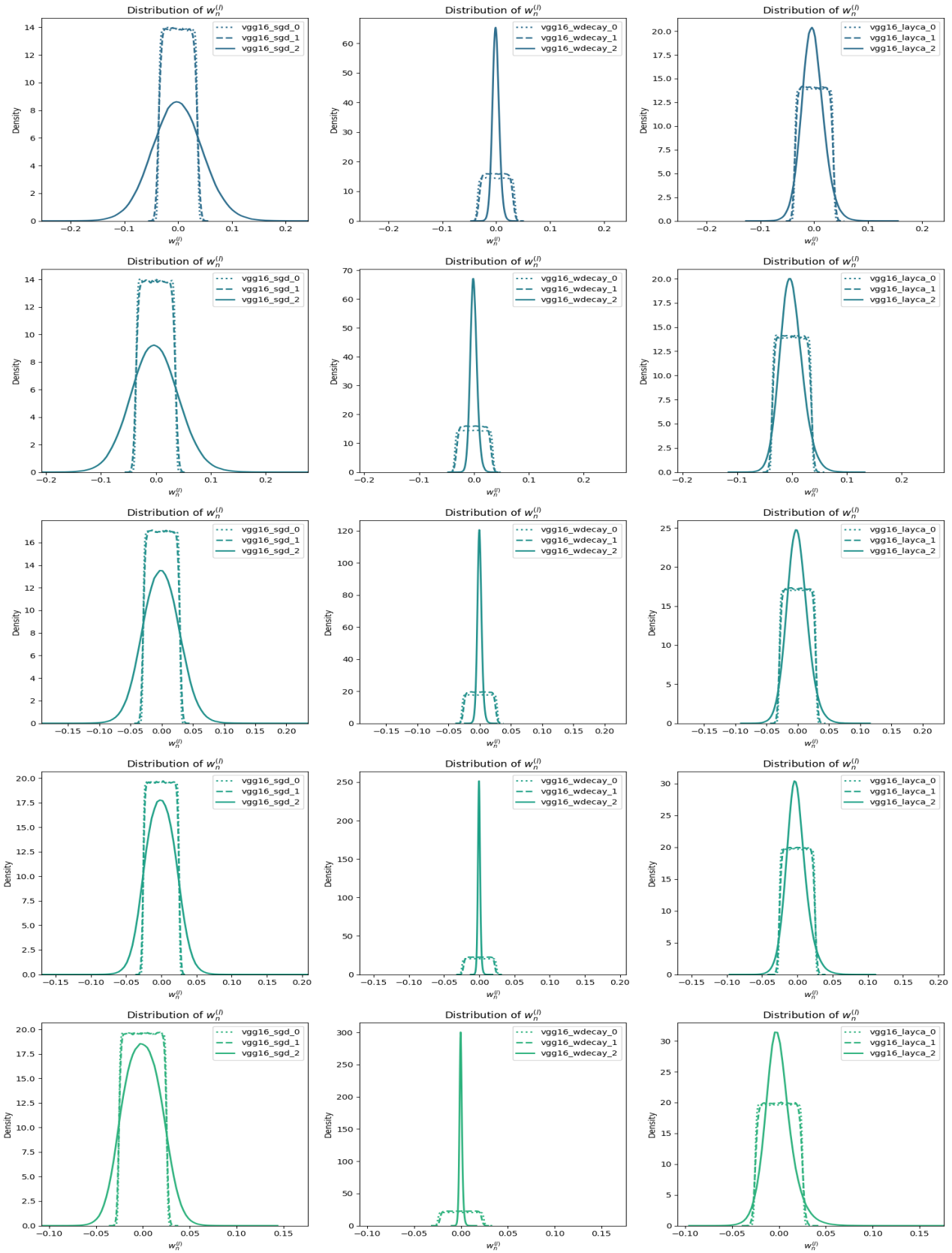


FIGURE 7.9 – **Distribution de $w_n^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la première couche, la deuxième ligne à la deuxième couche, etc.

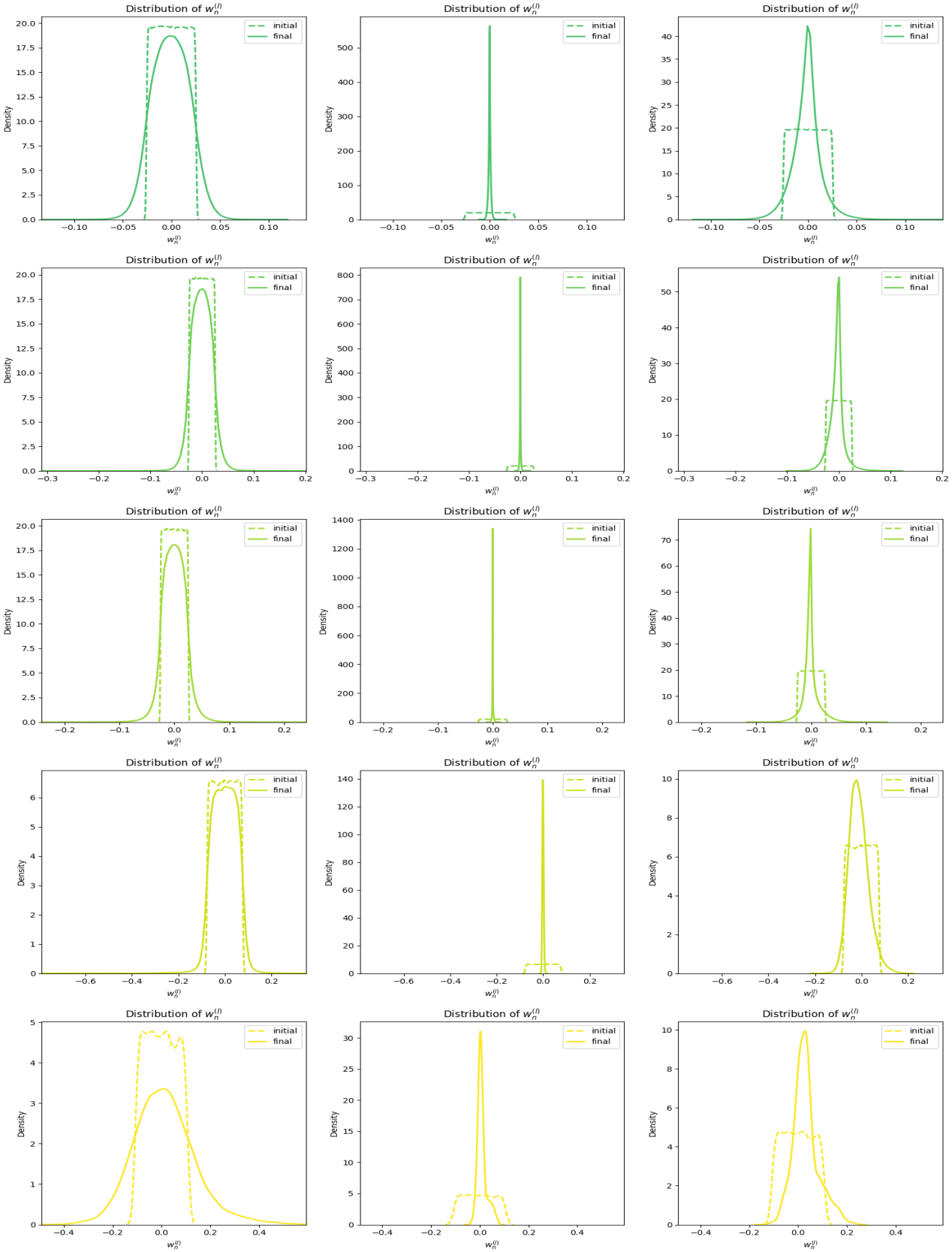


FIGURE 7.10 – **Distribution de $w_n^{(l)}$ pour l'ensemble des modèles sur l'ensemble des couches**. Les distributions des modèles *sgd*, *weight decay* et *layca* sont respectivement représentées sur la colonne de gauche, du milieu et de droite. La première ligne correspond à la première couche, la deuxième ligne à la deuxième couche, etc.

Ces poids mis à zéro sont donc des poids que les optimiseurs *weight decay* et *layca* ont jugé inutiles à la classification (appelés *poids parasites*). Nous verrons dans la section suivante que ce sont ces poids qui les rendent plus robustes au *pruning*.

Suite à ces observations, il est légitime de se demander pourquoi *weight decay* et *layca* favorisent ce genre de comportement.

7.2.5 La mise à zéro des poids parasites

Weight decay

Au cours de l'apprentissage, si un poids est inutile à la classification, il subira des variations tantôt dans un sens, tantôt dans le sens opposé, mais en moyenne sa valeur n'augmentera généralement pas significativement. L'optimiseur *weight decay* aura alors tendance à mettre ce poids à zéro puisque son amplitude sera réduite d'un facteur $(1-\lambda)$ à chaque *update* des poids. Ceci peut être formulé mathématiquement. Dans le développement qui suit, on laissera tomber la notation $w^{(l)}$ pour w, tout en sachant implicitement que l'on parle des poids d'une couche en particulier.

Soit $w_{t,i}$ la $i^{\text{ème}}$ composante du vecteur poids w_t après t *updates*. On peut écrire [3]

$$\begin{aligned} w_{t+1,i} &= (1-\lambda)w_{t,i} - \alpha_t \frac{\partial L}{\partial w_{t,i}} \\ &= (1-\lambda)^t w_{0,i} - \sum_{j=0}^{t} (1-\lambda)^{t-j} \alpha_j \frac{\partial L}{\partial w_{j,i}} \\ &\approx -\sum_{j=0}^{t} (1-\lambda)^{t-j} \alpha_j \frac{\partial L}{\partial w_{j,i}} \\ &\approx 0 \end{aligned} \tag{7.2}$$

La première approximation vient du fait que le nombre d'*updates* est important, ce qui amène automatiquement le terme $(1-\lambda)^t$ à zéro ou très près [4]. La deuxième approximation provient du terme $\sum_{j=0}^{t}(1-\lambda)^{t-j}\alpha_j \frac{\partial L}{\partial w_{j,i}}$ que l'on suppose aussi proche de zéro pour des raisons moins évidentes :

- le terme $(1-\lambda)^{t-j}$ annule les composantes de la somme proche du début de l'entraînement $(j=0)$.

[3] Note : les termes de *momentum* ne sont pas inclus.

[4] Dans nos simulations, $\lambda = 0.001$. Le meilleur modèle pour *weight decay* fini son entraînement après 25 *epochs*. Le nombre de données d'entraînement est 40000 et la taille d'un *batch* 128, on a donc $(1-\lambda)^t = (0.999)^{25*40000/128} = 0.0004 \approx 0$

- le terme α^j annule les composantes de la somme proche de la fin de l'entraînement ($j = t$).
- généralement le terme $\frac{\partial L}{\partial w_{j,i}}$ est petit en amplitude. Il changent souvent de signe car il n'intervient pas utilement dans le processus de classification. Il sera donc tantôt légèrement positif, tantôt légèrement négatif. Ceci est moins vrai en début d'entraînement mais de plus en plus vrai au fil du temps, car le réseau aura davantage tendance à se stabiliser à un optimum local. Cet optimum serait alors la meilleur solution pour capter les biais dans les données par rapport à la classification voulue. Ce dernier point reste à confirmer empiriquement.

Layca

L'optimiseur *layca* effectuera aussi cette mise à zéro des poids parasites. Formellement, supposons que le vecteur poids de la couche puisse se décomposer en deux vecteurs de bases orthogonales :

$$w = w_p + w_u$$

où w_p et w_u représentent respectivement la partie des poids parasites et utiles de la couche. Puisque *layca* maintient constante la norme du vecteur poids, on a $||w|| = ||w_0||$. Généralement, le gradient sera dans une direction favorisant l'augmentation de w_u. On voit que pour maintenir une norme constante, augmenter la composante utile à pour conséquence de diminuer la composante parasite *dans le long terme*. Ce phénomène est illustré en deux dimensions sur la FIGURE 7.11.

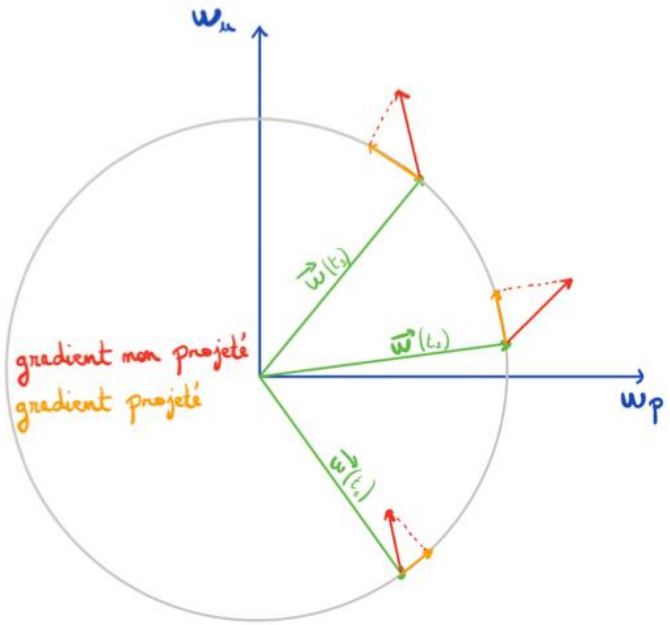


FIGURE 7.11 – Mise à zéro des liens parasites dans le cas de l'optimiseur *layca*.

La curieuse symétrie expliquée

On peut voir sur la FIGURE 7.7 une curieuse symétrie entre les termes $\frac{||w_0^{(l)}||}{||w_n^{(l)}||}$ et $\hat{w}_0^{(l)} \cdot \Delta\bar{w}^{(l)}$ pour *weight decay* 85% et *layca* 85%. En effet, pour la majorité des couches, on peut observe

$$\hat{w}_0 \cdot \Delta\bar{w} \approx -\frac{||w_0||}{||w_n||}$$

Si on développe la relation, on a

$$\begin{aligned}
&\frac{w_0}{||w_0||} \cdot \frac{\Delta w}{||w_n||} \approx -\frac{||w_0||}{||w_n||} \\
\iff & w_0 \cdot \Delta w \approx -||w_0||^2 \\
\iff & w_{0,1}\Delta w_1 + w_{0,2}\Delta w_2 + \ldots + w_{0,m}\Delta w_m \approx -(w_{0,1}^2 + w_{0,2}^2 + \ldots + w_{0,m}^2) \\
\iff & w_{0,1}(-\Delta w_1) + w_{0,2}(-\Delta w_2) + \ldots + w_{0,m}(-\Delta w_m) \approx w_{0,1}^2 + w_{0,2}^2 + \ldots + w_{0,m}^2
\end{aligned} \tag{7.3}$$

où m représente le nombre de poids de la couche. Un poids w_i mis à zéro au cours de l'apprentissage vérifie l'équation $\Delta w_i \approx -w_{0,i}$.

On remarque qu'au plus une couche possède un nombre important de ces poids parasites, au plus la similitude est vérifiée.

On conclus donc que cette symétrie réside dans la capacité de l'optimiseur à mettre à zéro un grand nombre de poids pendant l'entraînement.

7.3 Et si on l'ampute ?

Les travaux menés par [Zho+19] et [AS19] montrent qu'il est possible d'enlever certains poids du réseau convolutif sans perturber significativement sa capacité de généralisation. Dans cette section, nous appliquons aussi cette élagage sur une partie des poids de nos modèles. In fine, notre objectif est de déceler des comportements différents entre les modèles ainsi que de confirmer les résultats trouvés dans la section précédente.

7.3.1 *Pruning* aléatoire

Dans un premier temps, nous avons investiguons le comportement de nos modèles lorsqu'on leur enlève une partie de leur poids de manière aléatoire, suite à quoi nous avons évaluons les performances des modèles ainsi amputés.

Cette analyse nous permet d'évaluer *la robustesse* de nos modèles, c.à.d. leur capacité à préserver leur *test accuracy* face au *pruning*. Notre intuition est qu'un modèle plus (resp. moins) performant sera plus (resp. moins) robuste face la mise à zéro d'une partie de ses poids. Ce comportement serait dû au même phénomène qui rend *dropout* efficace.

Complétant ainsi l'explication communément admise de *dropout* (c.f. **Chapitre 4 : Préliminaires**), on se rendra compte dans ce travail que cette robustesse est directement en lien avec le fait que le réseau peut être vu comme constitué de plusieurs réseaux dont chacun d'entre eux permet la détection d'un type de biais dans les données entrées.

Les FIGURES 7.12, 7.13 et 7.14 montre les résultats d'une expérience que nous avons mené sur nos modèles. Nous avons tester les performances (*test accuracy*) de nos modèles lorsqu'on leur enlevait un certain pourcentage de leurs poids de manière aléatoire sur l'ensemble des couches.

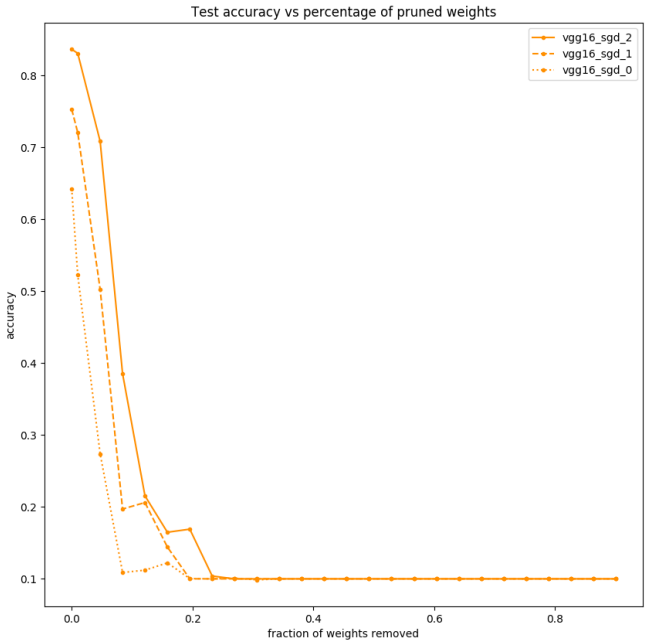


FIGURE 7.12 – **Évolution du *test accuracy* en fonction de la fraction des poids enlevés pour les modèles *sgd*.**

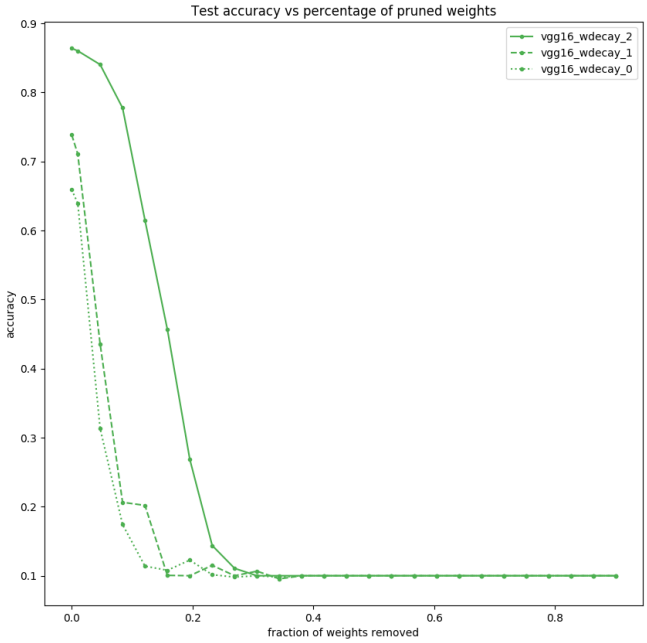


FIGURE 7.13 – **Évolution du *test accuracy* en fonction de la fraction des poids enlevés pour les modèles *weight decay*.**

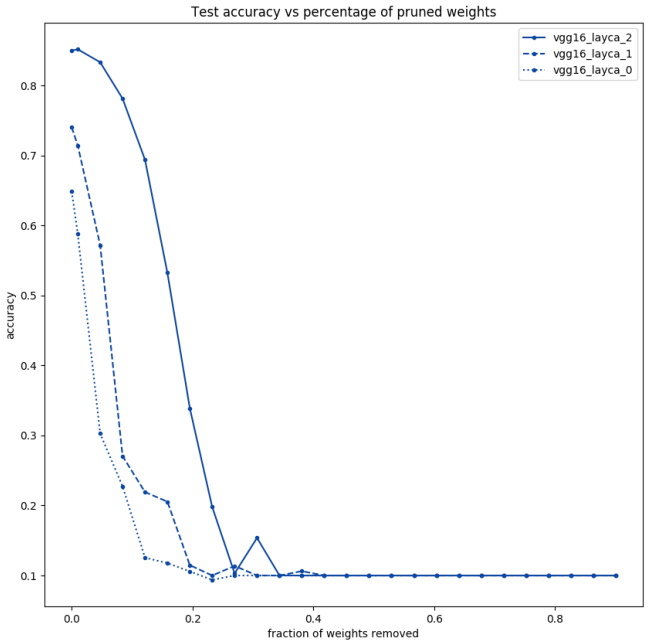


FIGURE 7.14 – **Évolution du *test accuracy* en fonction de la fraction des poids enlevés pour les modèles *layca*.**

On voit que pour l'ensemble des modèles, les performances tombent assez rapidement. On observe que les meilleurs modèles des trois optimiseurs sont plus robustes à l'amputation comme nous l'avions prédit. Cependant, on voit que le modèle 85% de *sgd* est moins robuste que celui de *wdecay* ou *layca*. Une hypothèse – affirmée durant le prochain chapitre – est que la structure du réseau créée durant l'apprentissage est directement liée à la résistance à l'élagage (le *prunning*), et ainsi également directement liée à la généralisation.

C'est ainsi que pour un optimiseur comme *sgd*, qui crée une structure utilisant de moins en moins des neurones du réseaux au fur et a mesure que ses performances augmentent, mais avec des neurones qui prennent en compte beaucoup d'entrées, sera directement plus sensible à l'élagage aléatoire.

Des optimiseurs tels que *weight decay* et *layca* vont quant à eux créer une structure utilisant plus de ressources, des neurones, du réseau, mais ceux-ci prendront moins d'entrées en compte. Ce comportement permet ainsi d'être plus résistant à l'élagage car il y a effectivement moins de poids utiles utilisés par neurones et plus de chemins possibles pour l'information.

Une hypothèse sous-jacente de celle-ci est que la diversité des biais captés par un modèle utilisant *weight decay* ou *layca* sera plus importante que celle d'un modèle utilisant *sgd*, de par

le nombre de chemins possibles pour l'information grandissant. [5].

7.3.2 Pruning des poids faibles

Dans un second temps, nous avons étudié la robustesse de nos modèles face à l'amputation de leur plus petit poids (en amplitude). La sortie d'un réseau étant principalement influencé par les poids de grande taille, nous imaginions que les réseaux seraient assez robustes à ce type d'amputation. Les FIGURES 7.15, 7.16 et 7.17 montre le *test accuracy* en fonction de la fraction des poids enlevés pour chacun des optimiseurs.

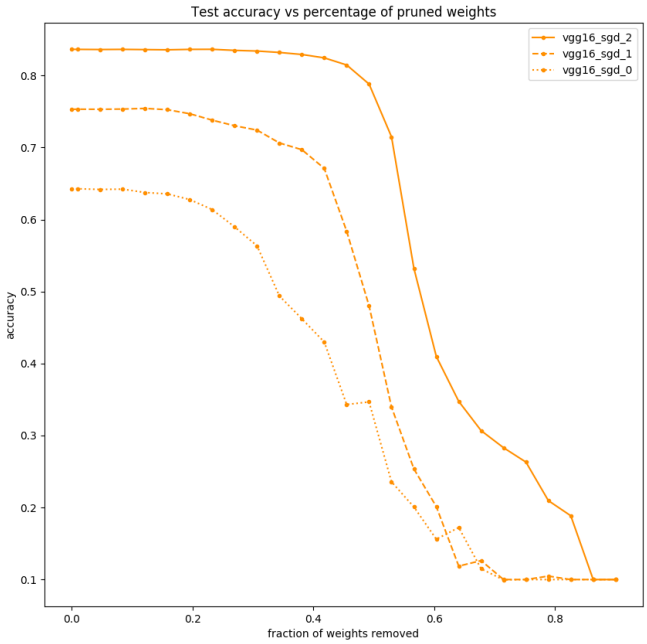


FIGURE 7.15 – **Évolution du *test accuracy* en fonction de la fraction des poids enlevés pour les modèles *sgd*.**

[5]Ceci expliquerait pourquoi le meilleur modèle *sgd* peine à dépasser les 84% de *validation accuracy*.

