كلية العلوم و التقنيات فاس
ⵜⴰⵖⵙⵓⵍⵜ ⵏ ⵜⵇⴰⵙⵙⴰⵢⵉⵏ ⴷ ⵜⵓⵙⵙⵙⵜⵉ
Faculté des Sciences et Techniques de Fès

جامعة سيدي محمد بن عبد الله
ⵜⴰⵙⴷⴰⵡⵜ ⵙⵉⴷⵉ ⵎⵓⵃⵎⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ
Université Sidi Mohamed Ben Abdellah

**DEPARTEMENT DES MATHEMATIQUES**

**Master Mathématiques et Applications aux Calculs Scientifiques**

**(MACS)**

**MEMOIRE DE FIN D'ETUDES**

**Pour l'obtention du Diplôme de Master Sciences et Techniques (MST)**

# Apprentissage par transfert dans les réseaux profonds : Application à la classification d'images

Réalisé par : **MALLOUK Otmane**

Encadré par : **Pr. ETTAOUIL Mohamed**
**Pr. JOUDAR Nour-Eddine**

**Soutenu le 19 Juillet 2022**

**Devant le jury composé de :**

| | |
|---|---|
| **- Pr. CHAIBI Ghizlane** | **FST Fès** |
| **- Pr. ETTAOUIL Mohamed** | **FST Fès** |
| **- Pr. EL KHAOULANI EL IDRISSI Rachid** | **FST Fès** |
| **- Pr. JOUDAR Nour-Eddine** | **ENSAM Rabat** |

**Année Universitaire 2021 / 2022**

# Table des matières

# Table des figures

# Liste des tableaux

# Remerciement

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidée lors de la rédaction de ce mémoire.

Je voudrais dans un premier temps remercier, mes directeurs de mémoire, Ettaouil Mohamed professeur de mathématiques appliquées à la Faculté des Sciences et Techniques de Fès, pour sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion, et Joudar Nour-eddine professeur à l'École Nationale Supérieur des Arts et Métiers de Rabat pour m'avoir accordé des entretiens et avoir répondu à mes questions. Il été d'un grand soutien dans l'élaboration de ce mémoire. Il a partagé ses connaissances et expériences dans ce milieu, tout en m'accordant sa confiance et une large indépendance dans l'exécution de missions valorisantes. Ainsi que pour m'avoir relu et corrigé mon mémoire. Ses conseils de rédaction ont été très précieux.

Je remercier également Les membres du jury : Monsieur EL KHAOULANI EL IDRISSI Rachid, professeur à la Faculté des sciences et techniques de Fès et Madame CHAIBI Ghizlane, professeur à la Faculté des sciences et techniques de Fès, qui m'ont honoré en acceptant d'évaluer et de juger mon travail.

Je tiens à témoigner toute ma reconnaissance aux personnes suivantes, pour leur aide dans la réalisation de ce mémoire : Je remercie mes très chers parents Belkacem et Aicha, qui ont toujours été là pour moi. Je remercie ma sœur Fatima, et mes frères Brahim et Mohammed, pour leurs encouragements.

Enfin, je remercie mes amis Ahlam et Lahcen qui ont toujours été là pour moi. Leur soutien inconditionnel et leurs encouragements ont été d'une grande aide.

À tous ces intervenants, je présente mes remerciements, mon respect et ma gratitude.

# Dédicace

Je dédie ce travail

A mes parents qui m'ont soutenu et encouragé durant ces années d'études.

Qu'ils trouvent ici le témoignage de ma profonde reconnaissance.

A mes frères, ma sœur, mes grands parents et ceux qui ont partagé avec moi tous les moments d'émotion lors de la réalisation de ce travail.

Ils m'ont chaleureusement supporté et encouragé tout au long de mon parcours.

A ma famille, mes proches et à ceux qui me donnent de l'amour et de la vivacité.

A tous mes amis qui m'ont toujours encouragé, et à qui je souhaite plus de succès.

A tous ceux que j'aime.

Merci !