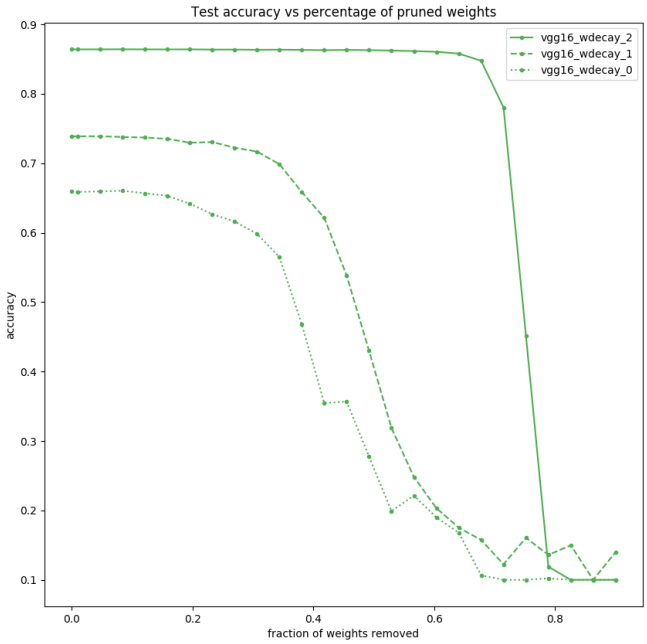


FIGURE 7.16 – **Évolution du *test accuracy* en fonction de la fraction des poids enlevés pour les modèles *weight decay*.**

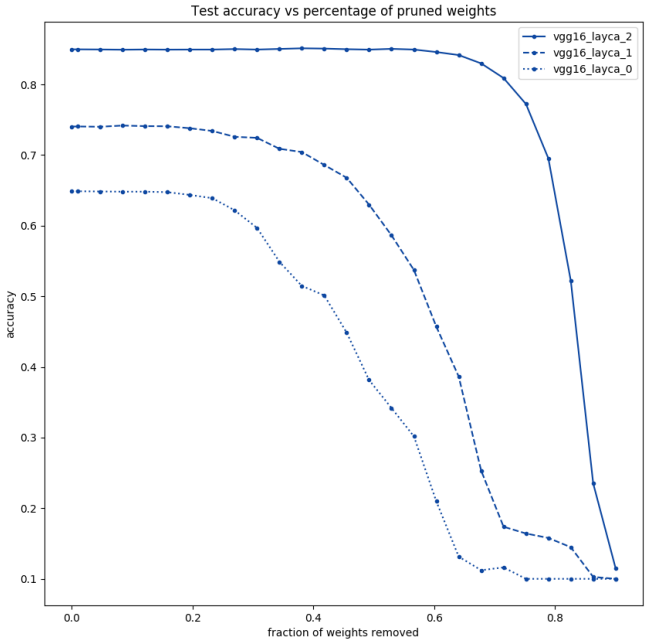


FIGURE 7.17 – **Évolution du *test accuracy* en fonction de la fraction des poids enlevés pour les modèles *layca*.**

Premièrement, on observe que l'effet de l'amputation est moins importante que dans le cas aléatoire. Ces résultats confirment que ce sont principalement les liens de grandes tailles qui définissent la sortie du réseau.

On voit aussi que les modèles *sgd* sont plus affectés par l'élagage que les autres optimiseurs, avec une robustesse légèrement plus forte pour *layca* que *weight decay*. Ceci confirment les résultats de la section précédente.

Pour *weight decay* et *layca*, la majorité des poids de chaque couche sont des poids de faibles amplitudes, proches de zéro. En les enlevant, la structure du réseau a peu changée, permettant ainsi de garder une bonne généralisation.

À l'opposé, le réseau crée par *sgd* est moins élaguée et possède une moins grande proportion de poids de faible amplitude. Retirer des poids du réseau a donc beaucoup plus d'impact sur la sortie parce que ces poids affectent davantage l'activation des neurones du réseau.

7.4 La cosinus distance, un signe de généralisation ?

Plusieurs observations ont été faites au cours de ce chapitre. La première est que la cosinus distance d'une couche du réseau est fonction 1) de l'augmentation de la norme du vecteur poids de la couche et 2) de la quantité de poids dont la variation est de signe contraire au signe initial. Nous avons ensuite montré que les poids des modèles *weight decay* et *layca* subissaient un plus grand nombre de ces variations, expliqué par la mise à zéro d'une majorité de leurs poids.

Nous avons ensuite montrés que ce même mécanisme rendait *weight decay* et *layca* plus robustes au *pruning* que *sgd*, confirmant les résultats obtenus par [AS19].

Nous concluons donc qu'une grande cosinus distance sur l'ensemble des couches n'est pas nécessaire pour bien généraliser mais est signe d'une meilleure aptitude à mettre à zéro les liens inutiles, amenant ainsi à des modèles dont la structure est plus épurée (libre de poids parasites), donc plus robuste au *pruning* et mieux choisie pour la classification voulue.

Ces structures ont donc un meilleur potentiel de généralisation et seraient peut être plus aptes à être ré-entraînés ([Zho+19]) avec les mêmes poids initiaux sur des échantillons plus "difficiles" ou des *batch* plus petits ([KN19]).

8

Activations neuronales

Sommaire

8.1 Introduction

Un deuxième grand champ de recherche de ce mémoire est de jeter un oeil sur l'activation neuronale d'un réseau de neurones entraîné. Après avoir analysé les poids dans le chapitre précédent, le présent chapitre à pour but de trouver des comportements ou des caractéristiques dans les activations qui nous informeraient si un modèle généralise mieux qu'un autre ou non.

Les expériences qui suivent ont été motivées par le désir de changer d'approche. De ne plus regarder les poids du réseau mais plutôt la façon dont il réagit, dont il s'active, par rapport aux images entrées.

Pour cette analyse sur l'activation, deux pistes ont principalement été étudiées. La première est de regarder une paire de neurones de la même couche, afin de voir l'*information mutuelle* de cette paire. Cette information mutuelle nous informera de la potentiel diversité et la redondance d'information véhiculée par le réseau. Effectivement, [ZF14] et [PY17] nous incitent à croire à l'importance qu'a le réseau à détecter un nombre élevé de caractéristiques différentes sur l'image entrée pour qu'il ait de bonnes performances. Pour appuyer nos résultats, nous regarderons aussi les neurones dits pathologiques, ainsi que l'information mutuelle entre les activations et les labels.

Motivé par le papier [CV18a], la seconde piste est de regarder le *coefficient de silhouette* des entrées des neurones de chaque couche afin de regarder plus attentivement le travail de séparation effectué par un neurone. Dans une optique dite de 'divide and conquer', on analysera les liens potentiels entre la clusterisation individuelle de chaque neurone et la bonne classification du réseau tout entier.

Pour procéder, nous sélectionnerons des images pour former un échantillon. Ensuite, cet échantillon sera donné en entré au réseau de neurones et nous regarderons l'activation qui en découle selon différentes méthodes et critères.

Pour la curiosité du lecteur et afin que celui se fasse une idée du comportement du réseau, la distribution des activations de l'échantillon par couche se trouve en Annexe (de A.1 à A.10).

8.2 Paramètres des expériences

Nous discuterons dans cette section du choix des paramètres expérimentaux.

Pour la création de l'échantillon, on prend aléatoirement n images des données d'entraînement.

Dans notre cas, nous prenons $n = 1000$. C'est $1/50e$ du nombre total d'images d'entraînement disponible ($N = 50000$). Ce nombre n a été choisi expérimentalement pour être un bon

compromis entre précision dans les résultats et temps d'exécution. Le principe est d'avoir une certaine reproductibilité des résultats, avec les mêmes tendances et le moins de fluctuation possible, en un temps raisonnable.

Effectivement, si n est choisi trop petit, l'échantillon n'est pas vraiment représentatif de l'ensemble du spectre d'activation. Les résultats seront alors trop bruités et auront un manque d'information à nous donner.

Mais si n est choisi trop grand, le temps d'exécution des expériences est trop long et la mémoire risque même de saturer.

Pour l'expérience suivante, le choix du nombre de 10000 paires de neurones pourrait paraître obsolète pour les couches comportant trop de neurones. En effet, les premières couches comportent 65536 neurones et ont donc un nombre énorme de possibilités de paires (C^2_{65536}). Pourtant avec 10000 paires, la variance des résultats est très faible. Et si l'on augmente le nombre de paires, la tendance des résultats reste la même. Nous jugerons donc l'information offerte par l'échantillon sur 10000 paires de neurones comme suffisamment représentative de la population.

En ce qui concerne le nombre de neurones, le nombre de répétition d'expérience, etc, choisis pour les autres expériences, nous procéderons de la même façon ; nous choisirons les paramètres afin d'avoir un équilibre entre une charge de calcul raisonnable et des résultats suffisamment représentatifs.

8.3 Information mutuelle entre paires de neurones

Dans cette première section, nous choisirons donc des paires de neurones par couche et analyserons leur information mutuelle permettant d'échelonner les résultats entre 0 (aucune information mutuelle) et 1 (corrélation parfaite). Plus celle-ci sera élevée, plus il y aura de redondance d'information, c'est à dire plus la connaissance d'une sortie d'un neurone nous donnera de l'information sur la sortie de l'autre neurone. On regarde donc si des paires de neurones d'une même couche nous donne beaucoup d'informations différentes.

Tout d'abord, rappelons nous ce qu'il y a à savoir sur le concept d'information mutuelle.

8.3.1 Information mutuelle

L'information mutuelle de deux variables aléatoires est une quantité mesurant la dépendance statistique de ces variables [Wik19d]. L'information mutuelle d'un couple (X, Y) de variables

représente leur degré de dépendance *au sens probabiliste*. L'information mutuelle est nulle si et seulement si les variables sont indépendantes, et croit lorsque la dépendance augmente. (Informellement, on dit que deux variables sont *indépendantes* si la réalisation de l'une n'apporte aucune information sur la réalisation de l'autre.)

Définition

Soit (X, Y) un couple de variables aléatoires de densité de probabilité jointe données par $P(X = x, Y = y)$. Notons les distributions marginales $P(X = x)$ et $P(Y = y)$. Alors l'information mutuelle est dans le cas discret :

$$I(X;Y) = \sum_{x,y} P(x,y) \log_2 \frac{P(x,y)}{P(x)\,P(y)}$$

Dans notre cas, l'information mutuelle normalisée va être calculé via les activations d'un premier neurone avec les activations d'un second, représentés respectivement par les vecteurs binaires X et Y.

Entropie

L'entropie (de Shannon) nous donne la quantité d'information contenue dans un signal. Elle nous sera utile pour normaliser l'information mutuelle.

La définition mathématique de l'entropie d'un vecteur X est :

$$H(X) = -\sum_{i=0}^{N} P(x = E_i) \log_2 \Big(P(x = E_i) \Big)$$

où $P(x = E_i)$ est la probabilité de trouver l'élément E_i dans ce vecteur X, sans donner d'importance à l'ordre, et N est le nombre d'éléments.

Dans notre cas binaire, $N = 1$. L'entropie sera alors comprise entre 0, si le vecteur des activations X est tout le temps activé ou inactivé, et 1, si il y a le même nombre d'éléments activés et inactivés dans le vecteur X.

Définition en fonction de l'entropie

L'information mutuelle mesure la quantité d'information apportée en moyenne par une réalisation de X sur les probabilités de réalisation de Y. En considérant qu'une distribution de probabilité représente notre connaissance sur un phénomène aléatoire, on mesure l'absence d'information par l'entropie de cette distribution. En ces termes, l'information mutuelle s'exprime par :

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y)$$

où H(X) et H(Y) sont des entropies, H(X|Y) et H(Y|X) sont des entropies conditionnelles, et H(Y, X) est l'entropie conjointe entre X et Y.

Une représentation visuel de ce concept se trouve sur le figure 8.1.

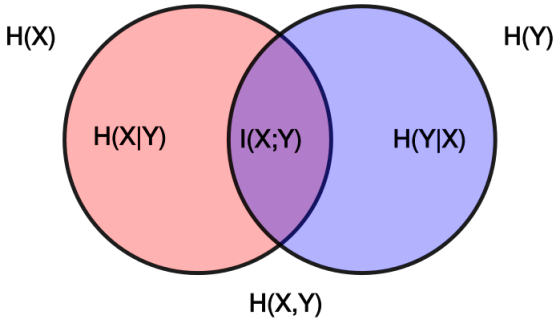


FIGURE 8.1 – Diagramme de Venn montrant les relations additives et soustractives des mesures de l'information associée aux variables corrélées X and Y. L'aire contenue par les 2 cercles est l'entropie jointe $H(X,Y)$. Le cercle de gauche (rouge et violet) est l'entropie individuelle $H(X)$, avec la partie rouge l'entropie conditionnelle $H(X|Y)$. Le cercle de droite (bleu et violet) est $H(Y)$, avec la partie bleue $H(Y|X)$. La partie violette est l'information mutuelle $I(X;Y)$.

Ainsi on voit que $I(X;Y) = 0$ ssi la connaissance des réalisations de X n'apporte aucune information sur les réalisations de Y, et inversement.

Information mutuelle normalisée

Nous rappellerons finalement la formule de l'information mutuelle normalisée :

$$\text{IM Normalisée} = \frac{2I(X;Y)}{H(X) + H(Y)}$$

où $H(.)$ est l'entropie et $I(.\,;.)$ est l'information mutuelle [Ano19d].

La IMN est une bonne mesure pour déterminer la qualité du clustering. Elle permet dans notre cas de se détacher des effets néfastes d'entropie lorsqu'il y a une majorité d'activation à 1 ou d'inactivation à 0.

Notons que c'est une mesure externe car elle a besoin des labels de classe, mais que la valeur absolue des labels n'affecte pas l'information mutuelle ; une permutation des valeurs d'étiquette de classe ou de cluster ne modifiera en aucun cas la valeur du score. De plus, cette métrique est en outre symétrique : le fait de changer les vrais labels avec labels prédits renverra la même

valeur de score.

Par la suite, quand nous parlerons d'information mutuelle, elle sera à chaque fois normalisée.

8.3.2 Choix des neurones

Mais quelles paires de neurones vont être les plus adaptés pour le calcul de cette information mutuelle normalisée ?

Dans notre cas, nous choisirons des paires de neurones aléatoirement sur la même couche. Le fait de prendre 2 neurones arbitrairement nous permet de gagner en rigueur et de cerner tous les comportements, tel que de l'information semblable captée par 2 neurones complètement éloignés. Nous avons donc pris l'observation la plus large possible pour ne passer à coté d'aucune éventualité.

8.3.3 Binarisation

Une fois 2 neurones sélectionnés, il faut binariser les activations de ceux-ci afin de pouvoir en calculer l'information mutuelle. C'est à dire que, pour calculer l'information mutuelle des 2 vecteurs, il faut que leurs activations soient labelisées à 0 ou 1. Effectivement, le calcul de l'information mutuel doit se faire avec des libellés discrets.

On remarque que selon le modèle, les paires de neurones ne s'activent pas pour les mêmes seuils. Il nous faut une mesure relative à chaque neurone de chaque modèle.

Nous avons donc opté pour une binarisation relative par rapport à la moyenne de l'activation du neurone. Au dessus de cette moyenne, nous binarisons l'activation à 1, alors qu'en dessous, nous binarisons à 0.

Il n'y a donc pas de problème de normalisation et tous les modèles sont comparables, puisque tout est relatif à l'activation du neurone lui-même.

8.3.4 Les résultats de l'information mutuelle entre 2 neurones

Sur la figure 8.2, on peut voir l'information mutuelle moyenne pour 10000 paires de neurones par couche et 1000 images d'activations.

Pour tous les modèles, on remarque que ces moyennes sont non-nulles au début du réseau. Elles sont légèrement moins importantes quand le réseau généralise bien.

Par contre, vers la fin du réseau c'est l'inverse qui se produit de manière plus conséquente. L'information mutuelle va être plus importante dans les dernières couches si le modèle généralise

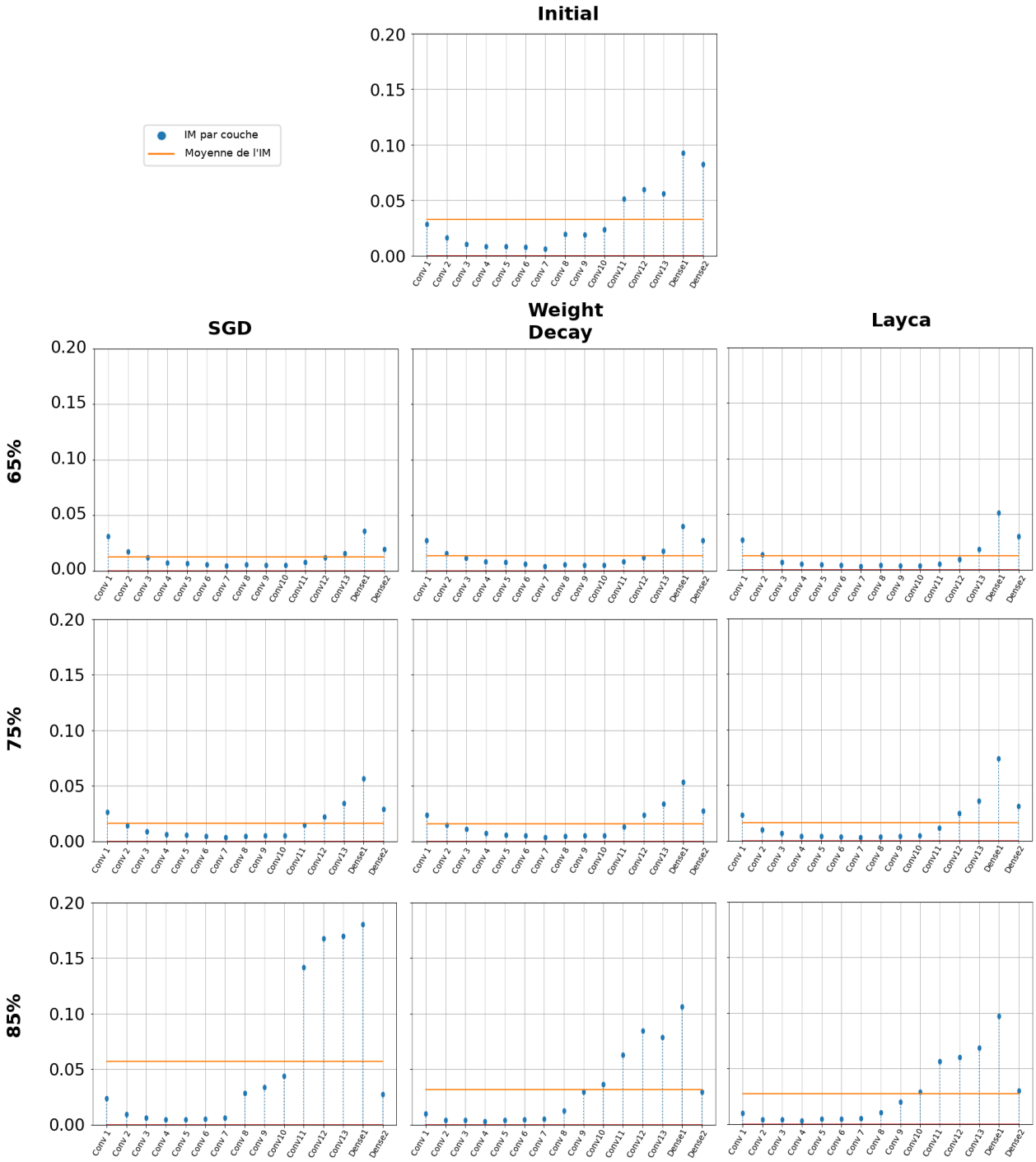


FIGURE 8.2 – Information mutuelle moyenne par couche pour 1000 images d'activation et 10000 paires de neurones

mieux, et cela de plus en plus tôt dans le réseau.

Pour l'information mutuelle du réseau initial (non entraîné), le premier comportement observable est une information mutuelle plus grande que pour les réseaux entraînés dans les premières couches.

On voit également qu'il y a une information mutuelle qui augmente fortement de la moitié

jusqu'à la fin du réseau.

Notons que le comportement des optimiseurs n'est pas drastiquement différent. Il y a pourtant une exception pour SGD 85% dont on remarque une forte augmentation dans les dernières couches (hormis la dernière).

Analyse et histogrammes de l'information mutuelle des paires de neurones

Tout d'abord, un comportement interpellant est celui des moyennes d'information mutuelle non-nulles dans les premières couches.

Pour nous aider à comprendre ce comportement, regardons l'histogramme de l'information mutuelle en annexe (A.11,A.12, A.13 et A.14), où la moyenne est affichée en rouge, et la moyenne ignorant l'information mutuelle valant 1 est représentée en bleu. (Souvent, cette seconde moyenne se superpose avec la moyenne rouge.)

On remarque dans les premières couches que la moyenne (rouge) est légèrement déportée vers la droite car certaines informations mutuelles ne sont pas nulles mais tirent vers 0.2. Cela est en partie dû au caractère convolutif du réseau, qui capte plusieurs fois la même information de l'image.

Ce caractère est le plus fortement visible sur le réseau non entraîné dû au fait qu'il ne travaille pas l'information transmise. Il va juste transmettre l'information de manière bruitée directement à la couche suivante, via ses poids initiaux.

Pour expliquer ce caractère convolutif du réseau, 2 neurones d'une même pair ont une chance non nulle d'avoir une partie de leur entrée qui se chevauchent, apportant donc la même information. Pour la première couche du modèle initial, cette même information de chevauchement représente un ratio de presque 1% ($\frac{9*4+18*4+12*4+6*8+3*4+27*(63/64)}{32*32*(3*3*3)} = 0.008773804$). Pourtant, ce n'est pas suffisant pour justifier la moyenne de plus de 2% (0.02707797) des résultats.

Cette information mutuelle plus élevée que prévu dans les premières couches nous montre la présence du biais de réalité intrinsèquement présent dans les images reçues. Ce biais est l'indicateur qu'un certain ordre est présent, c'est-à-dire que l'information donnée par un pixel est également donnée par un autre pixel. (Par exemple, l'information d'un ciel bleu par un neurone peut également se faire par un autre neurone dont les entrées captent d'autres pixels.)

Ces 2 comportements s'atténuent naturellement avec l'avancement dans le réseau. Plus rapidement pour les réseaux qui généralisent bien.

De plus, ce comportement est moindre dès la première couche d'un réseau qui généralise bien. Cela serait un indicateur d'un meilleur apprentissage individuel des neurones, qui détectent plus d'informations différentes de l'image entrée.

Plus d'informations différentes offrent directement plus de critère au réseau pour une meilleure classification. La classification utilise ainsi plus d'éléments utiles et la généralisation du modèle est alors meilleure.

Ensuite, un deuxième comportement interpellant se produit ; plus un modèle généralise bien, plus l'information mutuelle sera importante dans les dernières couches, et cela de plus en plus tôt dans les couches.

En regardant à nouveau les histogrammes de l'information mutuelle, on voit bien que plus le réseau généralise bien, plus le moyenne d'information mutuelle (en rouge) sera élevée. Ce comportement concerne toutes les dernières couches sauf la dernière.

On explique cela par le fait qu'à partir d'un certain moment dans le réseau, il y a un recoupement de l'information. Si l'information se recoupe plus tôt c'est parce que le réseau a pu séparer plus tôt des caractéristiques semblables de l'image, indiquant que les neurones du réseau ont déjà fait un premier travail de préclassification partiel plus important.

Pour le réseau initial, l'augmentation de son information mutuelle moyenne est principalement due à la présence de paires de neurones ayant une information mutuelle valant 1. La moyenne rouge va donc augmenter. Alors que la moyenne bleue (ignorant l'information mutuelle valant 1) reste proche de 0.

Cela est du en grande partie au caractère du réseau qui n'a pas encore appris. Nous verrons dans la sous-section suivante (8.3.5) que l'augmentation de la présence de paires de neurones ayant une information mutuelle égale à 1 est principalement due à la présence croissante de *neurones pathologiques*. Nous définirons ici un neurone pathologique comme un neurone dont les activations en sortie sont toujours 0 ou 1. Ces neurones ont des répercussions dans les couches suivantes, augmentant ainsi le nombre de neurones pathologiques et l'information mutuelle des prochaines couches.

Avant d'analyser les neurones pathologiques, nous remarquerons que sur les activations de la dernière couche dense, l'information mutuelle revient vers 0 pour les réseaux qui ont appris, et vers 0.09 pour le réseau initial. D'autre part, pour l'avant-dernière couche dense, l'information mutuelle est plus élevée quand le réseau généralise mieux. Ces critères pourraient constituer les symptômes d'une bonne généralisation.

8.3.5 Neurones pathologiques

Nous sélectionnons les paires de neurones en prenant un neurone au hasard, et puis un second également pris au hasard. Le nombre de paires pathologiques P est donc lié par une relation de type $P = p^2$, où p est le nombre de neurones pathologiques dans le réseau. Par exemple, on a environ 81/10000 paires pathologiques pour 9/100 neurones pathologiques. Cette relation prouve la cohérence de nos résultats.

Il est intéressant de dire que les paires pathologiques sont uniquement des paires toujours 'inactivées'. Un exemple est donné sur la figure 8.3. Dans le graphe de gauche, on observe le ratio de neurones pathologiques, alors que dans le graphe de droite on observe une occurrence du nombre de paires pathologiques. Sur cet exemple, on remarque sans surprise que nous avons toujours des paires pathologiques inactivées (0,0) vu que les neurones pathologiques renvoient toujours 0. On observe aussi la relation expliquée ci-dessus.

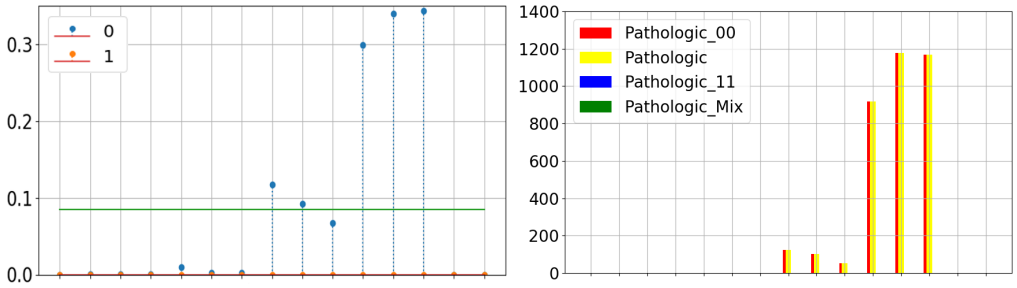


FIGURE 8.3 – Comparaison entre les neurones pathologiques et les paires pathologiques pour le modèle sgd 85%. L'axe horizontal représente l'aval des couches. Sur le graphe de gauche, on observe le ratio de neurones pathologiques renvoyant tout le temps 1 ou 0. On observe la moyenne en vert. Sur le graphe de droite, on peut observer le nombre de paires pathologiques. En jaune le nombre total, en rouge les paires toujours inactives, en bleu celles toujours actives et en vert les paires mixtes. On remarquera l'absence de bâtonnets bleus ou verts.

On peut voir le graphe du nombre de neurones pathologiques sur la figure 8.4. Ces neurones pathologiques ont clairement un impact sur le comportement de la moyenne d'information mutuelle. Dans les dernières couches, on voit que lorsqu'il y a une augmentation du nombre de neurones pathologiques, où il y a aussi une augmentation de l'information mutuelle. Effectivement, des paires de neurones qui renvoient toujours la même chose auront une information mutuelle de 1.

Cela explique pourquoi il y a une différence entre les moyennes rouge et bleue sur les histogrammes de l'information mutuelle et l'augmentation de l'information mutuelle dans le réseau initial, mais cela n'explique cependant pas pourquoi l'information mutuelle augmente

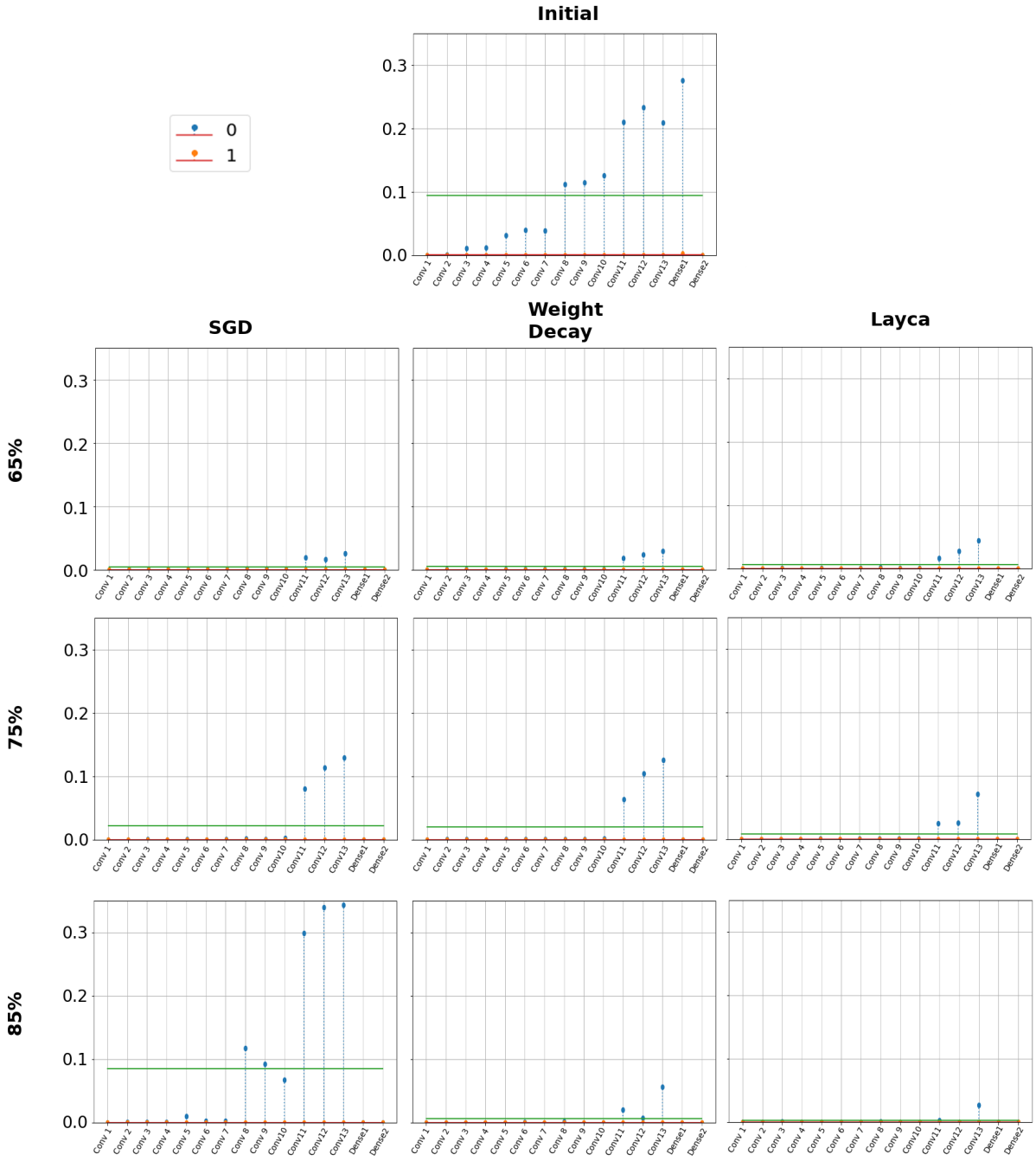


FIGURE 8.4 – Nombre de neurones pathologiques par couche pour 1000 images d'activation

dans les dernières couches dans les réseaux entraînes.

Cette augmentation pourrait être due à un changement d'entropie, voilà pourquoi nous avons investigué cette piste.

8.3.6 L'entropie

Le concept d'entropie a été expliqué plus tôt dans la sous-section "Information mutuelle".

On s'interroge sur l'influence de l'entropie sur notre information mutuelle normalisée. C'est-à-dire, est-ce qu'un nombre plus élevé de labels activés ou non-activés amènerait les changements d'information mutuelle visibles dans les résultats précédents ?

On veut donc confirmer que l'impacte d'un forte différence d'entropie, si il y a beaucoup plus présence de 1 ou de 0 dans les activations, n'aura pas trop d'influence sur l'information mutuelle de la pair.

Afin de nous en assurer, nous avons alors observé le ratio de zéros dans les activations. On peut voir ces résultats dans le tableau 8.1. Ils représentent la proportion moyenne des inactivations par couche.

Model	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	Conv 6	Conv 7	Conv 8	Conv 9	Conv 10	Conv 11	Conv 12	Conv 13	Dense 1	Dense 2
Initial	0.65	0.69	0.72	0.73	0.71	0.72	0.73	0.73	0.74	0.73	0.75	0.76	0.74	0.75	0.50
Sgd 0	0.65	0.66	0.66	0.66	0.67	0.67	0.67	0.68	0.68	0.69	0.71	0.72	0.72	0.66	0.89
Sgd 1	0.66	0.66	0.65	0.66	0.67	0.67	0.67	0.68	0.68	0.69	0.73	0.74	0.76	0.67	0.90
Sgd 2	0.67	0.66	0.65	0.67	0.67	0.67	0.67	0.71	0.71	0.70	0.74	0.74	0.75	0.63	0.90
Wd 0	0.65	0.67	0.66	0.66	0.67	0.67	0.67	0.68	0.68	0.69	0.71	0.72	0.73	0.67	0.90
Wd 1	0.66	0.66	0.65	0.66	0.67	0.67	0.67	0.68	0.68	0.69	0.72	0.74	0.75	0.67	0.90
Wd 2	0.68	0.67	0.62	0.67	0.66	0.67	0.68	0.68	0.67	0.69	0.68	0.67	0.72	0.75	0.90
Layca 0	0.65	0.67	0.66	0.67	0.67	0.67	0.67	0.68	0.68	0.68	0.71	0.71	0.73	0.67	0.90
Layca 1	0.65	0.66	0.66	0.66	0.67	0.67	0.67	0.67	0.67	0.68	0.70	0.71	0.75	0.68	0.90
Layca 2	0.67	0.67	0.62	0.67	0.66	0.66	0.67	0.67	0.68	0.69	0.67	0.67	0.72	0.75	0.90

TABLE 8.1 – Ratio du nombre de zéros dans les activations par modèle et par couche

Dans ce tableau, il y a une faible augmentation du nombre de zéros dans les inactivations au fur et à mesure qu'on avance dans le réseau. Cette augmentation s'arrête aux couches denses, dont la dernière est représentative de l'apprentissage (un ratio de 0.9 de 0 pour les réseaux entraînés correspondant à 1 classe sur 10 activée, ou un ratio de 0.5 de 0 représentant le caractère aléatoire du réseau initial dont les neurones finaux renvoient 0 avec une probabilité de 0.5).

Mais il faut remarquer que l'entropie ne varie pas significativement pour changer autant l'information mutuelle.

Ce tableau est cependant remplis d'informations dont la plupart sont sur-lignées en couleurs. Tout d'abord, pour le réseau initial, on observe environ 70% d'inactivations assez rapidement dans les couches. Pour Layca et Weight Decay, on observe que ce seuil des 70% est atteint plus tard quand la généralisation est meilleure, à l'inverse d'SGD où ce seuil est atteint plus tôt quand la généralisation est meilleure. On remarque également que Layca et Weight Decay ont en général un comportement plus semblable, à l'inverse de SGD. Nous noterons un creux dans la 3ème couche, quand ça généralise mieux chez Weight Decay et Layca, alors qu'un creux est présent dans la première couche dense chez SGD.

On remarque la même allure entre ce tableau et les graphes des neurones pathologiques.

Avec ces pourcentages relativement constant, on ferme la porte à l'hypothèse disant que l'entropie influencerait significativement l'augmentation de la MI. On est donc en présence d'un changement d'information mutuelle, non drastiquement influencée par un changement d'entropie du à l'augmentation du nombre de zéros ou de uns dans les activations.

(Rappelons ici que cette faible variation d'entropie est prise en considération par le caractère normalisé de l'information mutuelle [con19]).

La variation de ces moyennes d'information mutuelle est donc principalement due à un autre phénomène.

8.3.7 Information mutuelle avec les labels

Pour compléter l'expérience sur l'information mutuelle de paires de neurones, on a voulu situer notre information mutuelle de paires de neurones avec l'information mutuelle d'un neurone avec les labels. Si il n'y a pas de lien, le réseau crée ses propres sous-classes indépendamment de la classification. Au contraire, si il y a un lien, c'est que le réseau agence et crée des sous-classes directement en lien avec les labels de la classification voulue.

On peut donc voir sur la figure 8.5 l'information mutuelle entre l'activation d'un neurone et les labels associés à cette activation.

Premièrement, remarquons que pour le réseau initial il y a une corrélation quasi-nulle entre l'information mutuelle avec les labels et celle avec les paires de neurones. On fera l'hypothèse que cette très faible corrélation proviendrait du bruit et du caractère aléatoire de la création des poids du réseau.

Pour les réseaux entraînés, il y a une augmentation dans les dernières couches du réseaux. Cette augmentation est également plus importante lorsque le modèle généralise mieux.

On voit que l'allure de l'information mutuelle des paires (figure 8.2) est semblable avec l'information mutuelle avec les labels (figure 8.5), si on ignore l'influence des neurones pathologiques et le comportement des premières couches.

De plus, on observe que tout comme l'information mutuelle des paires de neurones, l'information mutuelle avec les labels augmentera de plus en plus tôt si le modèle généralise bien. Cette augmentation est plus nette dans le cas de l'information mutuelle avec les labels.

Pour un réseau à 65%, cela démarre vers la couche convolutionnelle 12 ou 13. Pour un réseau à 75%, cela démarre vers la couche convolutionnelle 10 ou 11. Pour un réseau à 85%,

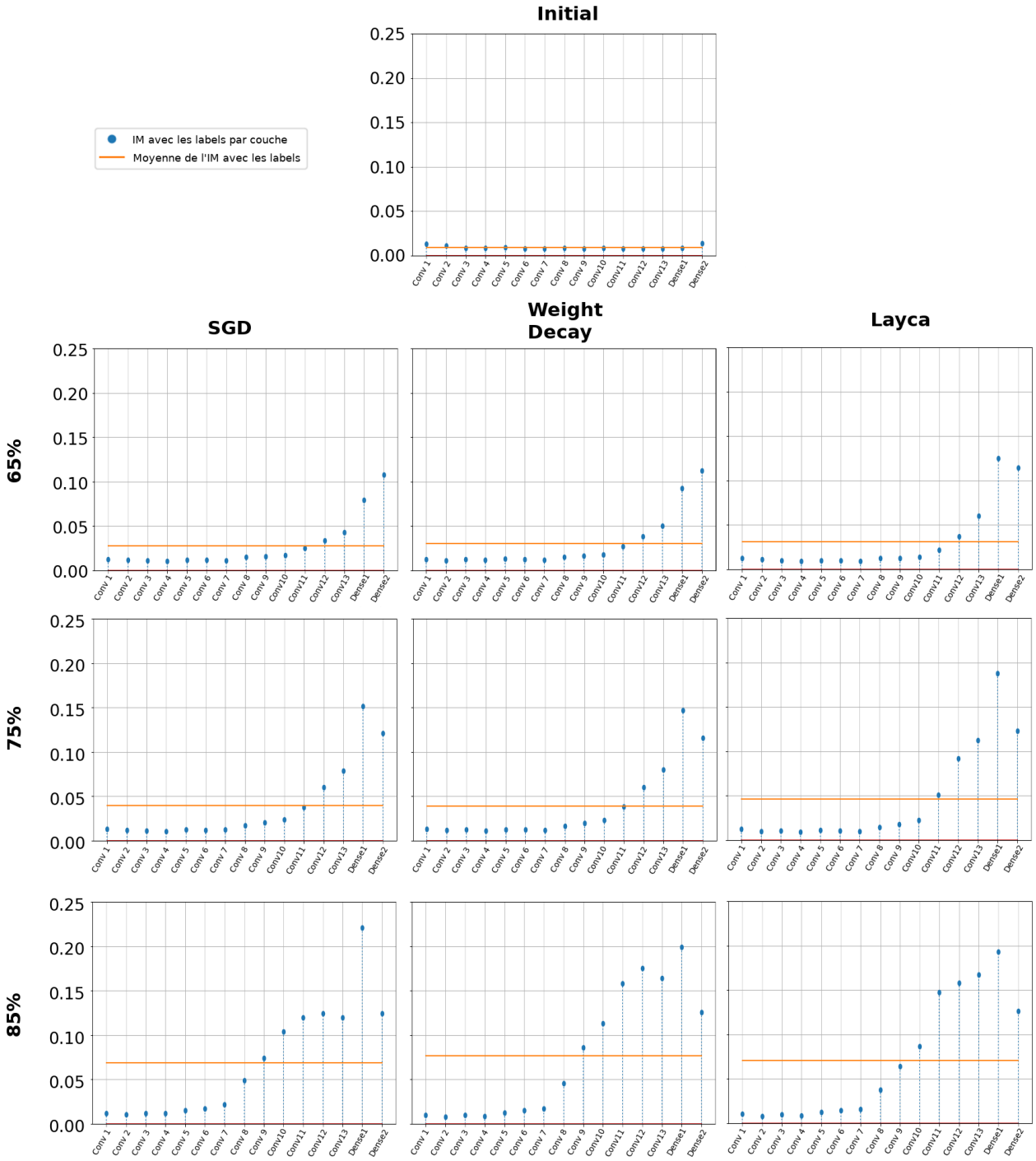


FIGURE 8.5 – Mutual information avec les labels par couche pour 1000 images d'activation et 10000 neurones

cela démarre vers la couche convolutionnelle 7 ou 8.

Il y a donc lien de dépendance entre ces 2 informations mutuelles. De plus, elles mettent

en avant la capacité du réseau à discerner des sous-classes en liens avec les classes recherchées. Ces sous-classes semblent être reconnues de plus en plus tôt par les neurones quand le modèle généralise mieux.

Afin d'investiguer plus précisément ce travail individuel de pré-classification en sous-classes par les neurones, nous passons maintenant à une toute autre façon de voir les activations dans le réseau en regardant de plus près la classification des clusters faite par chaque neurone.

8.4 Expérience sur le clustering

Dans cette section, nous verrons une deuxième approche qui se penchera plus précisément sur la manière dont les neurones du réseau classifient leur entrées en clusters.

Notre expérience a donc été motivée par une optique de '*divide and conquer*' ; on suppose que la manière dont les neurones vont classer individuellement leur entrées aura de l'influence sur la classification globale du réseau. Nous allons donc regarder les 2 clusters formés par le neurone : les entrées activées et non-activées. Et plus particulièrement nous allons voir la densité de ces clusters.

L'expérience est entre autres motivée par l'interrogation de savoir si les classifications des neurones sont robustes ou non. C'est à dire, est-ce que les autres activations autour d'une activation de référence sont-elles proches de cette dernière ? Les entrées activées sont elles denses entre elles ?

Si le cluster activé est dense, on suppose qu'une activation proche sera également activée. Si beaucoup d'éléments non-activés sont proches d'un cluster activé, on suppose qu'il y aura une mauvaise classification trop bruitée, et donc une mauvaise généralisation.

Il s'agit en fait de voir ici si il y a une cohérence.

8.4.1 Pondération

La figure 8.6 va fortement appuyer notre explication.

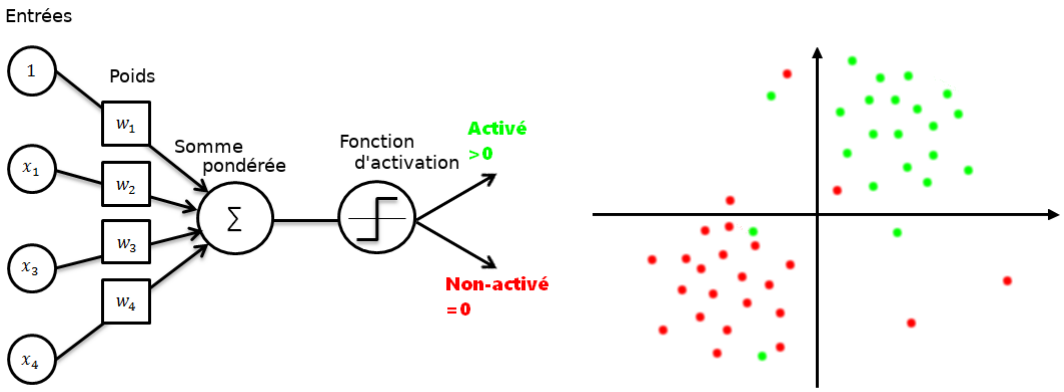


FIGURE 8.6 – Représentation conceptuelle en 2 dimensions des entrées clustérisées par le neurone

On peut donc voir les sorties de la couche précédente qui vont servir d'entrées au neurone. Celui-ci va les pondérer et les sommer afin de passer le résultat de la somme par une fonction d'activation ReLu. Si le résultat est non nul, on considère l'entrée comme activée. Sinon, on la considère comme non-activée. Nous aurons alors 2 clusters : les activés et les non-activés. Par la suite, nous allons mesurer la qualité de ces 2 clusters à être bien départagés.

Mais le problème est que les entrées ne vont pas toutes avoir le même impacte dans la somme. En effet, une entrée qui sera pondérée par un poids faible aura peu d'impact sur le fait que le neurone soit activé ou non. A l'inverse, une entrée fortement pondérée sera plus déterminante à l'activation.

Nous adapterons donc chaque valeur du vecteur des entrées en multipliant la différence (distance euclidienne) ou le produit (cosine distance) des composantes de ces vecteurs par l'amplitude du poids du lien correspondant.

8.4.2 Distance

Parmi toutes les distances possibles, nous en avons sélectionnées deux afin de nous offrir des approches différentes pour nos résultats : la distance euclidienne et la cosinus distance.

Une illustration comparative de ces distances se trouve sur la figure 8.7 [Ano16].

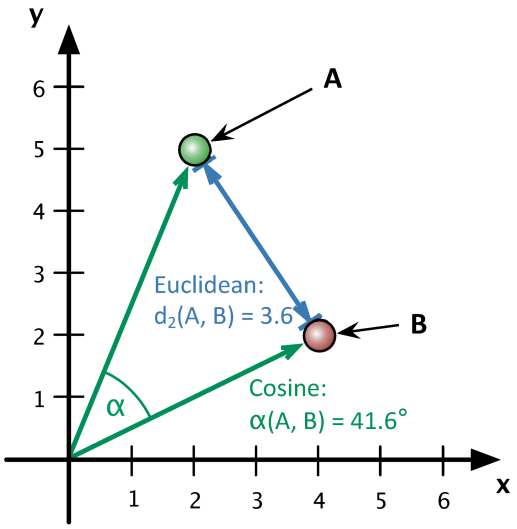


FIGURE 8.7 – Comparaison entre la distance Euclidienne et la similarité cosinus. Cette dernière est directement liée à la cosinus distance.

8.4.3 Distance Euclidienne

La distance Euclidienne, aussi appelée 2-distance, est une distance géométrique usuelle dans un espace multidimensionnel. Elle est définie par

$$\text{Distance Euclidienne(X,Y)} = \sqrt{\sum_i (x_i - y_i)^2}$$

où l'on prend la racine carré de la somme des différences des entrées des vecteurs X et Y.