# Introduction générale

Depuis les années 1950, un petit groupe d'intelligence artificielle (IA) [17], souvent appelé Apprentissage automatique(Machine Learning)[40],[63],[8] a révolutionné un certain nombre de domaines au cours des dernières décennies. Les réseaux de neurones (NN)[20],[45],[25] sont un sous-domaine de ML, et ce dernier qui a engendré l'apprentissage profond(Deep Learning) [33],[42],[36], [68],[22]. Les réseaux de neurones convolutifs [19],[24],[29] est l'un des réseaux de neurones les plus représentatifs dans le domaine de l'apprentissage profond. La vision par ordinateur [49] basée sur des réseaux de neurones convolutifs a permis aux gens d'accomplir ce qui était considéré comme impossible au cours des derniers siècles, comme la reconnaissance faciale[5], les véhicules autonomes[31], les supermarchés en libre-service[50] et les traitements médicaux intelligents[51]. En revanche, le scénario idéal de l'apprentissage automatique est qu'il existe de nombreuses instances d'apprentissage étiquetées[10], qui ont la même distribution que les données de test. Cependant, la collection des données d'apprentissage suffisantes est souvent coûteuse, chronophage ou même irréaliste dans de nombreux scénarios. L'apprentissage semi-supervisé[69] peut en partie résoudre ce problème en assouplissant le besoin de données étiquetées en masse. En règle générale, une approche semi-supervisée ne nécessite qu'un nombre limité de données étiquetées et utilise une grande quantité de données non étiquetées pour améliorer la précision de l'apprentissage, mais dans de nombreux cas, les instances non étiquetées sont également difficiles à collecter, ce qui rend généralement les modèles traditionnels résultants insatisfaisants.

L'insuffisance des données d'apprentissage est un problème incontournable dans certains domaines particuliers[12],[21]. La collection des données est complexe et coûteuse, ce qui rend extrêmement difficile la création d'un ensemble de données annotées à grande échelle et de haute qualité. Par exemple, chaque échantillon dans l'ensemble de données bio-informatiques démontre souvent un essai clinique ou un patient douloureux. De plus, même si nous obtenons un ensemble de données d'entraînement en payant un prix élevé, il est très facile de le rendre obsolète et ne peut donc pas être appliqué efficacement dans les nouvelles tâches.

L'apprentissage par transfert[41],[58],[70],[37] assouplit l'hypothèse selon laquelle les données d'entraînement doivent être indépendantes et distribuées de manière identique (i.i.d.)[67] avec les données de test, ce qui nous motive à utiliser l'apprentissage par transfert pour contrer le problème des données d'entraînement insuffisantes.

Dans l'apprentissage par transfert, les données d'apprentissage et les données de test ne doivent pas nécessairement être i.i.d., et le modèle dans le domaine cible n'a pas besoin d'être formé à partir de zéro, ce qui peut réduire considérablement la demande de données d'apprentissage et le temps d'apprentissage dans le domaine cible.
Notre projet est organisé comme suit :

1. Le premier chapitre est consacré à l'étude de quelques algorithmes d'apprentissage dans les réseaux profonds, ainsi que les caractéristiques de chaque algorithme. Nous allons aborder aussi les réseaux de neurones convolutifs(CNNs) ainsi que leur avantage par rapport aux réseau de neurones artificiels entièrement connectés dans le cadre de traitement des images.

2. Les modèles de réseau neuronal (NN) sont bien adaptés aux domaines où de grands ensembles de données étiquetées sont disponibles, car leur capacité peut facilement être augmentée en ajoutant plus de couches ou plus d'unités dans chaque couche. Cependant, les grands réseaux avec des millions ou des milliards de paramètres peuvent facilement sur-ajusté(Overfitting). En conséquence, des méthodes de régularisation des réseaux de neurones ont été développées, et qui feront l'objet du deuxième chapitre.
3. Le troisième chapitre qui est le noyau de ce projet où nous allons étudier l'apprentissage par transfert. Ce type d'apprentissage a été introduit essentiellement pour résoudre le problème de manque des données ainsi pour assouplit l'hypothèse selon laquelle les données d'apprentissage doivent être indépendantes et distribuées de manière identique (i.i.d.) avec les données de test.
4. Dans le quatrième chapitre, nous effectuerons des expériences visant à illustrer et justifier l'importance de l'apprentissage par transfert et nous verrons que cela résout un gros problème, surtout en ce qui concerne l'imagerie médicale lorsqu'il y a un réel manque de données, particulièrement en cas de nouvelle épidémie.

Chapitre 1

# Réseaux de neurones profonds

## 1.1 Introduction et Notations

Les réseaux de neurones artificiels(ANN) sont des algorithmes inspirés de la biologie, capables d'apprendre à réaliser une tache (classification,régression...). Un réseau de neurones est composé d'un ensemble de couches successives. Chaque couche est composée des neurones artificiels[27]. Un neurone de la $k^{eme}$ couche est connecté à tous les neurones de la $(k-1)^{eme}$ couche(FCANN). Entrainer un réseau revient alors à apprendre les poids synaptiques et les biais de chaque neurone.

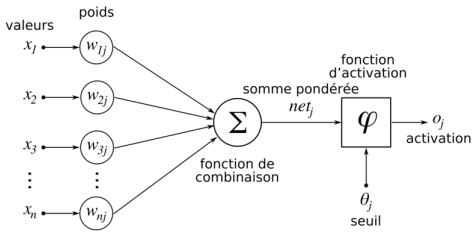



FIGURE 1.1 – Fonctionnement d'un neurone artificiel

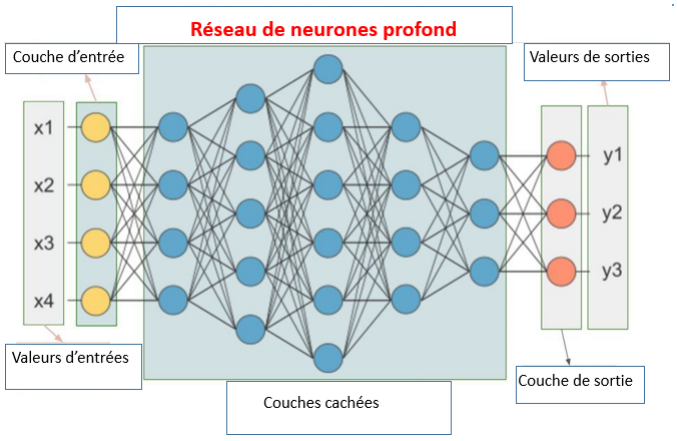



FIGURE 1.2 – Architecture d'un réseau profond

Étant donnée une base d'apprentissage

$$X = \{(x_1, d_1), (x_2, d_2), ..., (x_n, d_n)\}$$

avec :
$x_i$ est l'entrée $i$.
$d_i$ est la sortie attendue(désirée) associe à $x_i$.
On veut minimiser l'erreur quadratique(SE) :

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(d_i - y_i)^2 = \frac{1}{2}\sum_{i=1}^{n} E_i \tag{1.1}$$

avec :
$E_i = (d_i - y_i)^2$ est l'erreur pour l'entrée $x_i$(appelé aussi l'erreur local).
$y_i$ est la sortie produite par le réseau pour l'exemple(la donnée) $i$.
$W$ est la matrice des poids synaptiques du réseau.