8.4.4 Cosinus distance

La cosinus distance est une autre façon de calculer la distance entre 2 vecteurs. Sa formule mathématique est donnée par

$$\text{Cosinus Distance(X,Y)} = 1 - \frac{X \cdot Y}{||X|| \times ||Y||} = \frac{\sum_i x_i.y_i}{\sqrt{\sum_i x_i^2}.\sqrt{\sum_i y_i^2}}$$

Cette distance n'est donc pas influencée par l'amplitude des vecteurs.

La mesure de cosinus distance a été utilisé dans [CV18b] pour quantifier la distance angulaire entre les vecteurs de poids initiaux et finaux des couches du réseau. Nous utilisons également cette métrique de façon à comparer les courbes de cosinus distance de nos modèles entre elles et avec celles que l'on trouve dans [CV18b].

En deux dimensions, la cosinus distance est égale à un lorsque les deux vecteurs sont orthogonaux. Il faut cependant noter que les vecteurs des entrées des neurones de nos réseaux

résident en de très grande dimensions. L'intuition de cette distance tombe alors rapidement par la difficulté visualisation.

Notons aussi que nous seront donc également sujet au phénomène dit "Fléau de la dimension", ou *Curse of dimensionality*.

Cependant, la cosinus distance nous offrant des résultats plus riches, c'est celle-ci que nous présenterons par la suite.

8.4.5 Coefficient de Silhouette

Le coefficient de silhouette, ou *silhouette score*, est une méthode d'interprétation des clusters dans un échantillon de données [Wik19i]. Cette technique fourni une représentation de la qualité de la classification de chaque objet.

La valeur de silhouette est une mesure de la similarité d'un objet avec son propre cluster (cohésion) par rapport aux autres clusters (séparation). Le coefficient de silhouette va de -1 à +1, où une valeur élevée indique que l'objet est bien adapté à son propre cluster et mal adapté aux clusters voisins. Si la plupart des objets ont une valeur élevée, la configuration du clustering est appropriée. Si de nombreux points ont une valeur basse ou négative, la configuration du cluster peut comporter trop ou trop peu de clusters.

Le coefficient de silhouette peut être calculée avec n'importe quelle métrique de distance, telles les distances euclidienne et cosinus.

Elle se calcule comme suit : On imagine que les données ont été classées en k clusters. Dans notre cas $k = 2$, activés/non-activés. Pour chaque donnée/point i du cluster C_i, calculons la distance moyenne de ce point par rapport à tous les autres points du même cluster :

$$a_i = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$

où $d(i, j)$ est la distance entre i et j dans le cluster C_i. (On divise par $|C_i| - 1$ car on n'inclut pas la distance $d(i, i)$ dans la somme.) On peut interpréter a_i comme la mesure de la bonne classification du point i dans le cluster C_i. Plus la valeur est petite, plus le point est bien assigné.

Nous définissons ensuite la dissimilarité moyenne du point i en un groupe c comme la moyenne de la distance de i à tous les points de c. Pour chaque point $i \in C_i$, on défini :

$$b_i = \min_{i \neq j} \frac{1}{|C_j|} \sum_{j \in C_j} d(i, j)$$

la plus petite distance moyenne de i à tous les points de tout autre cluster, dont i n'est pas

membre. Le cluster avec cette dissimilarité moyenne la plus petite est dite "le cluster voisin" de i car c'est le cluster suivant le mieux ajusté pour le point i.

Nous définissons maintenant le coefficient de silhouette (une valeur) à un point de données i par

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, \text{ si } |C_i| > 1$$

et

$$s(i) = 0, \text{ si } |C_i| = 1$$

On peut donc écrire

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

Et il est alors évident que $-1 \leq s(i) \leq 1$.

Notez également que le score est égal à 0 pour les clusters de taille = 1. Cette contrainte est ajoutée pour empêcher le nombre de clusters d'augmenter de manière significative.

Pour que $s(i)$ soit proche de 1, nous avons besoin d'un $a(i)$ vraiment plus petit que $b(i)$. Comme $a(i)$ est une mesure de la différence de i par rapport à son propre cluster, une petite valeur signifie qu'il est bien adapté. De plus, un grand $b(i)$ implique que i est mal adapté à son cluster voisin. Ainsi, un $s(i)$ proche de 1 signifie que les données sont correctement groupées. Si $s(i)$ est proche de -1, alors, selon la même logique, i serait plus approprié s'il était groupé avec son cluster voisin. Un $s(i)$ proche de zéro signifie que la donnée est à la frontière de deux grappes naturelles.

Une description analogue de ce coefficient se trouve dans [G11]. Notons qu'il existe également d'autre méthodes pour décrire et mesurer la performance d'une classification en clusters.

8.4.6 Les résultats du coefficient de silhouette

Nous allons donc mesurer la qualité de la classification des entrées des neurones en 2 clusters, activés et non-activés, via le coefficient de silhouette.

Pour calculer nos valeurs de coefficient de silhouette, nous prendrons une moyenne sur les 10% des entrés activés les plus proches entre elles. Pour chacune de ces entrées, nous sélectionnerons les 20% des entrées les plus proches autour de celles-ci (activées et non-activées). C'est avec cette sélection que nous calculons finalement le coefficient de silhouette.

Nous répéterons ensuite cette expérience pour 20 neurones de chaque couche afin d'en tirer

une moyenne qui nous donnera une représentation pour la couche.

Voici les résultats des coefficients de silhouettes pour la distance euclidienne sur la figure 8.8, et ceux pour la cosinus distance sur la figure 8.9. Ils ont été calculé pour un échantillon de 1000 images. Il y a donc 1000 vecteurs d'entrées classés comme activés et non-activés.

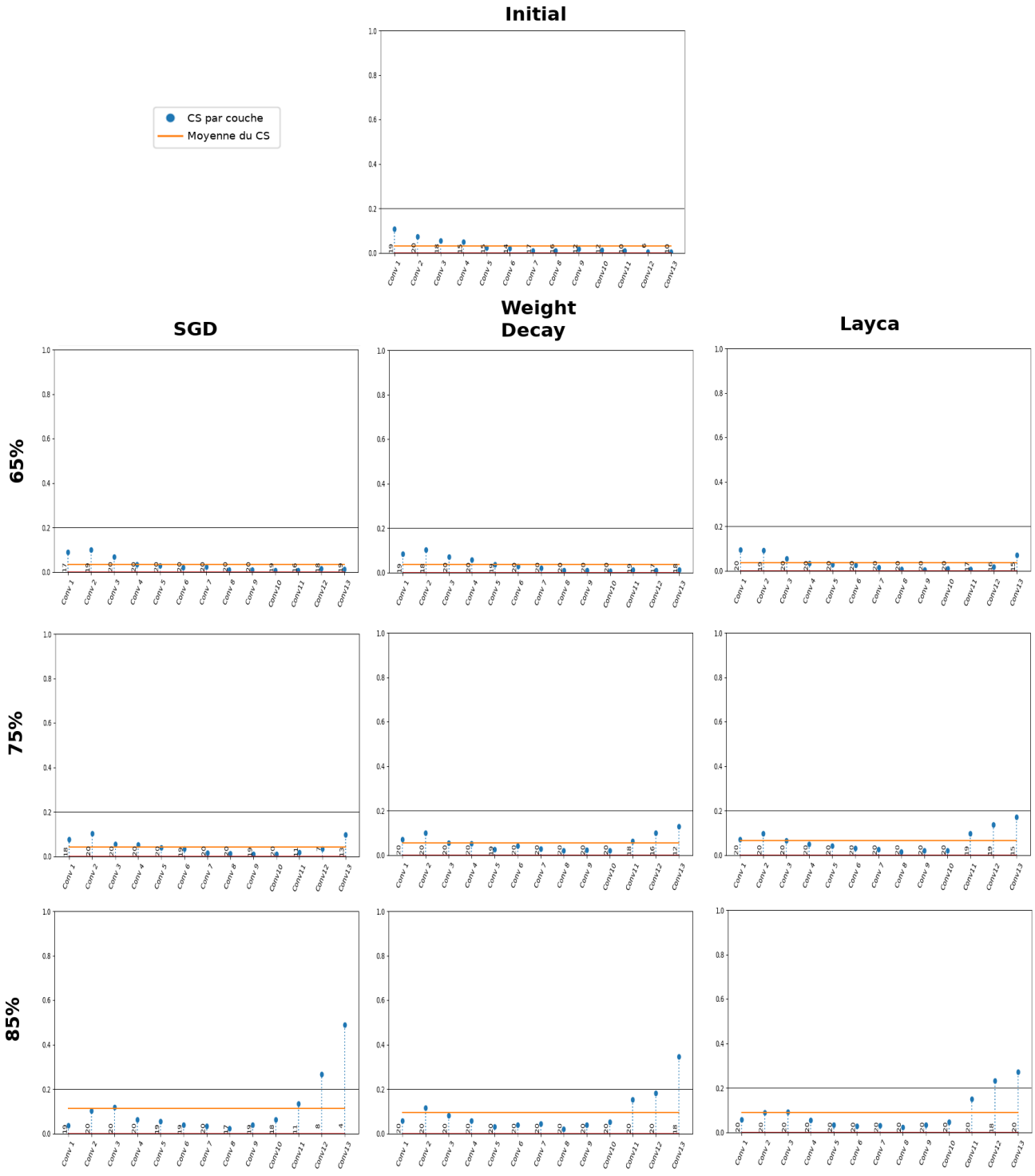


FIGURE 8.8 – Coefficient de silhouette par couche pour Ratio1=0.1, Ratio2=0.2, Dist=Euclidean, 1000 images d'activation et 20 neurones

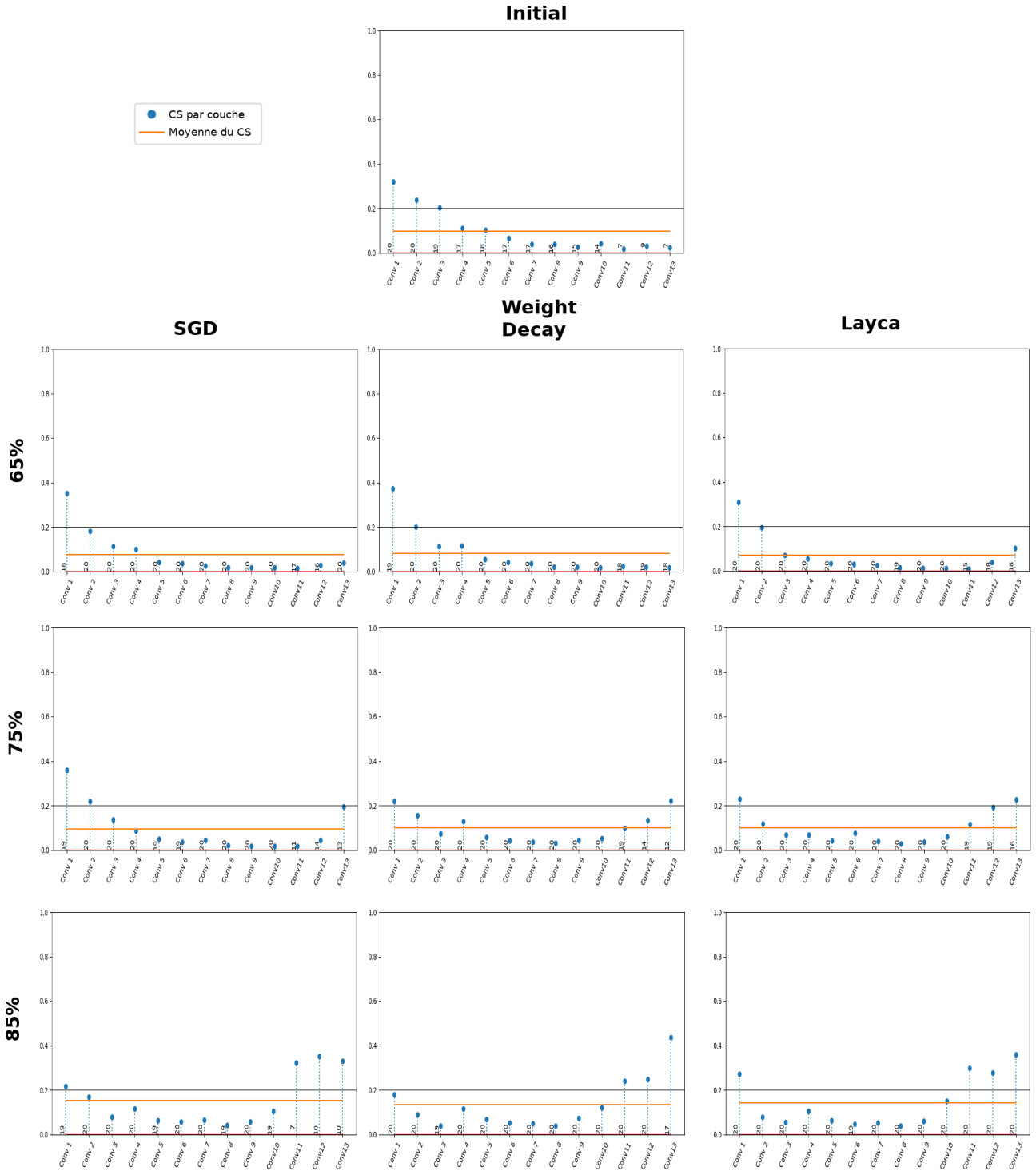


FIGURE 8.9 – Coefficient de silhouette par couche pour Ratio1=0.1, Ratio2=0.2, Dist=Cosine, 1000 images d'activation et 20 neurones

Dans les premières couches, une première observation (plus fortement visible sur la cosinus distance), est la diminution de ce coefficient de silhouette pour un modèle qui généralise mieux.

Notons ici que pour la distance euclidienne, le maximum se situe au niveau de la 2ème ou de la 3ème couche pour les réseaux entraînés. Tandis que pour la cosinus distance, le maximum se situe à la première couche. Et il y a aussi présence d'un second maximum local après la couche

de max-pooling.

Dans une deuxième observation, on remarque que les dernières couches possèdent un coefficient de silhouette plus élevé et plus tôt quand le modèle généralise mieux.

La troisième observation importante est la présence d'un coefficient de silhouette très faible dans les couches centrales du réseau.

Lorsqu'il n'y a qu'un seul type d'entrée autour de l'entrée activée sélectionnée, on ignore le calcul du coefficient de silhouette pour cette entrée. Ce cas de figure arrive essentiellement pour des cas où il n'y a que des entrées activées.

Plus largement, lorsqu'un neurone est tout le temps activé ou non activé, on ignore ce neurone. Sur 20 neurones, le chiffre à la base du résultat est le nombre de neurones pris en considération pour le coefficient.

Ce chiffre est légèrement plus petit pour la première couche. Il baisse plus fortement pour les dernières couches. Pour Weight Decay et Layca, la tendance est que plus le modèle généralise bien, moins ce chiffre diminue. Pour SGD, c'est l'inverse.

De plus, on remarque qu'il y a très peu de neurones ignorés au centre du réseau.

Analyse

Pour un réseau initial, on voit que les neurones des premières couches sont moins sensibles à une petite différence entre les entrées. Des entrées similaires donneront des activations similaires. Il va donc y avoir un coefficient de silhouette plus important entre les activées et les non activées.

La diminution du coefficient de silhouette dans les premières couches quand les modèles généralisent mieux indique que les neurones dédensifient plus rapidement les clusters de leurs entrées. La disposition des entrées activées et non activées est alors plus homogène dans l'espace.

Dans un bon apprentissage des premières couches, le but serait d'observer plusieurs types de comportement plutôt qu'un seul. Un réseau qui apprend mieux aura donc potentiellement des neurones qui font attention à plus de comportements différents dans leurs entrées.

C'est pour cela qu'au niveau des couches de max-pooling il y a une réaugmentation, car on va justement casser et ignorer cette subtilité d'information.

On voit une augmentation du coefficient de silhouette dans les dernières couches. Cela est

un indicateur d'un réseau composé de neurones qui clusterisent de manière plus distincte et dense.

On remarque que là où le coefficient de silhouette commence à monter est *approximativement* lié avec les généralisation, c'est à dire que plus le réseau généralise bien, plus tôt le coefficient de silhouette commence à monter. Pour rappel, les valeurs précises de précision de généralisation sont : SGD(64.27/75.31/83.64), WD(65.91/73.88/86.4) et LAYCA(64.88/74.04/84.97).

Cette augmentation de la densification des clusters serait due à la capacité des neurones à commencer à trouver des sous-classes. Des entrées similaires qui peuvent être activées de manière similaire.

Il y a un creux au niveau des couches centrales. Ce creux dans le coefficient de silhouette proche de la valeur 0 signifie que les entrées sont presque tous sur les bords des clusters. Les clusters n'ont pas de frontières suffisamment distinctes et ne sont pas assez dense. Cela pourrait être interpréter par une incapacité des neurones à pouvoir déjà dégager des sous-classes, c'est à dire des clusters d'activations ou de non-activations dans les entrées lié à la classification finale voulue.

La petite partie des neurones ignorés dans le début du réseau serait du au fait qu'il y a une partie des neurones qui sont tout le temps activés mais à des intensités différentes. Cela serait d'ailleurs la principale raison pour ignorer des neurones.

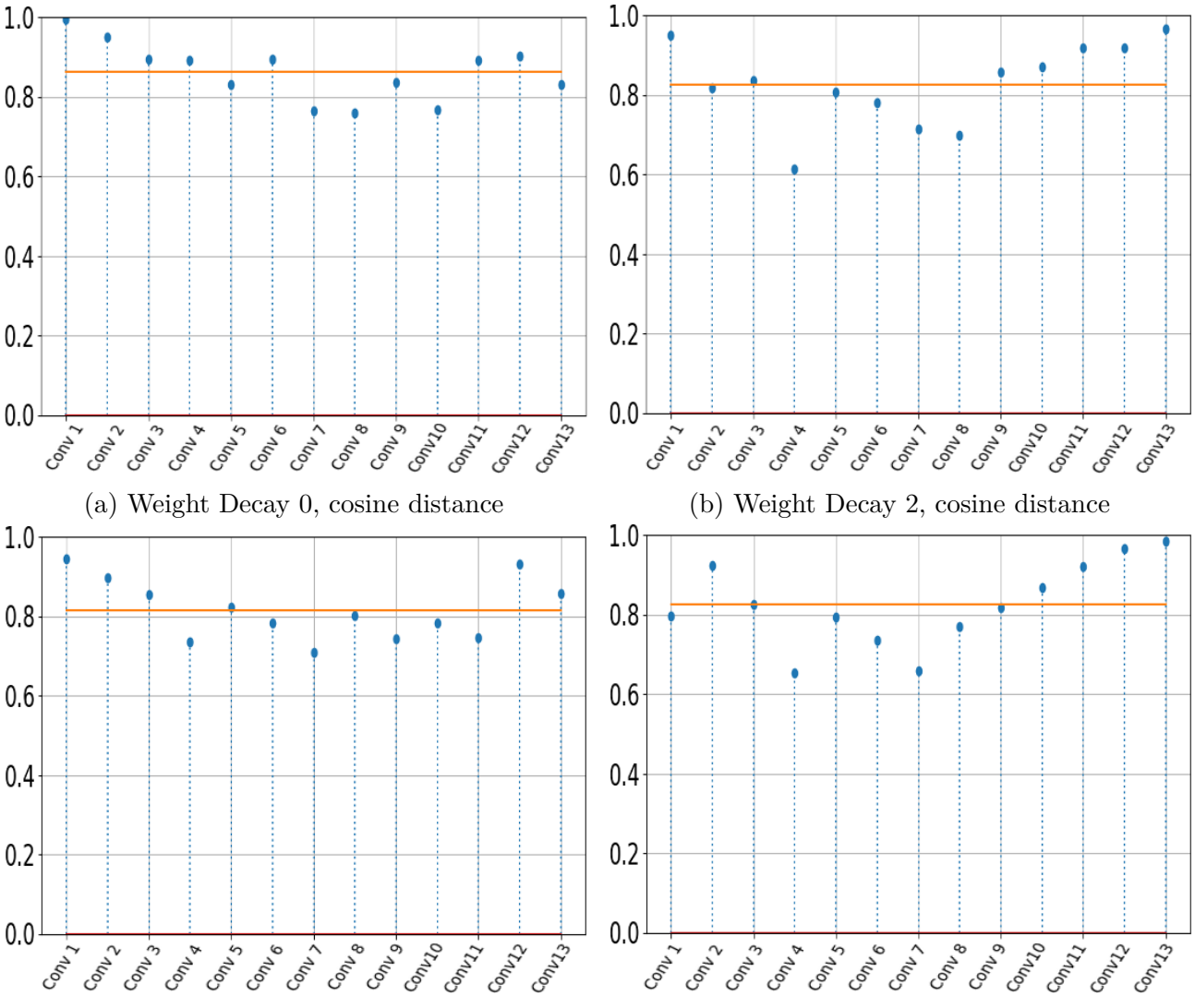
Dans le centre du réseau, tous les neurones sont utilisés pour dégager des clusters, et ainsi l'information utile à la classification. Voilà pourquoi il y a peu de neurones ignorés.

Vers la fin du réseau, on remarque une augmentation du nombre de neurones ignorés. Tout comme au début, il y a une présence plus importante de neurones qui sont toujours ou ne seront jamais activés par leur entrée. La raison dépendrait plus subtilement du type d'apprentissage du modèle.

En analogie avec nos expériences précédentes, on remarque SGD ne se comporte pas comme Weight Decay et Layca. On voit qu'il va plus facilement amener des neurones à ne pas s'activer ou inversement à s'activer très souvent, créant une pseudo circulation structurée des activations. Weight Decay et Layca force plus tous les neurones à plus souvent participer à la classification, entraînant une dédensification de cette circulation structurée.

8.4.7 Labels autour

Nous avons fait une expérience secondaire au coefficient de silhouette afin de regarder la proportion d'entrées activées et non-activées autour des entrées activées les plus proches. Pour cette expérience, nous regarderons 5% des activés les plus proches, et 5% autour de ceux-ci.



(a) Weight Decay 0, cosine distance

(b) Weight Decay 2, cosine distance

(c) Weight Decay 0, distance Euclidienne

(d) Weight Decay 2, distance Euclidienne

FIGURE 8.10 – Ratio des entrées activées sur les entrées non-activées autour des 5% des entrées activées les plus proches entre elles, et dans un voisinage de 5% pour Weight Decay 0 et 2.

Les résultats pour Weight Decay 0 et 2 sur la figure 8.10 nous montre que, autant pour la cosinus distance que la distance euclidienne, on peut voir au début du réseau une diminution de la proportion d'entrées activées autour des entrées activées sélectionnées quand le réseau généralise mieux. Il y a par contre une augmentation vers la fin du réseau.

On remarque aussi ce comportement un peu plus subtilement sur SGD, Layca et Weight Decay 1.

C'est aussi un symptôme d'une dédensification des premières couches due au fait que les neurones travail plus l'information, et font un meilleur travail de classification. C'est à dire que le neurone a travaillé en moyenne plus efficacement, et plus tôt l'information du réseau est ré-assemblée intelligemment. Une information mieux présentée de couche en couche aura comme impacte de faciliter la clustérisation, et facilitera la classification intelligente des images, amenant une bonne généralisation. Et non une mémorisation de comportement plus aléatoire, amenant une généralisation plus faible.

8.4.8 Ratio autour

Une autre expérience de clustérisation à été faite par la suite afin de confirmer et d'investiguer un peu plus l'expérience du coefficient de silhouette. Le but est de faire varier la distance de considération pour le calcul du coefficient de silhouette autour du point activé . On regarde donc dans un voisinage des activés sélectionnés la variation du coefficient de silhouette en fonction du changement de taille de ce voisinage. Ce voisinage des activations disponibles pour le coefficient de silhouette sera échelonné par ratio.

Ici, nous prendrons $n = 2000$ images d'activation pour plus de précision car on va prendre des petits ratios de n par la suite. Notons également qu'il fallut prendre des moyennes sur suffisamment d'essais (20) pour arriver à des résultats suffisamment fiables et consistants.