**Remarque 1.1.1:**

Ici on a choisi l'erreur quadratique. Il existe différentes formules pour calculer l'erreur entre la sortie attendue $d_i$ et la sortie produite par le réseau $y_i = F(x_i)$. Par exemple, l'erreur absolue moyenne donnée par :

$$E(w) = \frac{1}{n}\sum_{i=1}^{n}|d_i - y_i| \tag{1.2}$$

Le choix de la fonction erreur dépend du problème, par exemple si la sortie produite est une probabilité $\tilde{y}_i$ avec $0 \leq \tilde{y}_i \leq 1$, Alors la fonction erreur adaptée est l'entropie croisée binaire

$$E(w) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \ln(\tilde{y}_i) + (1 - \tilde{y}_i)\ln(1 - \tilde{y}_i)) \tag{1.3}$$

## 1.2 Apprentissage et optimisation dans les réseaux profonds

### 1.2.1 Descente de gradient

L'objectif de la méthode de descente de gradient est de trouver un minimum d'une fonction de plusieurs variables le plus rapidement possible. L'idée est très simple, on sait que le vecteur opposé au gradient indique une direction vers des plus petites valeurs de la fonction, il suffit donc de suivre d'un pas cette direction et de recommencer. Cependant, afin d'être encore plus rapide, il est possible d'ajouter plusieurs paramètres qui demandent pas mal d'ingénierie pour être bien choisis.
Imaginons une goutte d'eau en haut d'une colline. La goutte d'eau descend en suivant la ligne de plus grande pente et elle s'arrête lorsqu'elle atteigne un point bas. C'est exactement ce que fait la descente de gradient : Partant d'un point sur une surface, on cherche la pente la plus grande en calculant le gradient et on descend d'un petit pas, on recommence à partir du nouveau point jusqu'à atteindre un minimum local.

1. Descente du gradient classique(cgd). [48] On suppose que la fonction erreur $E$ est différentiable.

**Algorithm 1** Descente de gradient classique

**ENTRÉES:** Un point initial $w_0$.

**ENTRÉES:** Un niveau d'erreur $\epsilon > 0$.

•**Itération.** On calcule une suite de points $w_1, w_2, ..., w_k, ...$ par récurrence de la façon suivante. Supposons que l'on ait déjà obtenu le point $w_k$ :

(a) On calcule $\nabla E(w_k)$

(b) On choisit un pas $\delta_k$ et on calcule :

$$w_{k+1} \leftarrow w_k - \delta_k \nabla E(w_k)$$

•**Arrêt.**On s'arrête lorsque $||\nabla E(w_k)|| < \epsilon$

**Remarque 1.2.1**

(a) Évidemment, plus on choisit le point initial $w_0$ proche d'un minimum local, plus l'algorithme va aboutir rapidement. Cependant, comme on ne sait pas où est ce minimum local (c'est ce que l'on cherche), le plus simple est de choisir un $w_0$ au hasard.

(b) Le choix du pas $\delta_k$ est crucial. On sait que l'on peut choisir $\delta_k$ assez petit de façon à avoir $E(w_{k+1}) < E(w_k)$ car dans la direction de $-\nabla E(w_k)$ la fonction $E$ décroît. On peut fixer à l'avance un pas $\delta$ commun à toutes les itérations, par exemple $\delta = 0.01$. On pourrait également tester à chaque itération plusieurs valeurs de $\delta$ par balayage ($\delta = 0.001$, puis $\delta = 0.002$...) et choisir pour $\delta_k$ celui en lequel $E$ prend la plus petite valeur.

2. Descente du gradient stochastique(sgd). [48] La descente de gradient stochastique est une façon d'optimiser les calculs de la descente de gradient pour une fonction d'erreur associée à une grande série de données. Au lieu de calculer un gradient compliqué et un nouveau point pour l'ensemble des données, on calcule un gradient simple et un nouveau point par donnée. SGD sélectionne au hasard un point de données dans l'ensemble de données à chaque itération.
Pour minimiser l'erreur et déterminer les meilleurs paramètres, on peut appliquer la méthode du gradient classique. Pour appliquer cette formule, il faut calculer des gradients $\nabla E(w_k)$, or

$$\nabla E(w_k) = \sum_{i=1}^{n} \nabla E_i(w_k) \tag{1.4}$$

Il faut donc calculer une somme de $n$ termes à chaque itération, ce qui pose des problèmes d'efficacité pour de grandes valeurs de $n$.
Pour diminuer la quantité de calculs, l'idée est de considérer à chaque itération un seul gradient $E_i$ à la place de $E$. C'est-à-dire :

$$w_{k+1} \leftarrow w_k - \delta_k \nabla E_i(w_k) \tag{1.5}$$

pour une seule erreur $E_i$ (correspondant à la donnée numéro $i$). L'itération suivante se basera sur l'erreur $E_{i+1}$.
**Quel est l'intérêt de cette méthode ?**
Dans la méthode de gradient classique, on calcule à chaque itération un « gros » gradient (associe à la totalité des n données) qui nous rapproche d'un grand pas vers le minimum. Ici on calcule n « petits » gradients qui nous rapprochent du minimum.

3. Descente par lot(mini-batch). [48] Il existe une méthode intermédiaire entre la descente de gradient classique (qui tient compte de toutes les données à chaque itération) et la descente de gradient stochastique (qui n'utilise qu'une seule donnée à chaque itération).
La descente de gradient par **lots** (**mini-lots, mini-batch**) est une méthode intermédiaire : on divise les données par paquets de taille $m$. Pour chaque paquet (appelé « lot(batch) »), on calcule un gradient et on effectue une itération.
Au bout de $n/m$ itérations, on a parcouru tout le jeu de données : cela s'appelle une **époque**(epoche).
La formule est donc :

$$w_{k+1} \leftarrow w_k - \delta_k \nabla (E_{j_0+1} + E_{j_0+2} + ... + E_{j_0+m})(w_k) \tag{1.6}$$

Pour $w_{k+2}$, on part de $w_{k+1}$ et on utilise le gradient de la fonction

$$E_{j0+m+1} + E_{j0+m+2} + ... + E_{j0+2m} \tag{1.7}$$

**Remarque 1.2.2**

(a) Pour $m = 1$, c'est exactement la descente de gradient stochastique. Pour $m = n$, c'est la descente de gradient classique.

(b) Cette méthode combine le meilleur des deux mondes : la taille des données utilisées à chaque itération peut être adaptée à la mémoire et le fait de travailler par lots évite les pas erratiques de la descente stochastique pure.

### 1.2.2 Accélération(Moment, Nesterov)

Le choix du pas $\delta$ n'est pas la seule amélioration possible de la méthode du gradient, nous allons voir comment la modifier à l'aide du « moment ». Commençons par revenir à l'analogie de la descente du gradient classique qui correspond à une goutte d'eau qui descend une montagne : la goutte emprunte le chemin qui suit la courbe de plus grande pente, quitte à serpenter et osciller lors de la descente. Imaginons que l'on lance maintenant une balle assez lourde du haut de la même montagne. Cette balle va suivre, comme la goutte d'eau, le chemin de la plus forte pente, mais une fois lancée elle va acquérir de l'inertie, appelée moment, qui va atténuer ses changements de direction. Ainsi la balle ne s'embarrasse pas des petits aléas du terrain et dévale la pente plus rapidement que la goutte d'eau.
Nos petits aléas de terrain nous viennent du fait que l'on ne calcule pas exactement le gradient de la fonction d'erreur en utilisant tout le jeu de données à chaque fois, mais seulement un échantillon. Cela peut conduire à certains gradients mal orientés. L'inertie de la balle est en quelque sorte la mémoire de la trajectoire passée qui corrige les mouvements erratiques.

1. **Moment(Momentum SGD)**
L'objectif principal de la méthode dite de Moment [48] est d'accélérer le processus de descente de gradient, et ceci en rajoutant un vecteur de vélocité.
La formule de la descente de gradient avec moment est :

$$v_{k+1} \leftarrow \mu v_k - \delta \nabla E(w_k)$$
$$w_{k+1} \leftarrow w_k + v_{k+1}$$

où $\delta, \mu \in \mathbb{R}$. Le vecteur $v_{k+1}$ est calculé au début de chaque itération et représente la mise à jour de la vélocité d'une "balle dévalant une pente".