Les résultats pour les couches 1, 2, 3, 11, 12 et 13 sont présents sur les figures A.15, A.16, A.17, A.18, A.19 et A.20 en annexe. Il y a souvent une instabilité au début du graphe. C'est du au fait qu'on prend des ratios très petits (1/2000), et donc que le coefficient de silhouette est pris sur très peu d'échantillons.

Pour les couches du début (Couche 1, 2 et 3), on observe que le coefficient de silhouette a une tendance générale à diminuer quand le modèle généralise mieux. On voit que plus un modèle généralise bien, plus il a tendance à dé-densifier tôt ses clusters.

Dans les couches intermédiaires, le coefficient de silhouette est quasi-nul tout le temps.

Dans les couches de la fin (Couche 11, 12 et 13), on remarque que le coefficient de silhouette est presque nulle ou faible pour les réseaux qui généralise moins bien. Mais plus le réseau généralise bien, plus tôt on verra les clusters se densifier. Tout comme pour l'expérience précédente, il y a de plus en plus de présence de neurones ignorés dans les dernières couches. Des neurones qui ne seront jamais ou toujours activés.

Les résultats concernant la distance euclidienne n'ont pas montré de différence notable. Ni entre les différents modèles, ni entre les différents niveaux de généralisation. La valeur est juste plus faiblement visible.

Analyse

Cette expérience confirme nos résultats précédents. Il y a un mélange plus homogène des clusters dans l'espace pour les premières couches quand ça généralise mieux.

Le creux dans les graphes du silhouette score est plus étroit, indiquant que la création de sous-classes se ferait du coup plus rapidement et de manière plus forte, indiquant un réseau plus performant. Une meilleure préclassification de l'information, c'est-à-dire des entrées mieux disposées, amènerait les dernières couches denses à mieux pouvoir classifier ces mêmes entrées, et leur attribuer plus facilement un label correspondant. Il en découlera une classification plus intelligente, et donc une meilleure généralisation.

Il faut cependant noter que le coefficient de silhouette reste relativement le même pour un même neurone, peu importe le ratio. Les résultats nous montrent la moyenne de CS des différentes tendances provenant de différents neurones. C'est donc plus une tendance des clusters à être plus ou moins dense qui apparaît.

8.5 Mise en commun des expériences

Dans les expériences précédentes, nous avons dégagé plusieurs comportements dont voici un bref aperçu.

1. Pour l'information mutuelle entre 2 neurones (IM), il y a une diminution de leur valeur pour les premières couches quand le modèle généralise mieux. Pour la fin du réseau, c'est l'inverse qui se produit, et de plus en plus tôt.

2. Ces neurones pathologiques vont directement avoir de l'influence sur l'IM en augmentant la moyenne via l'ajout de paires dont l'IM vaut 1.

3. Les neurones pathologiques sont présents différemment en fonction de l'apprentissage, avec un pic anormal pour SGD qui généralise mieux. L'IM ne change pas uniquement grâce

aux neurones pathologiques.

4. Il y a une entropie qui varie légèrement, n'amenant pas de gros changement sur l'information mutuelle normalisée.

5. La moyenne du nombre de zéros dans les activations est un symptôme d'une bonne généralisation, mais lui aussi dépend du type d'optimiseur (SGD se comporte différemment de Weight Decay et Layca). De plus, ses variations sont liées au nombre de neurones pathologiques.

6. L'information mutuelle avec les labels (IML) a une valeur qui augmente aussi dans les dernières couches de plus en plus tôt dans le réseau quand le modèle généralise mieux. Cette augmentation se situe plus tôt que l'IM.

7. Pour le coefficient de silhouette (CS), on observe un comportement similaire à IM, aux mêmes valeurs.

8. Notons que pour l'IM ou le CS, le "creux" des couches centrales est moins long quand le réseau généralise mieux.

9. On remarque une tendance générale à la dédensification des clusters des neurones dans les premières couches, et inversement une densification dans les dernières quand le modèle généralise mieux.

10. Cette observation est confirmée par l'expérience des labels autour et celle des ratios autour.

Essayons maintenant de réunir ces observations.

La diminution de l'IM dans les premières couches quand on généralise mieux pourrait être une indication que les neurones du réseau sont de manière générale et individuelle plus efficaces et arrivent à distinguer plus d'informations différentes d'une même image. Le coefficient de silhouette nous confirme l'augmentation de la diversité par la dédensification des clusters quand la généralisation est meilleure.

Vers la fin, l'IM est plus élevée parce que l'information utile à la classification donnée par un neurone se recoupe de plus en plus avec l'information d'autres neurones. Les neurones de couches en couches continuent à travailler cette information, mais il devient de plus en plus visible que plusieurs des neurones de chaque couche dégagent des mêmes comportements pour la classification, des sous-classes similaires. Ces comportements seraient les biais utiles à la classification. Notons qu'il est possible que les sous-classes deviennent de plus en plus exclusives.

Finalement, il y a une chute d'IM dans la dernière couche dense due à la classification des dernières entrées dans les classes voulues.

Notons que la présence du creux dans les résultats n'est en réalité pas due à une absence de travail des neurones, mais plutôt à une absence d'ordre visible. Chaque neurone essaie de classer l'information de la manière la plus utile possible. Mieux le modèle fonctionne, plus étroit sera ce creux.

L'augmentation de l'IML, c'est le signe que le réseau arrive à *dégager plus d'informations relatives à la classification voulue*. C'est-à-dire que plus on avance dans les dernières couches, plus le réseau peaufine son traitement de l'information. Un réseau qui généralise bien véhiculera donc de l'information dégagée plus tôt dans le réseau.

Pour faire une analogie, au début, le réseau regarde du concret (les bords d'une image par exemple), puis va mélanger ce concret pour créer des concepts et des caractéristiques abstraits, et finalement arriver vers la fin du réseau à dégager le concept que l'on souhaite classifier (exemple "un chat").

On voit que l'IM et le CS augmentent seulement quelques couches après l'IML. Ils en sont des symptômes visibles plus tard, indirectement liés à l'IML.

De couche en couche, le *travail de classification en sous-classes de chaque neurone* est meilleur quand ça généralise mieux et donc que la classification finale est elle-même plus ordonnée, efficace et intelligente.

Un *modèle généralisera mieux* si ses hyperparamètres sont choisis au mieux pour l'apprentissage des neurones. Ceux-ci permettent aux neurones d'être plus efficaces dans leur traitement de l'information de l'image. Le travail de classification se fait individuellement, dégageant et travaillant de l'information de couche en couche, pour arriver à une meilleure classification entre les 2 dernières couches.

Notons qu'il est fortement probable qu'il y ait un set d'hyperparamètres optimal pour chaque base de données et chaque architecture. De plus, il devrait y avoir une taille de réseau minimum, un nombre minimum de couches qui modifie l'information. Il y a probablement présence de spécifications requises sur l'architecture du modèle. De plus, il ne faut pas oublier de prendre en considération la grandeur des biais de la base de données d'entraînement, sur lesquels le modèle

va se baser pour faire sa classification finale.

[SZ15] nous apprend que la profondeur du réseau compte pour avoir une meilleure généralisation dans la reconnaissance d'image dans les CNN. Nos résultats sont probablement en lien avec cette hypothèse vu que l'on se rend compte que le réseau a besoin de plusieurs couches de neurones qui travaillent l'information afin d'arriver à dégager l'information nécessaire à la classification. Avec plus de couches, la création au niveau neuronal de clusters plus distincts de couche en couche peut être faite. C'est-à-dire qu'avec un réseau plus grand, on peut faire une classification plus intelligente.

Si le réseau est trop petit et qu'on arrive à 100% sur les données d'entraînement, c'est probablement dû à une bonne mémorisation plutôt qu'à une bonne généralisation, ou à la captation de mauvais biais.

Si on considère l'existence de la formation de sous-classes, cela indiquerait une plus grande diversité dans ces sous-classes et ainsi une façon plus intelligente de les créer. Ces sous-classes vont avoir un impact positif sur la création de sous-classes dans la couche suivante et ainsi de suite. Les sous-classes de l'avant-dernière couche servant finalement à la classification recherchée.

Sans trop spéculer, les dernières expériences de Simon (image A.21 en annexe) nous pousse à croire en ce résultat, car l'information qui passe dans les réseaux qui généralisent mieux semble plus ordonnée et plus 'pure'. Il y a moins d'aléatoire et plus de 'patterns', c'est-à-dire des sous-classes plus significatives formées.

Par comparaison, on voit également dans [ZF14] qu'il y a une importance à distinguer *plus* de features *complètement différentes* dans les premières couches. Nous confirmerons cette approche par nos résultats sur l'information mutuelle et le coefficient de silhouette. Il y a effectivement une plus grande diversité de l'information dans les premières couches dans les réseaux qui généralisent mieux, visible par la diminution du CS et de l'IM dans les premières couches.

Pour situer notre travail par rapport à un autre papier intéressant, [KN19] nous dit que les modèles apprennent obligatoirement grâce aux *exemples faciles*. Mais que c'est pour les *exemples difficiles* que les modèles se trompent le plus et donc apprennent mieux à généraliser. Nous pensons que ce comportement est fortement lié au biais des images dans la base de données d'entraînement car il en est la cause de l'apprentissage. Que des images faciles, avec des biais les plus rencontrés dans les images, permettent plus facilement au réseau de créer des sous-classes liées aux classes, mais que les exemples difficiles, avec moins de biais semblables, permettent au réseau de mieux généraliser car il permet au réseau de mieux choisir l'importance des sous-classes

qui dégagent les biais utiles pour la classification.

Le réseau fonctionnant mieux permet de découvrir plus tôt la *création de sous-classes* utilisées pour la classification. Ces sous-classes sont directement liées aux classes recherchées et donc aux labels. Voilà pourquoi afin de voir si un modèle généralise mieux, il suffit de voir si l'IML augmente plus tôt dans le réseau. (Sous hypothèse que l'architecture est la même et que la base de données est suffisamment diversifiée.)

Cependant, les autres expériences nous en apprennent davantage sur la façon dont fonctionnent les optimiseurs.

Dans nos expériences sur le ratio autour, il nous semble remarquer des neurones dont la tendance est entre 0.4 et 0.6 (à la différence de neurones dont la tendance est plus basse). Ces neurones seraient représentatifs de la structure active du réseau, qui représente les branches plus 'pures' du réseau, très utiles à la classification. Leur détection permettrait de trouver un certain ratio par couche, et ainsi de donner une mesure de la création de la structure du réseau. Cette mesure serait fort différente entre les différentes méthodes de création des modèles (SGD, Weight Decay et Layca).

En ce qui concerne les différences d'apprentissage des modèles, on voit que SGD essayerait de créer un "arbre", une *structure* d'activation dans son réseau convolutif. Avec des branches plus actives mais plus subtiles et moins nombreuses à la fin. Effectivement, il y a une augmentation du nombre de neurones pathologiques inactifs et une augmentation de neurones toujours actifs quand on s'approche des "racines". Dans ses couches denses, SGD va utiliser au maximum ses ressources associatives pour dégager la classification requise. Ce symptôme est capté dans le tableau 8.1.

Un réseau tel que SGD sera visiblement plus efficace s'il arrive à placer ses "racines actives" principales plus profondément, et donc on verra une augmentation plus tôt dans les couches du nombre de neurones pathologiques. Il y aura aussi une première couche dense plus fortement utilisée.

Pour Layca et WD, la structure active va se former également dans le réseau convolutif mais de manière plus complète et répartie, moins centralisée. Une plus grande partie des ressources des neurones va être utilisée jusque plus loin dans le réseau. De plus, avec la plus grande mise à zéro des poids, il le réseau est alors moins "touffu". Ceci expliquerait pourquoi ils résisteraient

mieux au prunning.

Un réseau tel que Layca ou WD sera visiblement plus efficace s'il utilise plus de neurones dans ses couches convolutive et s'il doit moins utiliser de neurones utiles à la classification dans sa première couche dense.

8.6 Conclusion

Ce chapitre nous a donc montré qu'il existe des symptômes visibles lorsqu'un modèle généralise mieux. Ces symptômes sont la création intelligente, couche après couche, de sous-classes dans les neurones du réseau. Ces sous-classifications sont créées de manière à être le plus utile à la classification finale voulue, c'est-à-dire en distinguant les biais relatifs à celle-ci. Une classification plus intelligente est directement liée à de meilleures performances en matière de généralisation.

Le bon apprentissage des modèles est donc lié aux hyperparamètres qui vont influencer l'efficacité du travail de classification individuel des neurones.

9 Conclusions

9.1 Principales constatations

Au terme de ce travail, le phénomène de généralisation est mieux compris, et ce grâce à l'étude de différents réseaux entraînés par différents optimiseurs : *sgd*, *weight decay* et *layca*.

Dans la première partie de notre analyse, nous avons étudié la dynamique des poids lors de l'entraînement. En continuation des recherches menés par [CV18b], nous avons montré que la grandeur de la cosinus distance reflétait la capacité d'un optimiseur à mettre une grande partie de ses poids à zéro. Cette aptitude a été mise en avant pour *weight decay* et *layca* particulièrement. La structure du réseau crée par ces optimiseurs est beaucoup plus nette et dépourvue de poids parasites, ce qui leur permet d'être plus robustes à l'élagage aléatoire et ciblé de leurs poids.

Dans la seconde partie de notre analyse, nous avons étudié le comportement des activations de nos modèles.

Partant d'une recherche sur la redondance d'information, on a vu que l'information mutuelle entre neurones (IM) suit un comportement induit par l'information mutuelle avec les classes (IMC). Lorsque la généralisation est meilleure, l'information mutuelle augmentera plus tôt dans le réseau. Ce phénomène met en avant la création de sous-classes, liées à la classification voulue. De plus quand la généralisation est meilleure, on observe une diminution de l'IM et du coefficient de silhouette (CS) dans les premières couches et une augmentation de ceux-ci dans les dernières couches. Ces phénomènes mettent aussi en avant une meilleure capacité des neurones à créer ces sous-classes quand les hyper-paramètres sont bien choisis.

L'étude des ratios d'activations nous montre aussi que *weight decay* et *layca* vont permettre une régularisation supplémentaire, amenant ainsi plus de diversité dans les biais captés par le réseau. Cette régularisation est similaire à *dropout*.

On met donc en avant trois caractéristiques de la généralisation. La première est qu'il faut

capter des biais adéquats [1]. La seconde est que plus l'on observe de biais différents, plus notre classification sera robuste, et ainsi la généralisation en profitera. C'est ce qu'essaie d'apporter l'usage de *dropout*, *weight decay* et *layca* dans l'apprentissage, et c'est ce qui est mis en avant par le *prunning*. La troisième est qu'un réseau plus élagué aura tendance à montrer que le réseau à capter les bons biais. Cependant, notons qu'il ne faut pas spécialement avoir un réseau bien élagué pour capter les bons biais.

9.2 Limitations

Notre travail possède quelques limitations. Elles sont dues d'une part à la technologie utilisée pour faire tourner nos simulations et d'autre part par le choix des méthodes utilisés. Par exemple, l'analyse des activations requérait d'appliquer un plus grand nombre de stimuli (images) au réseau.

Citons ici d'autres limitations :

- Nous nous limitons à une seule architecture convolutive *VGG-16*.
- Cette architecture a été modifiée selon nos vouloirs, utilisant la *batch normalisation* et n'utilisant pas *dropout*.
- Nous n'utilisons qu'un certain panel de paramètres parmi une large gamme que ce soit pour la fonction objectif, la fonction d'activation des neurones, etc.

En plus d'expériences supplémentaires à faire, il y a encore beaucoup de changements possibles sur nos expériences dues à ces limitations.

9.3 Travail futur

Nous verrons ici les questions restées sans réponses et les directions de recherche futures.

Pour la dynamique des poids, il pourrait être intéressant d'étudier plus en profondeur l'aptitude de *weight decay* et *layca* de mettre des poids parasites à zéro (autrement dit, d'atteindre un cosinus distance proche de 1). Comme mentionné brièvement dans le **Chapitre 6**, *layca* n'atteint pas toujours systématiquement un cosinus distance de 1 sur ses couches. Ceci indique son inaptitude à mettre certains poids parasites à zéro. Ceci n'est pas encore bien compris.

[1] Il est important de rappeler que si les données entrées ont de mauvais biais en commun (exemple : trop peu de diversité dans les images), on n'est pas certain que les biais captés par le réseaux soient adéquats.

Pour l'activation, il serait peut-être intéressant de voir l'information mutuelle entre deux neurones de couches différentes. En effet, il serait possible que de l'information semblable se retrouve sur des couches différentes.

On pourrait aussi regarder autre chose que l'information mutuel, comme la SVCCA [Mai+17].

La détection et l'analyse des biais dans les bases de données pourrait être largement plus investie. Principalement leurs liens avec les concepts de classes et la détection de ces biais par les neurones (seuls ou en réseau).

Des expériences supplémentaires pourraient être faites afin d'investiguer plus attentivement les bénéfices de *layca*. Intuitivement, il nous semble que l'avantage majeur de *layca* sur *sgd* et *weight decay* est qu'il permet de ne pas laisser tomber des liens trop rapidement lors de l'entraînement, permettant ainsi de laisser toutes les chances possibles à la structure du réseau de bien se former tout en agissant de manière négative sur le bruit.

L'investigation des bénéfices des autres méthodes pourrait amener la création de méthodes mixes. Celles-ci pourraient tirer parti du meilleur de chacune afin d'arriver aux meilleures performances en terme de généralisation possible. Par exemple, utiliser *sgd* au début de l'apprentissage, pour ensuite utiliser *layca* et finir par *weight decay*.

Finalement, une analogie intéressante pourrait être faite avec le cerveau et sa plasticité neuronale [Wik19f].

9.4 Derniers mots

Dans ce travail, nous espérons avoir retiré un peu du voile qui cache la connaissance du processus de généralisation dans les *CNN* en mettant en lumière certains comportements liés à ce phénomène. Cependant, il reste encore de nombreuses observations et expériences plus poussées à faire pour affirmer ou contredire nos résultats et hypothèses.

Nous espérons que la lecture de cette dissertation vous a plu et nous vous remercions de votre intérêt. Pour toutes questions, remarques ou corrections, n'hésitez surtout pas à nous contacter.

novak_mathias@hotmail.com
antoinevanhoof@hotmail.com

Bibliographie

[J+91] Hertz J. et al. « Introduction To The Theory Of Neural Computation ». *Physics Today - PHYS TODAY* 44 (jan. 1991). DOI : 10.1063/1.2810360.

[Dan+11] Ciresan DAN et al. « Flexible, High Performance Convolutional Neural Networks for Image Classification » (2011).

[G11] Menardi G. « Density-based Silhouette diagnostics for clustering methods ». *Stat Comput 21 :295* (2011). URL : https://doi.org/10.1007/s11222-010-9169-0.

[DUJ12] Ciresan DAN, Meier UELI et Schmidhuber JÜRGEN. « Multi-column deep neural networks for image classification » (juin 2012).

[Ale13a] Krizhevsky ALEX. « ImageNet Classification with Deep Convolutional Neural Networks » (nov. 2013).

[Ale13b] Krizhevsky ALEX. « ImageNet Classification with Deep Convolutional Neural Networks » (nov. 2013).

[Lab13] LISA LAB. « Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation » (août 2013).

[Sri+14] Nitish SRIVASTAVA et al. « Dropout : A Simple Way to Prevent Neural Networks from Overfitting ». *Journal of Machine Learning Research 15* (juin 2014).

[ZF14] Matthew D. ZEILER et Rob FERGUS. « Visualizing and Understanding Convolutional Networks » (2014).

[al15] Goodfellow et AL. « Deep Learning book » (2015).

[IS15] Sergey IOFFE et Christian SZEGEDY. « Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift » (2015).

[SZ15] Karen SIMONYAN et Andrew ZISSERMAN. « Very Deep Convolutional Networks for Large-Scale Image Recognition ». *CoRR* abs/1409.1556 (2015).

[YZ15] Gal YARIN et Ghahramani ZOUBIN. « Dropout as a Bayesian Approximation : Representing Model Uncertainty in Deep Learning ». abs/1506.02142 (juin 2015).

[Ano16] ANONYMOUS. « Outliers or Key Profiles ? Understanding Distance Measures for Authorship Attribution » (juil. 2016). URL : http://dh2016.adho.org/abstracts/253.

[Ken16] Brian KENG. « A Probabilistic Interpretation of Regularization » (août 2016). URL : http://bjlkeng.github.io/posts/probabilistic-interpretation-of-regularization/.

[Kes+16] Nitish Shirish KESKAR et al. « On Large-Batch Training for Deep Learning : Generalization Gap and Sharp Minima » (sept. 2016).

[Tas16] Ahmet TASPINAR. « The perceptron » (déc. 2016). URL : https://jontysinai.github.io/jekyll/update/2017/11/11/the-perceptron.html.

[Aru+17] Kai ARULKUMARAN et al. « Deep Reinforcement Learning : A brief survey » (2017).

[Bus17] Vitaly BUSHAEV. « Stochastic Gradient Descent with momentum » (déc. 2017). URL : https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d.

[Col17] Jaron COLLIS. « Glossary of Deep Learning : Batch Normalisation » (juin 2017). URL : https://medium.com/deeper-learning/glossary-of-deep-learning-batch-normalisation-8266dcd2fa82.

[Dev+17] Arpit DEVANSH et al. « A Closer Look at Memorization in Deep Networks ». *CoRR* abs/1706.05394 (juin 2017).

[EID17] Hoffer ELAD, Hubara ITAY et Soudry DANIEL. « Train longer, generalize better : closing the generalization gap in large batch training oF neural networks ». abs/1705.08741 (mai 2017).

[Gom+17] Aidan N. GOMEZ et al. « Targeted Dropout » (oct. 2017).

[Ham17] Habibi Aghdam HAMED. « Guide to convolutional neural networks : a practical application to traffic-sign detection and classification » (mai 2017).

[Kru+17] David KRUEGER et al. « Deep Nets Don't Learn via Memorization ». *International Conference on Learning Representations - Workshop track* (2017), p. 1–4.

[Lau17] Suki LAU. *Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning*. https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1. (Accessed on 08/10/2019). Juil. 2017.

[Mai+17] Raghu MAITHRA et al. « SVCCA : Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability ». abs/1706.05806 (juin 2017).

[Nag17] Anuja Nagpal. « L1 and L2 Regularization Methods » (oct. 2017). URL : https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c.

[Nie17] Michael Nielsen. « http ://neuralnetworksanddeeplearning.com/chap1.html » (déc. 2017).

[PY17] Dabkowski Piotr et Gal Yarin. « Real Time Image Saliency for Black Box Classifiers ». abs/1705.07857 (mai 2017).

[Sin17] Jonty Sinai. « The perceptron » (nov. 2017). URL : https://jontysinai.github.io/jekyll/update/2017/11/11/the-perceptron.html.

[SSN17] Masanori Suganuma, Shinichi Shirakawa et Tomoharu Nagao. « A genetic programming approach to designing convolutional neural network architectures » (2017).

[Tic17] Dmitrij Tichonov. « Debunking loss functions in deep learning » (oct. 2017). URL : https://medium.com/@dmitrijtichonov/debunking-loss-functions-in-deep-learning-4b9abc4c8d4c.

[Val17] Venelin Valkov. « Creating a Neural Network from Scratch—TensorFlow for Hackers (Part IV) » (mai 2017). URL : https://medium.com/@curiousily/tensorflow-for-hackers-part-iv-neural-network-from-scratch-1a4f504dfa8.

[Zha+17] Chiyuan Zhang et al. « Understanding deep learning requires rethinking generalization ». *CoRR* abs/1611.03530 (2017).

[ALL18] Zeyuan Allen-Zhu, Yuanzhi Li et Yingyu Liang. « Learning and Generalization in Overparameterized Neural Networks, Going Beyond Two Layers ». *CoRR* abs/1811.04918 (2018).