2. **Nesterov**
   Dans la méthode précédente, le moment et le gradient sont calculés au même point $w_k$. La méthode de Nesterov [48] est une variante de cette méthode. Elle consiste à appliquer d'abord le moment, pour obtenir un point $w'_k$, puis de calculer le gradient en ce point (et non en $w_k$).
   La formule est donc :

$$\begin{aligned} w'_k &\leftarrow w_k + \mu v_k \\ v_{k+1} &\leftarrow \mu v_k - \delta \nabla E(w'_k) \\ w_{k+1} &\leftarrow w_k + v_{k+1} \end{aligned}$$

   C'est un petit avantage par rapport à la méthode du moment puisqu'on calcule le gradient au point $w'_k$ qui est censé être plus près de la solution $w_{min}$ que $w_k$.

## 1.3 Extensions de la descente de gradient

Il existe plusieurs variantes de l'algorithme de descente de gradient. Dans la suite, nous présentons trois méthodes qui sont Adagrad, RMSProp, Adam.

1. **Adagrad.** [48] Le principe de la méthode Adagrad est de faire que le taux d'apprentissage s'adapte aux paramètres, faisant de sorte qu'il s'ajuste automatiquement, en fonction de "l'éparsité" des paramètres. Adagrad abaisse progressivement le taux d'apprentissage mais pas de la même manière pour tous les paramètres : les dimensions à pente plus prononcée voient leur taux abaissé plus rapidement que celles à pente douce. Plus formellement, le pas est décrit par :

$$\forall i, (w_{k+1})_i \leftarrow (w_k)_i - \alpha \frac{(\nabla E(w_k))_i}{\sqrt{\sum_{j=1}^{k} (\nabla E(w_j))_i^2}}, \ \alpha > 0$$

2. **RMSProp.** [48] Cet algorithme ajuste automatiquement le taux d'apprentissage à chaque paramètre, comme Adagrad. Cependant, il ne cumule que les gradients issus des itérations récentes. Pour cela, il utilise une moyenne glissante :

$$\begin{aligned} \forall i, (\nabla_{k+1})_i &\leftarrow \delta(\nabla_k)_i - (1-\delta)(\nabla E(w_k))_i^2 \\ \forall i, (w_{k+1})_i &\leftarrow (w_k)_i - \alpha \frac{(\nabla E(w_k))_i}{\sqrt{(\nabla_{k+1})_i}}, \ \alpha > 0 \end{aligned}$$

   Adagrad modifie le taux d'apprentissage à chaque itération $k$ pour chaque paramètre $w_i$. Ici, $\delta(\nabla_k)_i$ est la moyenne quadratique glissante du gradient. La division du gradient de la fonction objective par la racine de la moyenne quadratique glissante (c'est à dire l'amplitude) améliore la convergence.

3. **Adam.** Adam [48] est l'un des algorithmes les plus récents et les plus efficaces pour l'optimisation par descente de gradient. Le principe est le même que pour Adagrad et RMSProp : il adapte automatiquement le taux d'apprentissage pour chaque paramètre. Sa particularité est de calculer $(m_k, v_k)$ des "estimations adaptatives des moments". Il

peut donc être vu comme une généralisation de l'algorithme Adagrad :

$$\forall i, (m_{k+1})_i \leftarrow \beta_1 (m_k)_i + (1-\beta_1)(\nabla E(w_k))_i^2$$
$$\forall i, (v_{k+1})_i \leftarrow \beta_2 (v_k)_i + (1-\beta_2)(\nabla E(w_k))_i^2$$
$$\forall i, (w_{k+1})_i \leftarrow (w_k)_i - \alpha \frac{\sqrt{1-\beta_2}}{1-\beta_1} \frac{(m_k)_i}{\sqrt{(v_k)_i + \epsilon}}$$
$$\alpha, \epsilon > 0, \beta_1, \beta_2 \in ]0,1[$$

Ici $m_k$ est le premier moment du gradient (la moyenne) et $v_k$ est son second moment (variance non-centrée). $\epsilon$ est un paramètre de précision. Les paramètres $\beta_1$ et $\beta_2$ sont utilisés pour réaliser des moyennes d'exécution sur les moments $m_k$ et $v_k$ respectivement.

## 1.4 Rétropagation

Les poids dans le réseau de neurones sont au préalable initialisés avec des valeurs aléatoires. On considère ensuite la base d'apprentissage $X$ définie précédemment. L'algorithme de rétropagation du gradient[30] suit les étapes suivantes :

1. Soient un échantillon $x_i$ que l'on présente à l'entrée du réseau de neurones et $d_i$ la sortie désirée associe à échantillon.

2. On propage le signal en avant dans les couches du réseau de neurones $x_k^{(n-1)} \rightarrow x_j^{(n)}$ avec $n$ le numéro de la couche, et $k$ et $j$ les numéros des neurones sur leur couche respective.

3. La propagation vers l'avant se calcule à l'aide de la fonction d'activation $g$, de la fonction d'agrégation $h$ (souvent un produit scalaire entre les poids et les entrées du neurone voir 1.1) et des poids synaptiques $w_{jk}$ entre le neurone $x_k^{(n-1)}$ et le neurone $x_j^{(n)}$ tel que :

$$x_j^{(n)} = g^{(n)}(h_j^{(n)}) = g^{(n)}(\sum_k w_{jk}^{(n)} x_k^{(n-1)})$$

4. Lorsque la propagation vers l'avant est terminée, on obtient à la sortie le résultat $y_i$.

5. On calcule alors l'erreur entre la sortie donnée par le réseau $y_i$ et le vecteur $d_i$ désiré à la sortie pour cet échantillon. Pour chaque neurone $i$ dans la couche de sortie, on calcule :

$$e_i^{sortie} = g'(h_i^{sortie})(y_i - d_i)$$

6. On propage l'erreur vers l'arrière $e_i^{(n)} \rightarrow e_j^{(n-1)}$, grâce à la formule :

$$e_j^{(n-1)} = g'^{(n-1)}(h_j^{(n-1)}) \sum_i w_{ij}^{(n)} e_i^{(n)}$$

ou

$$e_i^{(n)} = e_i^{(sortie)} = (y_i - d_i) \frac{\partial y_i}{\partial h_i^{(n)}}$$

7. On met à jour les poids dans toutes les couches :

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \delta e_i^{(l)} x_j^{(l-1)}$$

*Démonstration.* Soit $E$ l'erreur(quadratique) à minimiser :

$$E(w) = \frac{1}{2}\sum_i (y_i - d_i)^2$$

En utilisant la règle de dérivation en chaine :

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial h}\frac{\partial h}{\partial w}$$

on a :

$$\begin{aligned}\frac{\partial E}{\partial w_{ab}^{(l)}} &= \sum_i (y_i - d_i) g'^{(n)}(h_i^{(n)}) \sum_k w_{ik}^{(n)} \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}} \\ &= \sum_k \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}} \sum_k w_{ik}^{(n)} g'^{(n)}(h_i^{(n)})(y_i - d_i) \\ &= \sum_k \frac{\partial x_k^{(n-1)}}{\partial w_{ab}^{(l)}} \sum_k w_{ik}^{(n)} e_i^{(n)}\end{aligned}$$

$$avec \quad e_i^{(n)} = g'^{(n)}(h_i^{(n)})(y_i - d_i)$$

En utilisant la même technique sur la dérivée partielle de $x_k^{(n-1)}$, on obtient :

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_k \frac{\partial x_k^{(n-2)}}{\partial w_{ab}^{(l)}} \sum_k w_{ik}^{(n-1)} e_i^{(n-1)}$$

En itérant l'algorithme jusqu'à la couche $l$, on arrive à :

$$\frac{\partial E}{\partial w_{ab}^{(l)}} = \sum_k \frac{\partial x_k^{(l)}}{\partial w_{ab}^{(l)}} \sum_k w_{ik}^{(l+1)} e_i^{(l+1)} = x_b^{(l-1)} e_a^{(l)}$$