[Bab18] Boris Babenko. « weight decay vs L2 regularization » (avr. 2018). URL : https://bbabenko.github.io/weight-decay/.

[CV18a] Simon Carbonnelle et Christophe De Vleeschouwer. « Discovering the mechanics of hidden neurons » (2018). URL : https://openreview.net/pdf?id=H1srNebAZ.

[CV18b] Simon Carbonnelle et Christophe De Vleeschouwer. « On layer-level control of DNN training and its impact on generalization ». *CoRR* abs/1806.01603 (2018). URL : https://github.com/Simoncarbo/An-experimental-study-of-layer-level-training-speed-and-its-impact-on-generalization.

[Cav18] Michele Cavaioni. « DeepLearning series : Convolutional Neural Networks » (fév. 2018). URL : https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524.

[Céc18] Hautecoeur Cécile. « Impact of coherence and occurrence in the training of CNN for anomaly detection » (2018).

[Dom18] Carlo Luschi Dominic Masters. « Revisiting Small Batch Training for Deep Neural Networks ». abs/1804.07612 (avr. 2018).

[HL18] Sneha H.L. « 2D Convolution in Image Processing » (nov. 2018). URL : https://www.allaboutcircuits.com/technical-articles/two-dimensional-convolution-in-image-processing/.

[JPY18] Mukhoti J., Stenetorp P. et Gal Y. « On the Importance of Strong Baselines in Bayesian Deep Learning ». *CoRR* abs/1811.09385v2 (nov. 2018).

[Jai18] Shubham Jain. « An Overview of Regularization Techniques in Deep Learning » (avr. 2018). URL : https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/.

[Jan18] Harrison Jansma. « Intuit and Implement : Batch Normalization » (août 2018). URL : https://towardsdatascience.com/intuit-and-implement-batch-normalization-c05480333c5b.

[Pal18] Shay Palachy. « Understanding the scaling of L2 regularization in the context of neural networks » (nov. 2018). URL : https://towardsdatascience.com/understanding-the-scaling-of-l%5C%C2%5C%B2-regularization-in-the-context-of-neural-networks-e3d25f8b50db.

[Sah18] Sumit Saha. « A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way » (déc. 2018). URL : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[Tre+18] Jonathan Tremblay et al. « Training Deep Networks with Synthetic Data : Bridging the Reality Gap by Domain Randomization ». *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2018), p. 1082–10828.

[AS19] Subutai Ahmad et Luiz Scheinkman. « How Can We Be So Dense ? The Benefits of Using Highly Sparse Representations ». *CoRR* abs/1903.11257 (2019).

[Ano19a] Anonymous. « An experimental study of layer-level training speed and its impact on generalization » (2019). URL : https://openreview.net/pdf?id=HkeILsRqFQ.

[Ano19b] Anonymous. « Categorical crossentropy » (mai 2019). URL : https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy.

[Ano19c] Anonymous. « Neural Networks » (2019). URL : http://kseow.com/nn/.

[Ano19d] Anonymous. « Normalized Mutual Information. Estimating Clustering Quality » (2019). URL : https://course.ccs.neu.edu/cs6140sp15/7_locality_cluster/Assignment-6/NMI.pdf?fbclid=IwAR3BybxzAFcup2PuJ7qcZ1dZAFgOeSfHuShecnArSXqOnOc

[Cod19] Jean Carlo CODOGNO. « Neural network using stochastic gradient descent » (mai 2019). URL : https://www.kaggle.com/jcodogno/neural-network-using-sgd.

[con19] 41 CONTRIBUTORS. « scikit-learn/sklearn/metrics/cluster/supervised.py » (avr. 2019). URL : https://github.com/scikit-learn/scikit-learn/blob/b7b4d3e2f/sklearn/metrics/cluster/supervised.py.

[FNN19] Stanislav FORT, Pawel Krzysztof NOWAK et Srini NARAYANAN. « Stiffness : A New Perspective on Generalization in Neural Networks ». *CoRR* abs/1901.09491 (2019).

[KN19] Ikki KISHIDA et Hideki NAKAYAMA. *Empirical Study of Easy and Hard Examples in CNN Training*. 2019. URL : https://openreview.net/forum?id=HJepJh0qKX.

[Wik19a] WIKIPÉDIA. « Apprentissage supervisé » (mai 2019).

[Wik19b] WIKIPÉDIA. « Batch normalization » (mai 2019).

[Wik19c] WIKIPÉDIA. « Early stopping » (mai 2019).

[Wik19d] WIKIPÉDIA. « Information mutuelles » (mai 2019).

[Wik19e] WIKIPÉDIA. « Neurone formel » (mai 2019).

[Wik19f] WIKIPÉDIA. « Plasticité neuronale » (mai 2019).

[Wik19g] WIKIPÉDIA. « Réseau de neurones artificiels » (mai 2019).

[Wik19h] WIKIPÉDIA. « Shannon Entropy » (mai 2019).

[Wik19i] WIKIPÉDIA. « Silhouette (clustering) » (mai 2019).

[Wik19j] WIKIPÉDIA. « Stochastic gradient descent » (mai 2019).

[Wik19k] WIKIPÉDIA. « Weight Decay » (mai 2019).

[Wu19] David WU. « L2 Regularization and Batch Norm » (jan. 2019). URL : https://blog.janestreet.com/l2-regularization-and-batch-norm/.

[Zho+19] Hattie ZHOU et al. « Deconstructing Lottery Tickets : Zeros, Signs, and the Supermask » (2019).

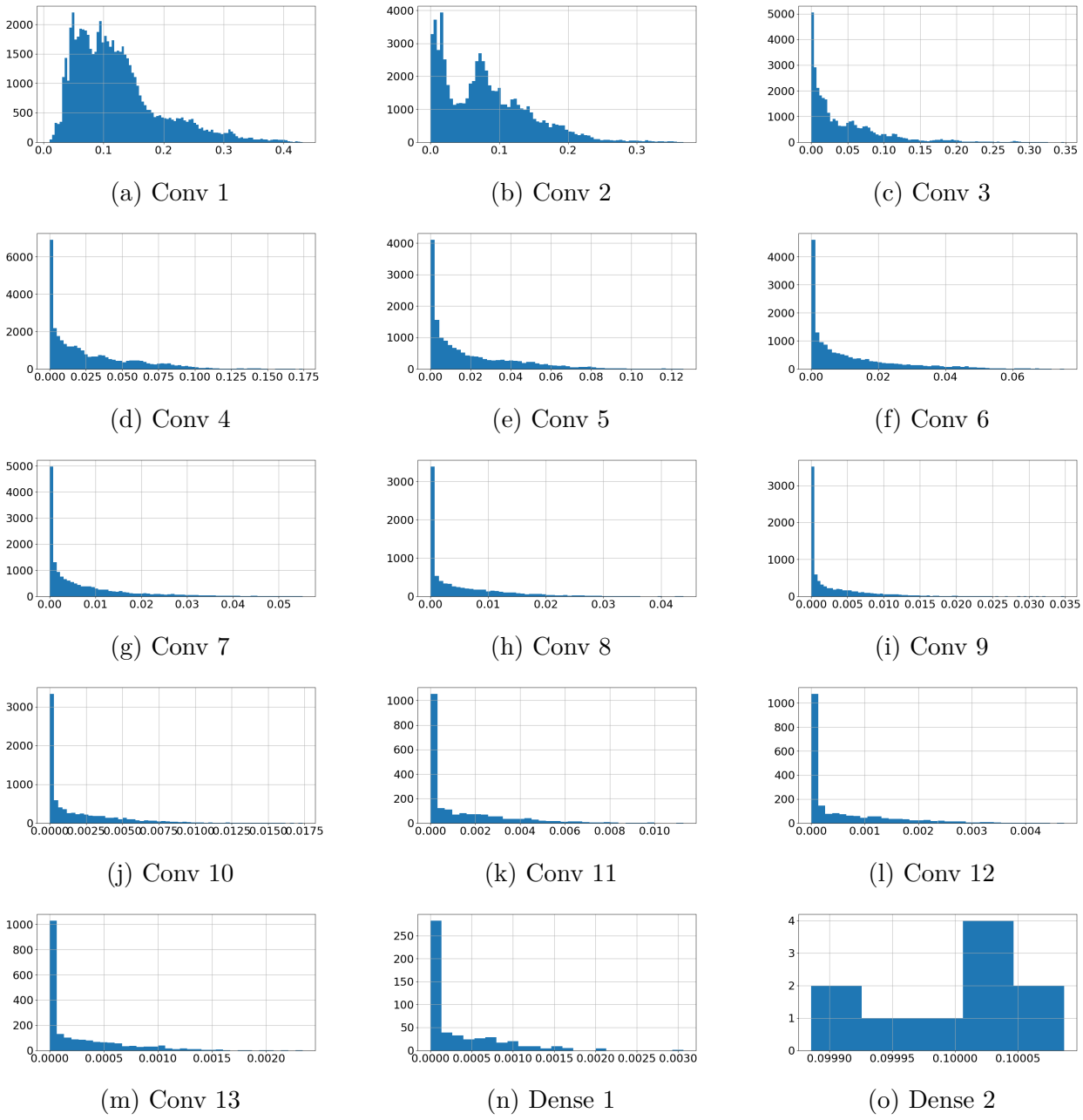
A Appendice

Activations

Histogramme de la moyenne des activations par couche

On peut voir sur ces graphes l'histogramme de la moyenne des activations par couche. La moyenne est faite sur un échantillon aléatoire de $n = 2000$ images. La définition du nombre de barres dans les histogrammes est équivalente au nombre d'activations (et donc de neurones) par couche.

Réseau Initial



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

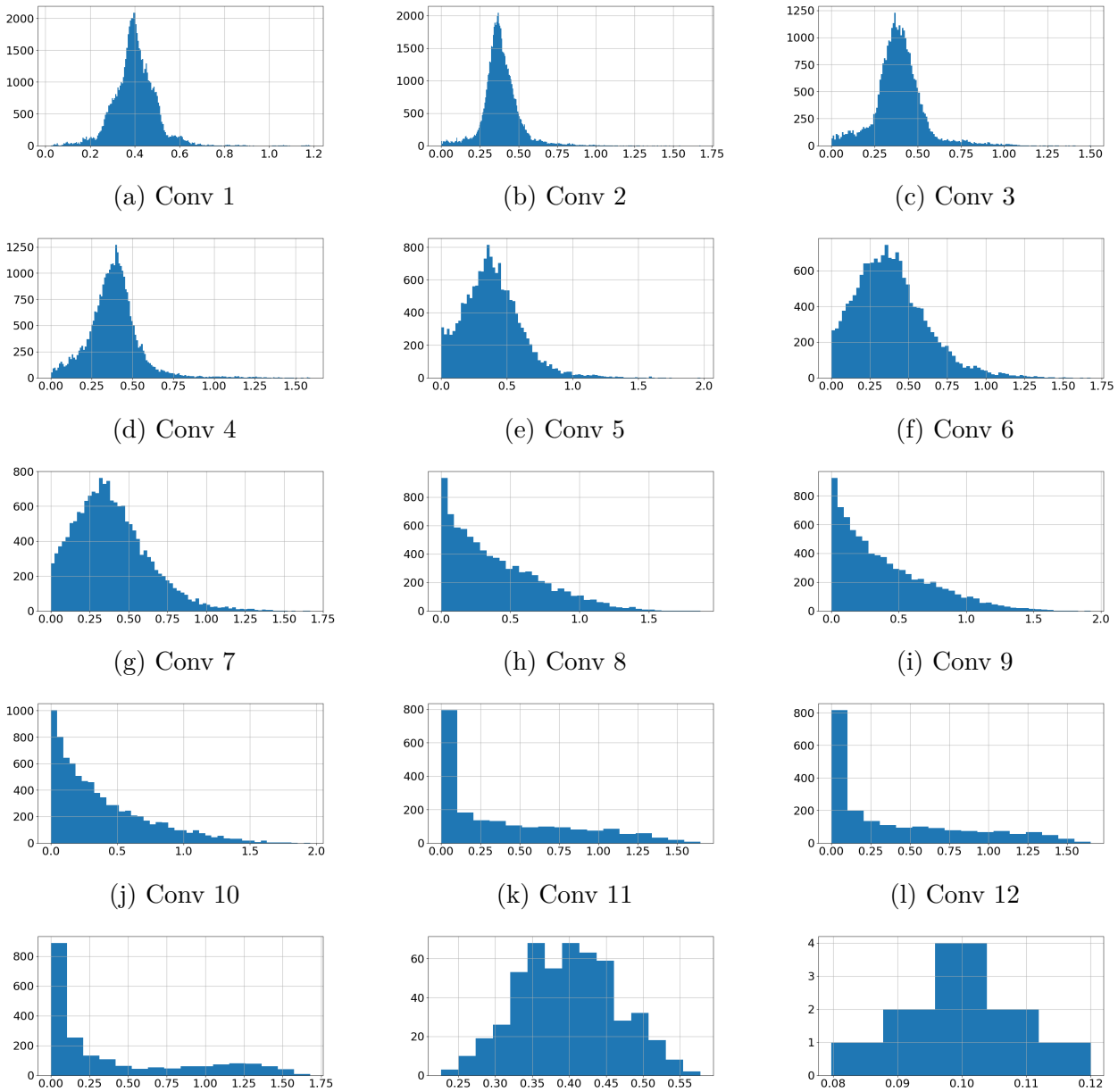
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.1 – Réseau Initial. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau SGD 65%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

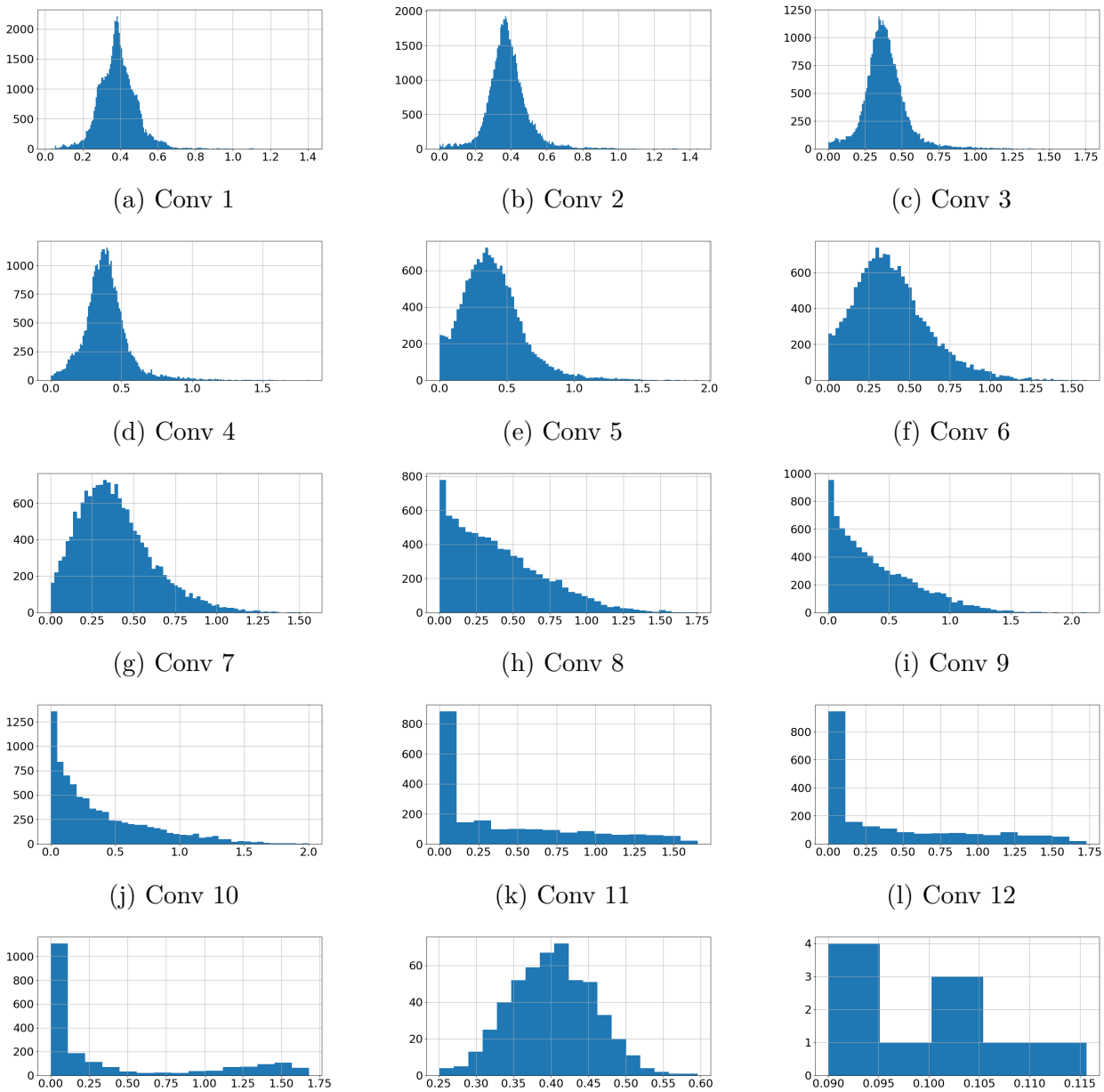
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.2 – Réseau SGD 0. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau SGD 75%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

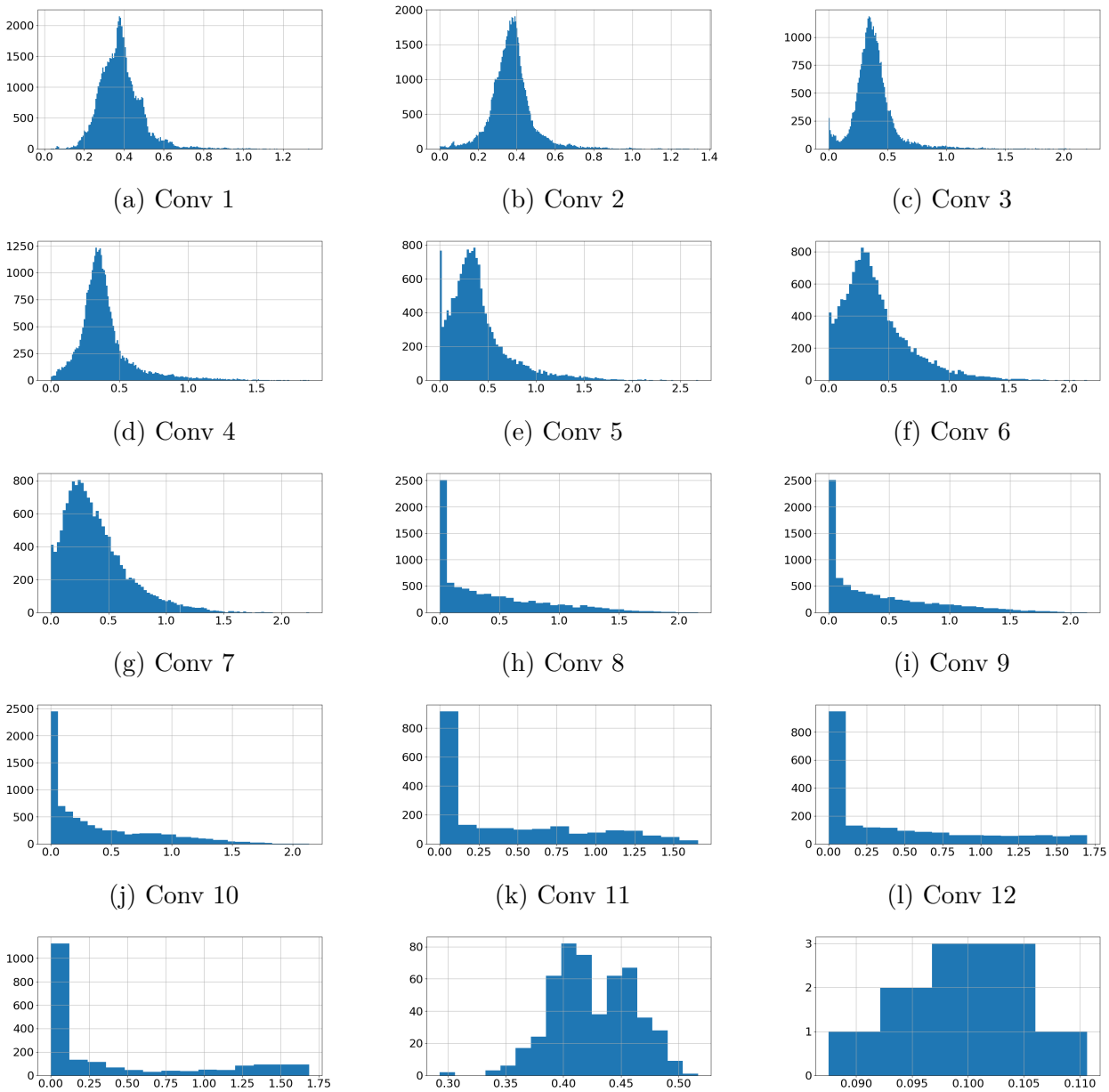
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.3 – Réseau SGD 1. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau SGD 85%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

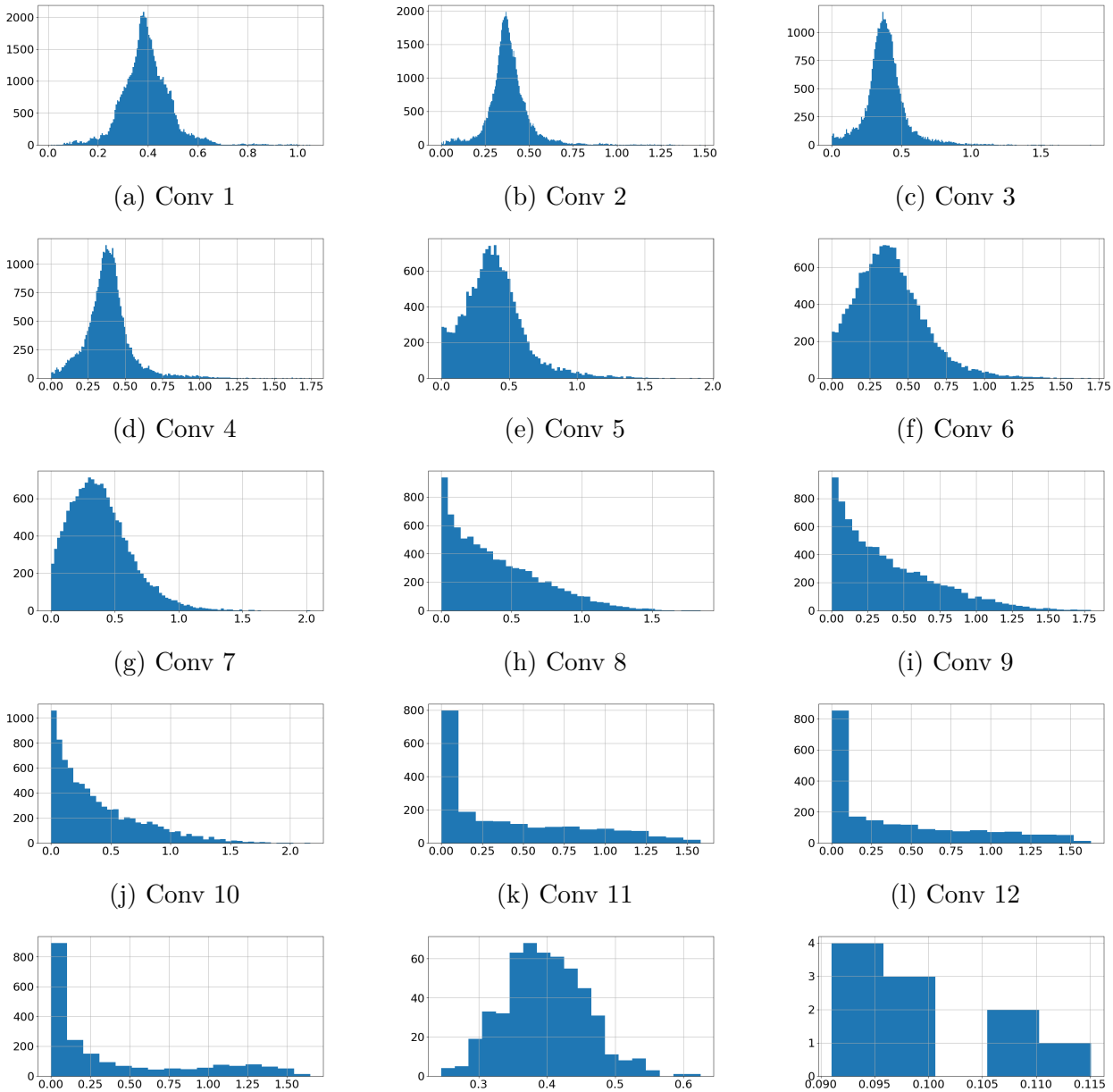
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.4 – Réseau SGD 2. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau Weight Decay 65%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

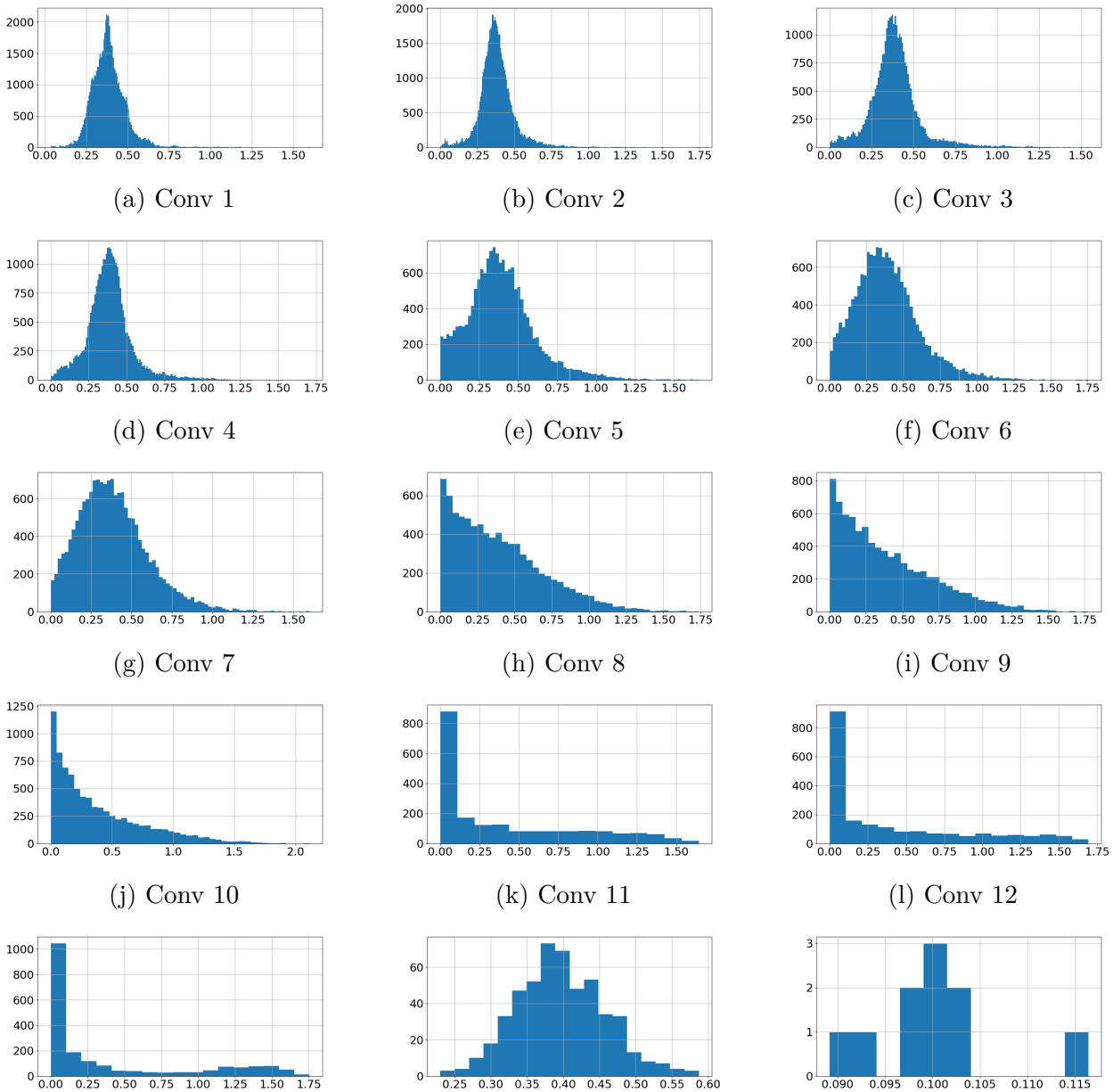
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.5 – Réseau WD 0. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau Weight Decay 75%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

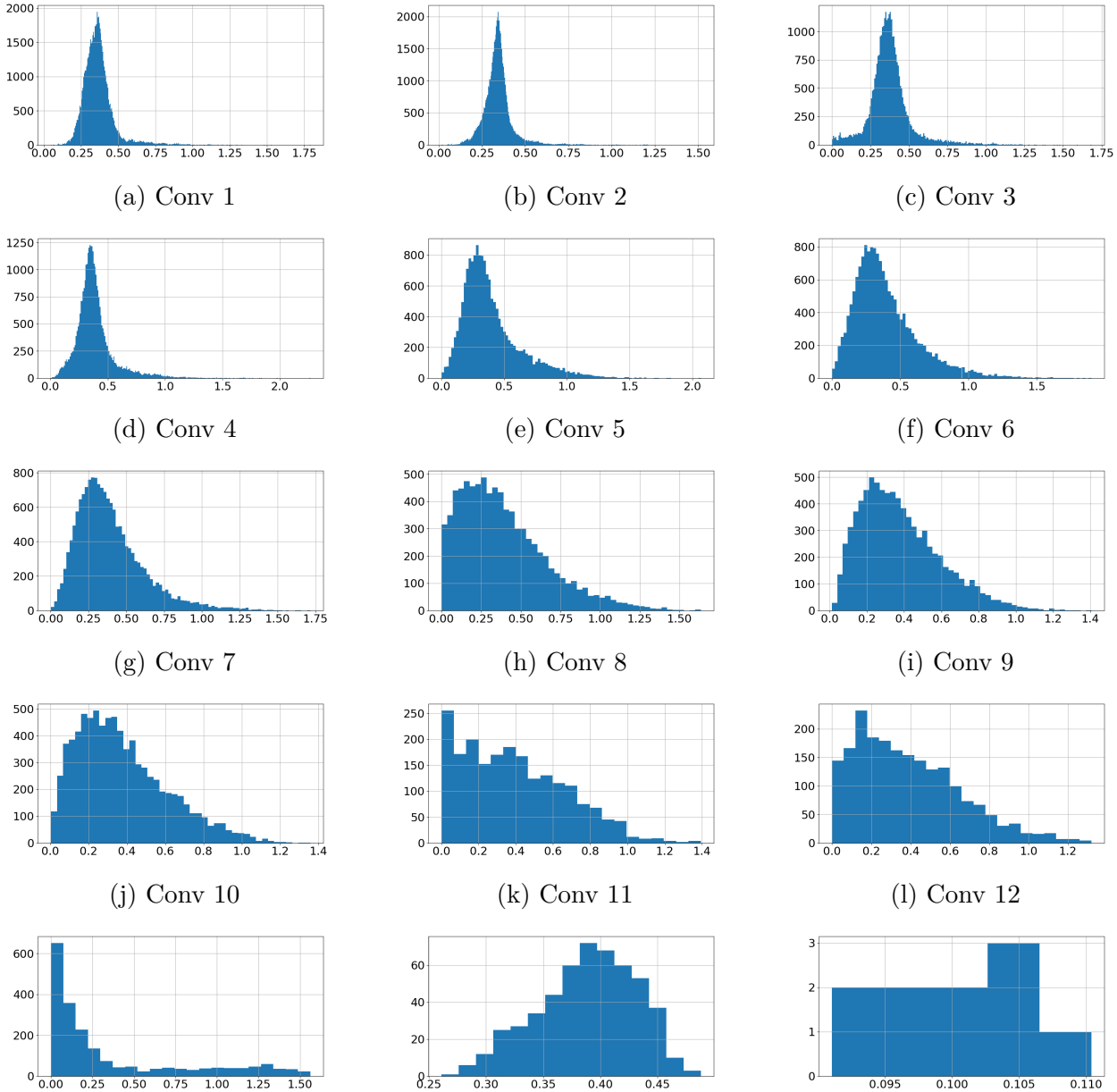
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.6 – Réseau WD 1. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau Weight Decay 85%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

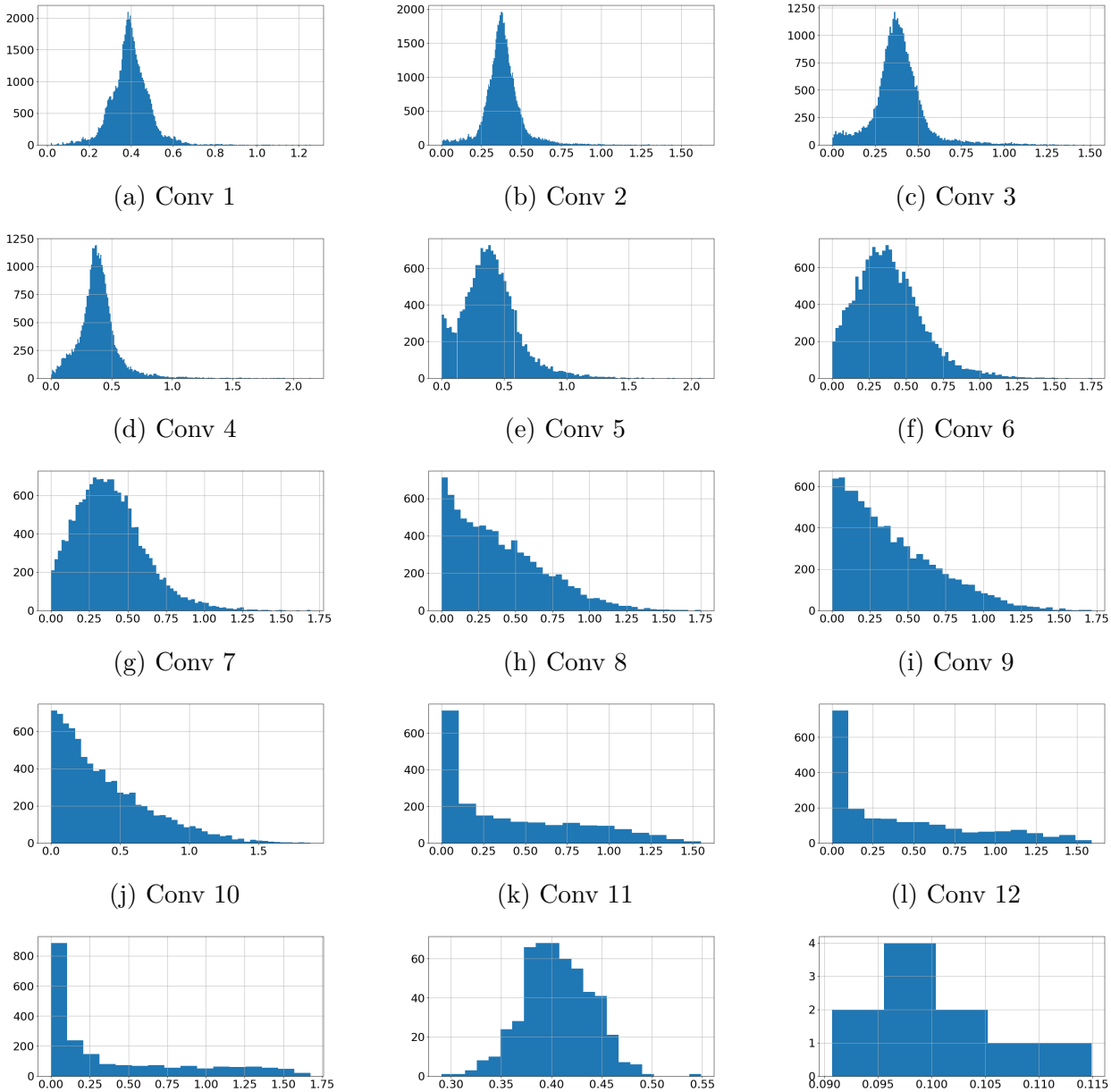
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.7 – Réseau WD 2. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau Layca 65%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

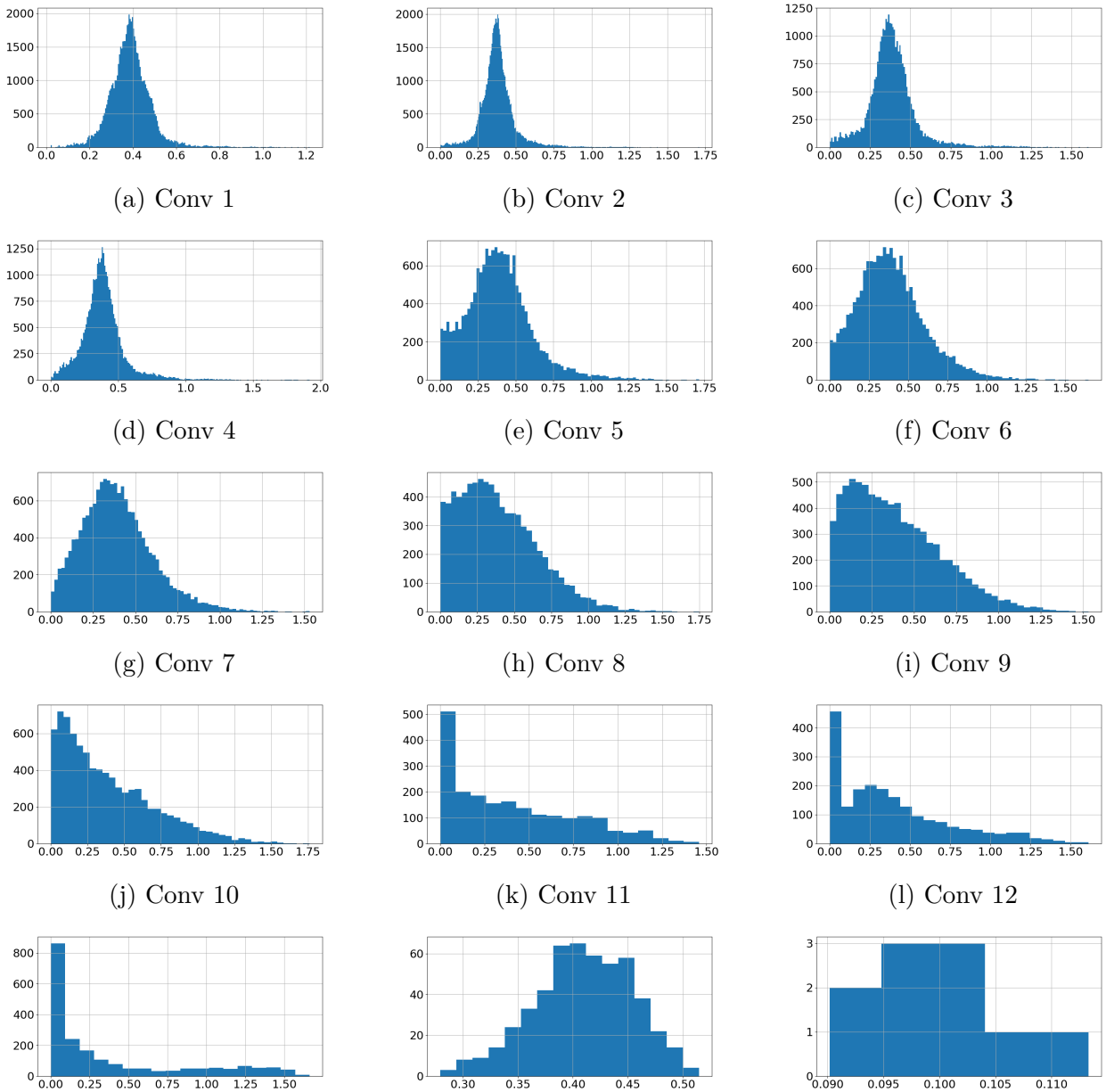
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.8 – Réseau Layca 0. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau Layca 75%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

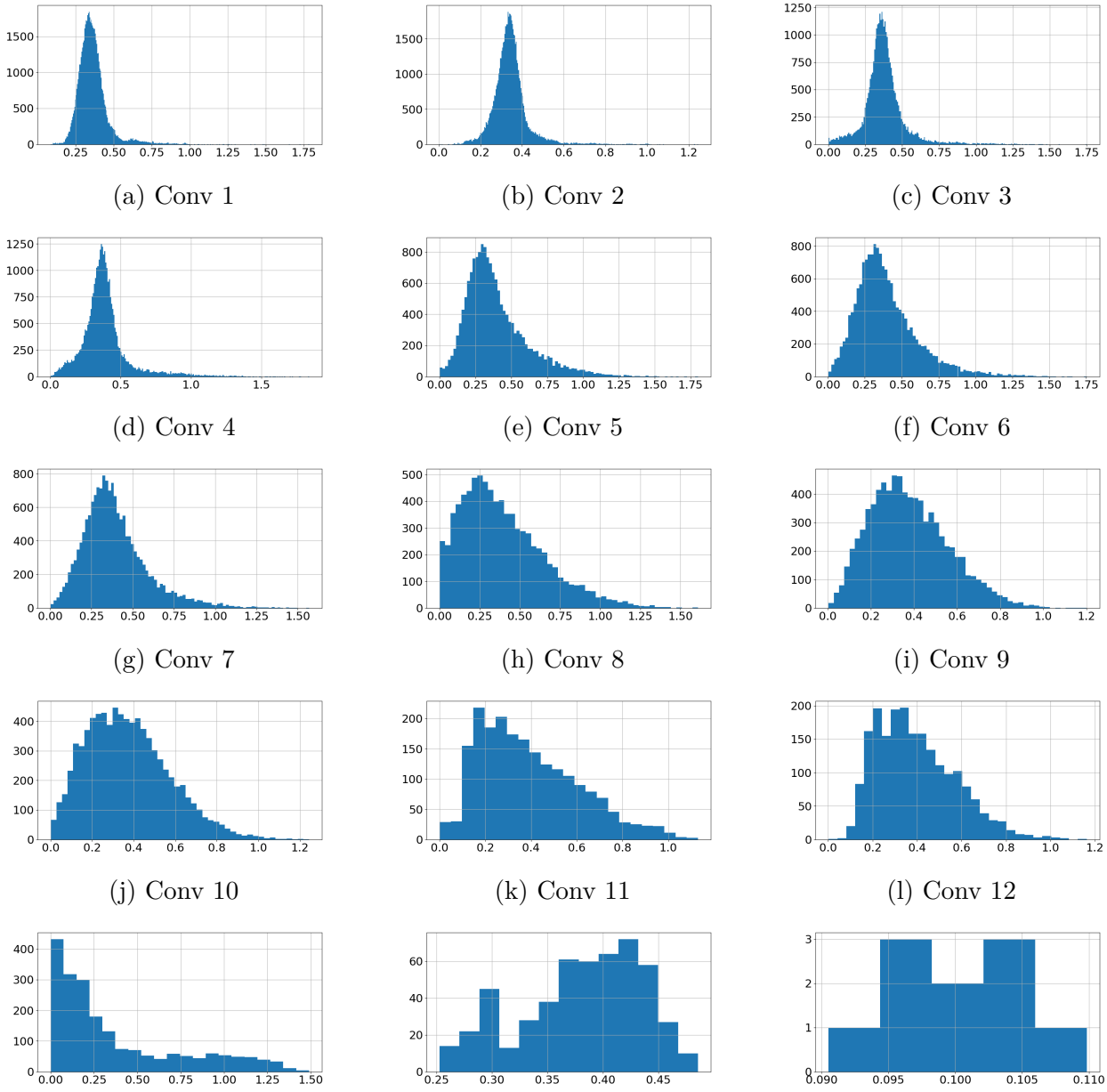
(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.9 – Réseau Layca 1. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

Réseau Layca 85%



(a) Conv 1 (b) Conv 2 (c) Conv 3

(d) Conv 4 (e) Conv 5 (f) Conv 6

(g) Conv 7 (h) Conv 8 (i) Conv 9

(j) Conv 10 (k) Conv 11 (l) Conv 12

(m) Conv 13 (n) Dense 1 (o) Dense 2

FIGURE A.10 – Réseau Layca 2. L'axe horizontale représente la valeur de l'activation, et l'axe verticale son occurrence dans la couche.

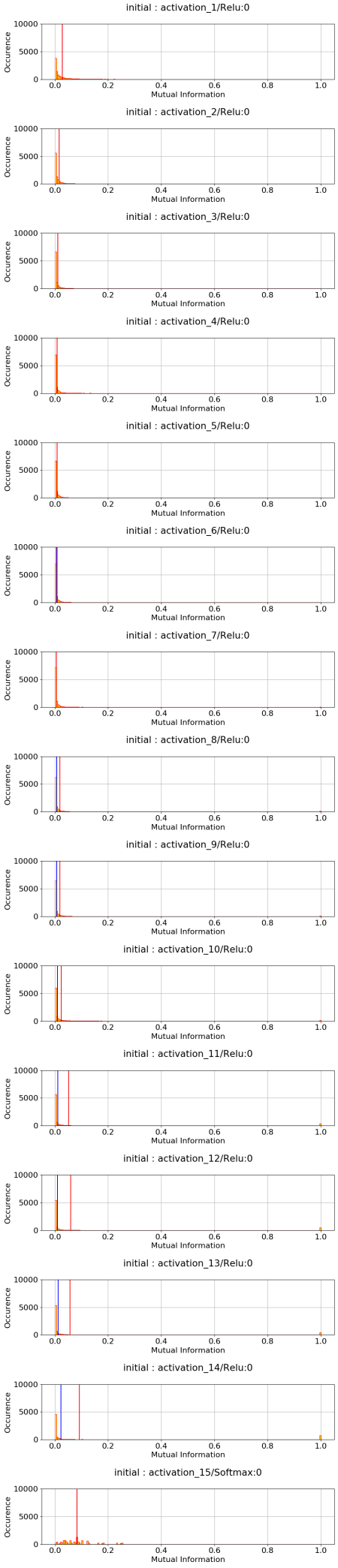


FIGURE A.11 – MI histogramme pour un réseau Initial

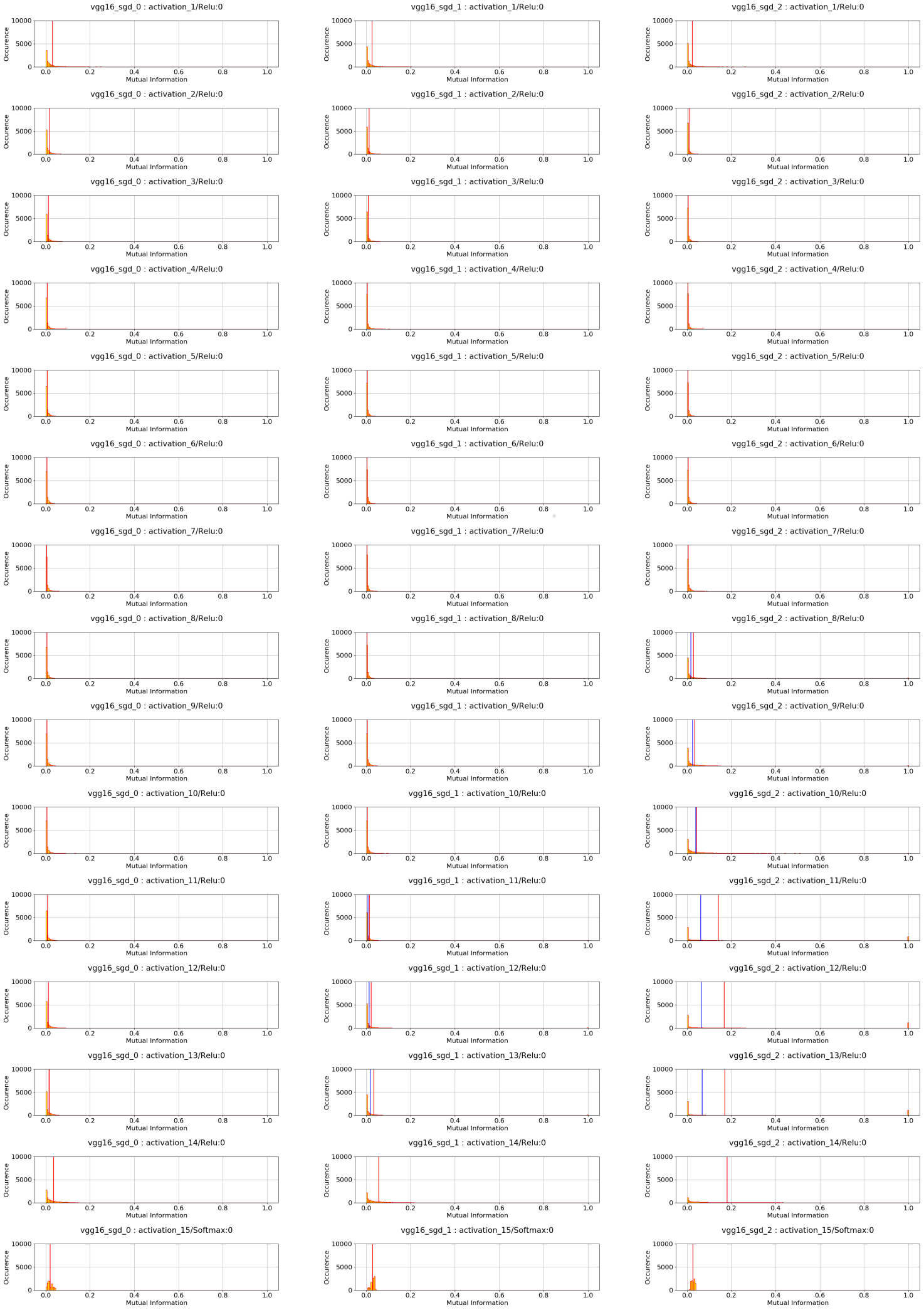


FIGURE A.12 – MI histogramme pour SGD (65%,75% et 85%)

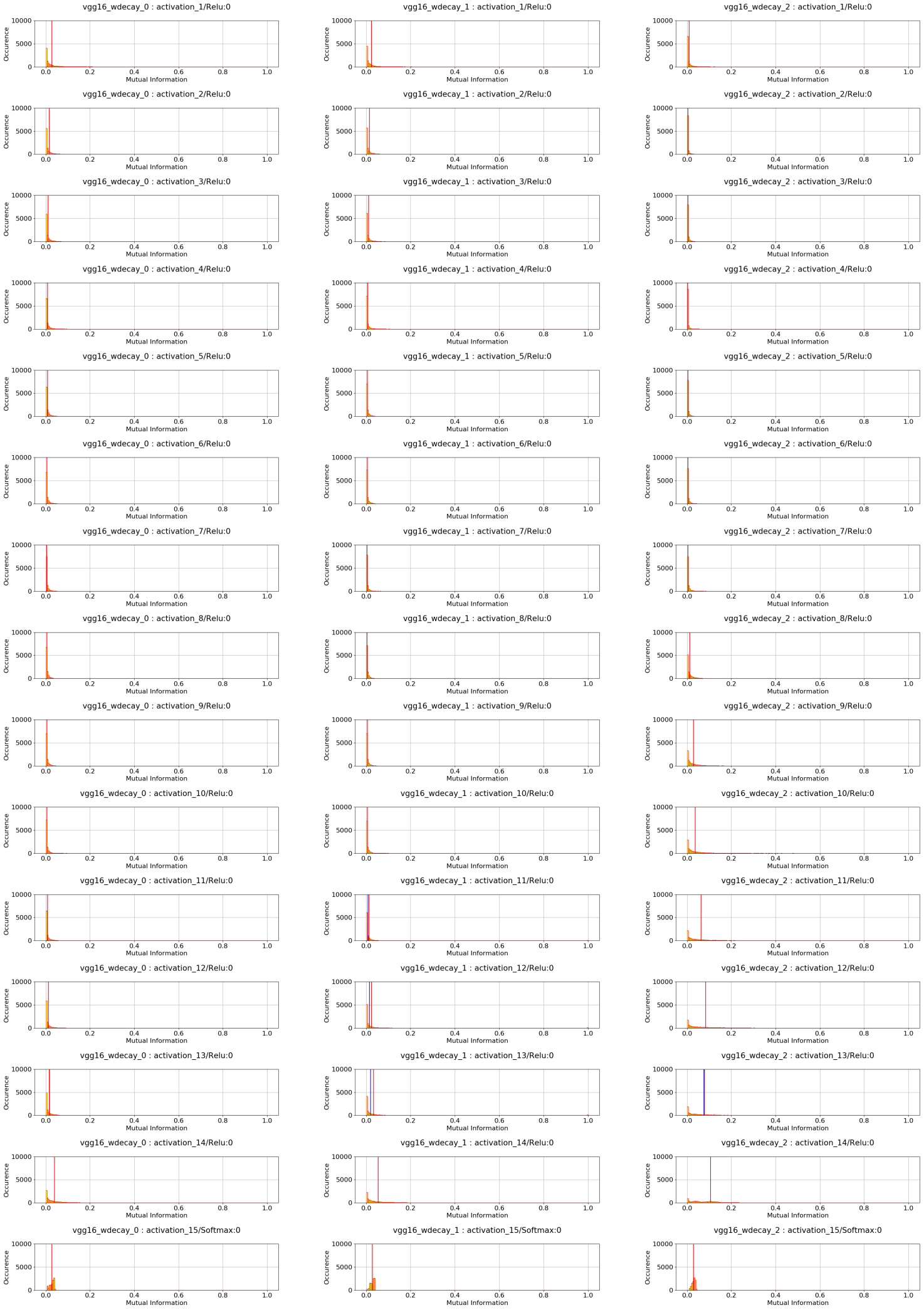
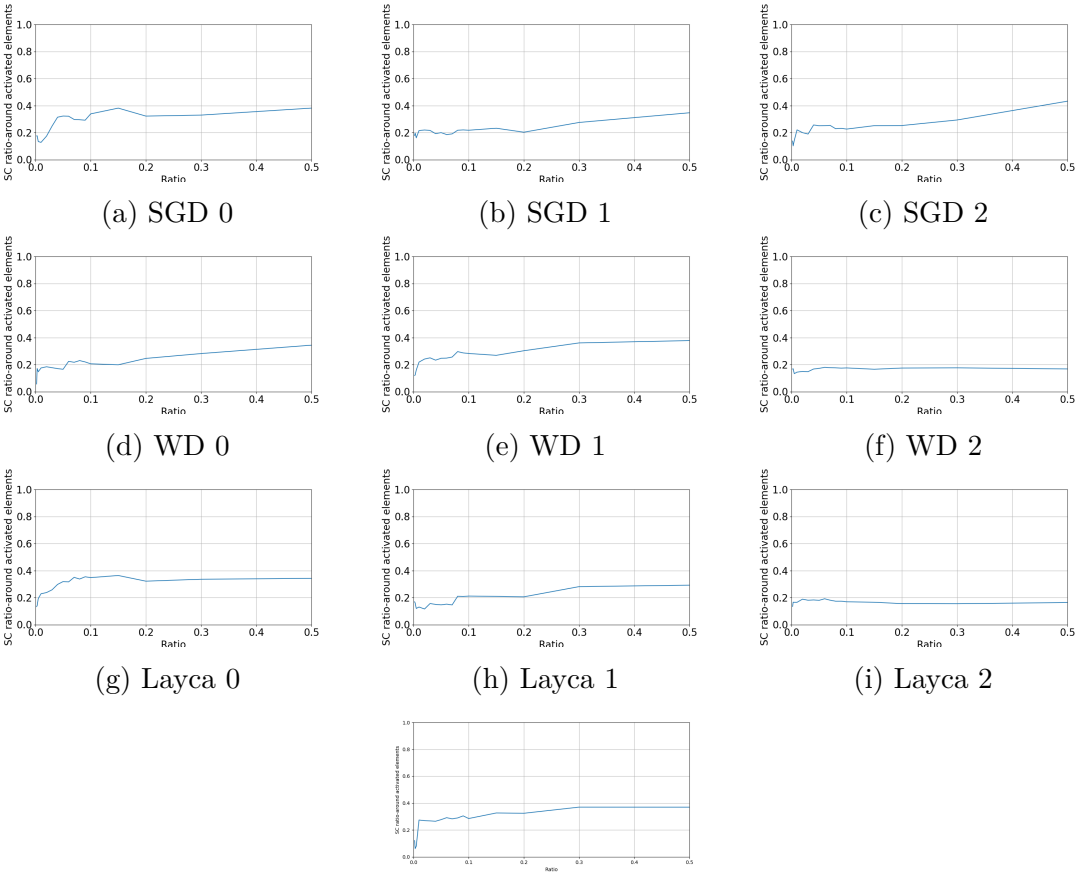


FIGURE A.13 – MI histogramme pour WD (65%,75% et 85%)

FIGURE A.14 – MI histogramme pour Layca (65%,75% et 85%)

Ratio Around



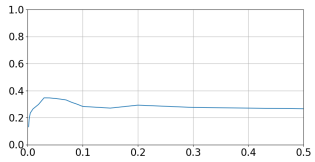
(a) SGD 0 (b) SGD 1 (c) SGD 2

(d) WD 0 (e) WD 1 (f) WD 2

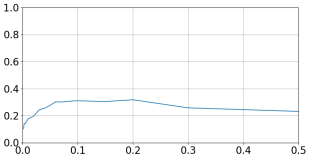
(g) Layca 0 (h) Layca 1 (i) Layca 2

(j) Initial

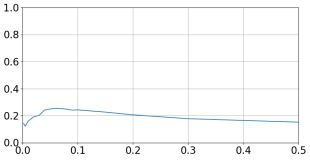
FIGURE A.15 – Ratio Around pour la couche 1



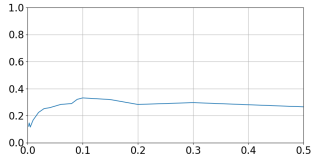
(a) SGD 0



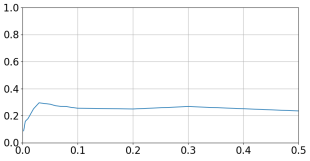
(b) SGD 1



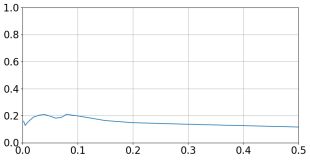
(c) SGD 2



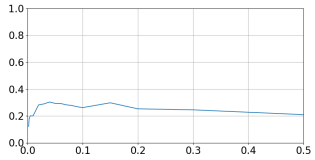
(d) WD 0



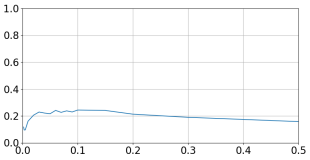
(e) WD 1



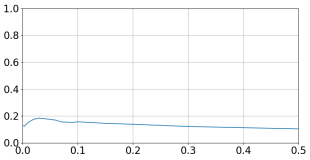
(f) WD 2



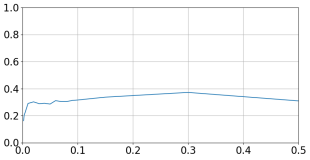
(g) Layca 0



(h) Layca 1

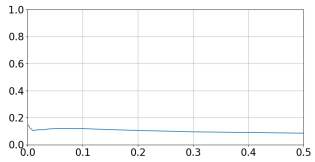


(i) Layca 2

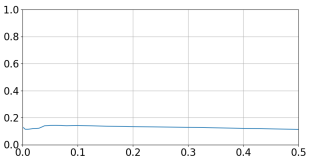


(j) Initial

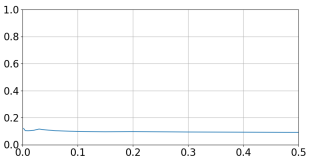
FIGURE A.16 – Ratio Around pour la couche 2



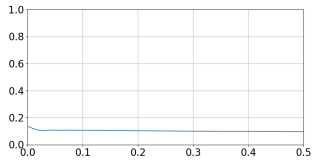
(a) SGD 0



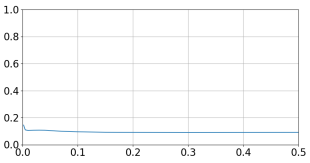
(b) SGD 1



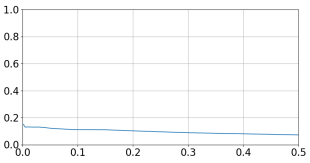
(c) SGD 2



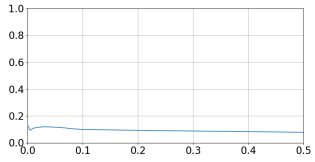
(d) WD 0



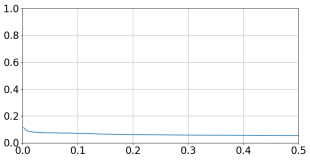
(e) WD 1



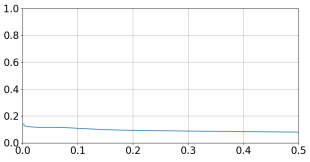
(f) WD 2



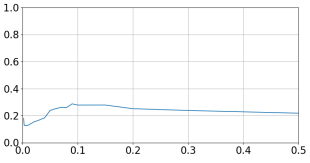
(g) Layca 0



(h) Layca 1

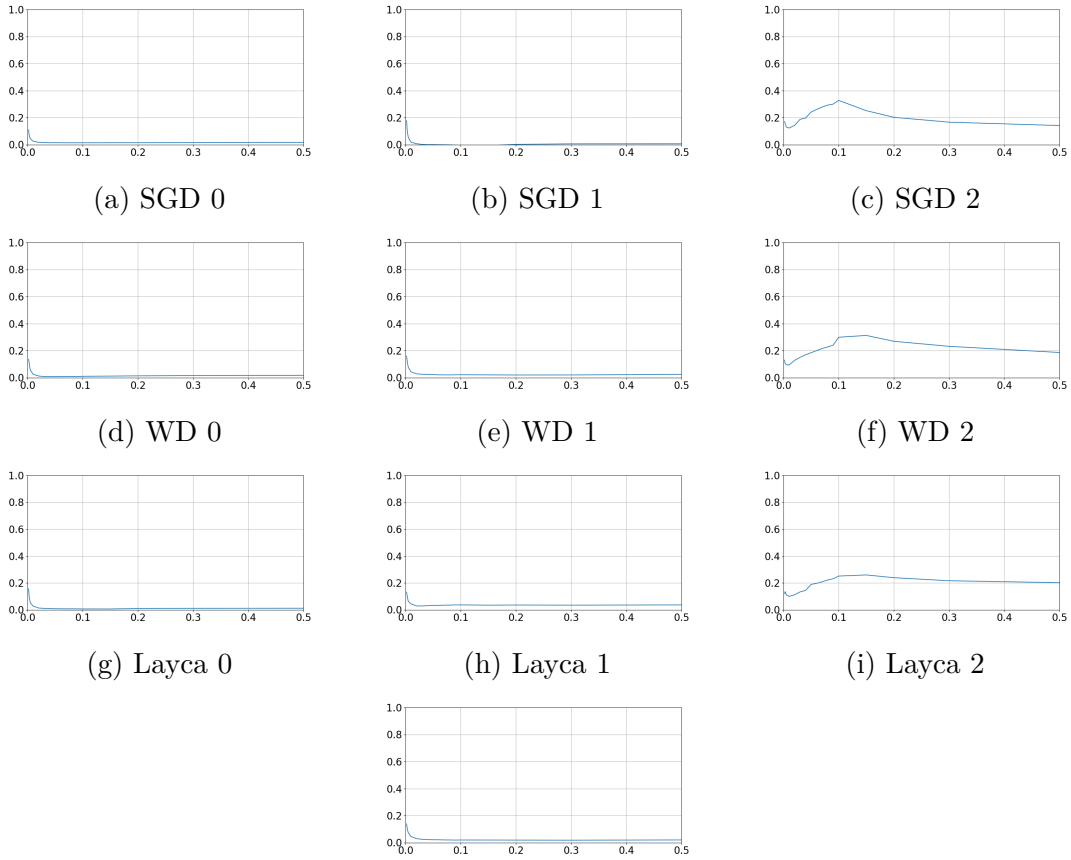


(i) Layca 2



(j) Initial

FIGURE A.17 – Ratio Around pour la couche 3



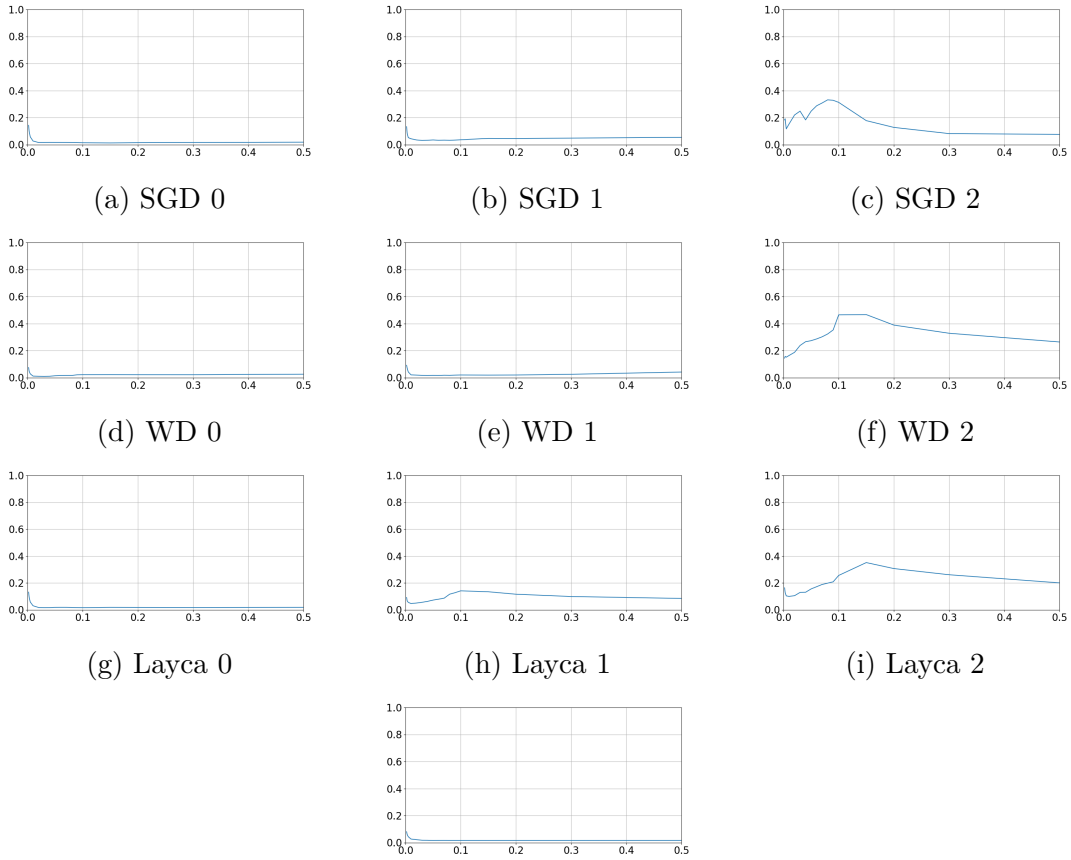
(a) SGD 0 (b) SGD 1 (c) SGD 2

(d) WD 0 (e) WD 1 (f) WD 2

(g) Layca 0 (h) Layca 1 (i) Layca 2

(j) Initial

FIGURE A.18 – Ratio Around pour la couche 11



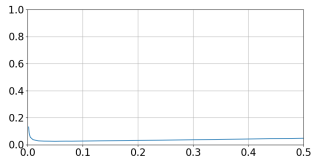
(a) SGD 0 (b) SGD 1 (c) SGD 2

(d) WD 0 (e) WD 1 (f) WD 2

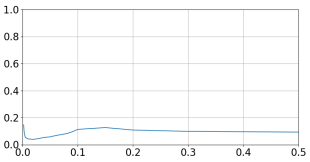
(g) Layca 0 (h) Layca 1 (i) Layca 2

(j) Initial

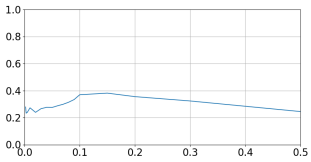
FIGURE A.19 – Ratio Around pour la couche 12



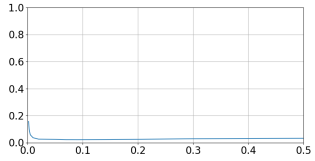
(a) SGD 0



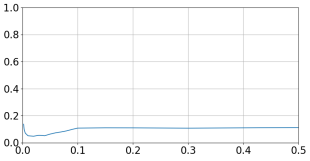
(b) SGD 1



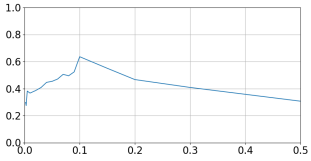
(c) SGD 2



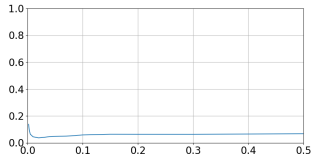
(d) WD 0



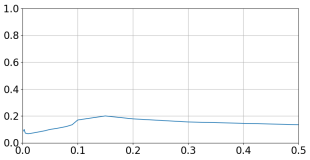
(e) WD 1



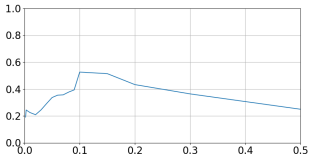
(f) WD 2



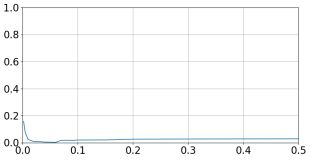
(g) Layca 0



(h) Layca 1



(i) Layca 2



(j) Initial

FIGURE A.20 – Ratio Around pour la couche 13

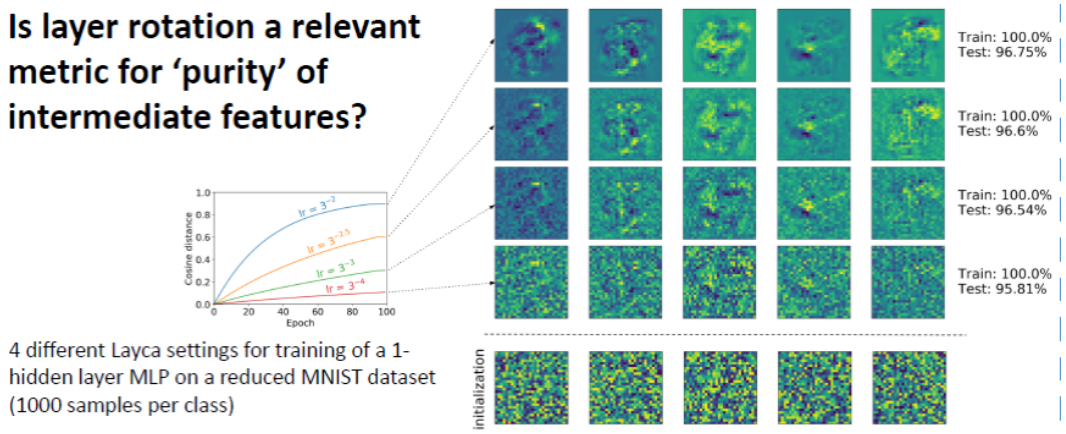


FIGURE A.21 – Courtoisie de Simon (juin 2019) bientôt publiée, montrant une certaine pureté dans les modèles qui généralisent mieux

A.1 Code

Le code est disponible sur github à l'adresse suivante : `https://github.com/Corvinus96/MasterThesis/`

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**