□

## 1.5 Réseaux de neurones convolutifs

L'architecture des ANN discutée précédemment est appelée réseaux de neurones FC (FCNN). La raison est que chaque neurone d'une couche $i$ est connecté à tous les neurones des couches $i-1$ et $i+1$. Chaque connexion entre deux neurones a deux paramètres : le poids et le biais. Ajouter plus de couches et de neurones augmente le nombre de paramètres. Par conséquent, l'apprentissage de tels réseaux prend beaucoup de temps, même sur des appareils dotés de plusieurs unités de traitement graphique (GPU) et de plusieurs unités centrales de traitement (CPU). Il devient impossible de former de tels réseaux sur des PC avec des capacités de traitement et de mémoire limités.

Dans l'analyse de données multidimensionnelles telles que les images, les CNN (également connus sous le nom de ConvNets) sont plus efficaces en termes de temps et de mémoire que les réseaux FC.

ANN est la base du CNN, avec quelques modifications ajoutées pour le rendre adapté à l'analyse de grandes quantités de données. La connexion de tous les neurones augmente le nombre de paramètres même lors de l'analyse de très petites images (par exemple, une de $150 \times 150$ pixels). La couche d'entrée dans ce cas aura 22500 neurones. La connecter à un autre couche cachée avec 500 neurones, le nombre de paramètres requis est de $22500 \times 500 = 11250000$. Les applications du monde réel peuvent fonctionner avec des images à haute dimension où la plus petite dimension

peut avoir 1000 pixels et plus.
Pour une image d'entrée de taille $1000 \times 1000$ et une couche cachée de 2000 neurones, le nombre de paramètres est égal à 2 milliards. Notez que l'image d'entrée est grise.

### 1.5.1 Algèbre de convolution

Cette section est consacrée à une étude théorique dans laquelle nous présenterons quelques définitions et propriétés mathématiques qui forment la base des réseaux de neurones convolutifs.

**Théorème 1.5.1: [64]**

Soit $f, g \in L^1(\mathbb{R}^d)$ , pour presque tout $x \in \mathbb{R}^d$, $y \mapsto f(y)g(x-y)$ est intégrable et la fonction :

$$(f \star g)(x) := \int_{\mathbb{R}^d} f(y)g(x-y)dy \tag{1.8}$$

est elle même intégrable sur $\mathbb{R}^d$, de plus :

$$||f \star g||_1 < ||f||_1||g||_1$$

*Démonstration.* Par le théorème de Tonelli[9], $f \otimes g : (x, y) \mapsto f(y)g(x)$ est intégrable sur $\mathbb{R}^d \times \mathbb{R}^d$ et $||f \otimes g||_{L^1(\mathbb{R}^d \times \mathbb{R}^d)} = ||f||_{L^1(\mathbb{R}^d)}||g||_{L^1(\mathbb{R}^d)}$.
L'application :

$$\phi : (x, y) \mapsto (x - y, y)$$

est un $\mathcal{C}^1-$difféomorphisme de $\mathbb{R}^d \times \mathbb{R}^d$de jacobien 1, donc par le théorème de changement de variable :

$$||f \otimes g||_{L^1(\mathbb{R}^d \times \mathbb{R}^d)} = ||(f \otimes g) \circ \phi||_{L^1(\mathbb{R}^d \times \mathbb{R}^d)} \tag{1.9}$$

$$= \int_{\mathbb{R}^d \times \mathbb{R}^d} |f(y)g(x-y)|dxdy \tag{1.10}$$

Donc, la fonction $y \mapsto f(y)g(x-y)$ est Lebesque intégrable sur $\mathbb{R}^d \times \mathbb{R}^d$ et la conclusion est une conséquence directe du théorème de Fubini. □

**Définition 1.5.1: [64]**

Le produit de convolution de deux fonctions réelles ou complexes $f$ et $g$, est une autre fonction, qui se note $f \star g$ et qui est définie par :

$$(f \star g)(x) := \int_{\mathbb{R}^d} f(y)g(x-y)dy \tag{1.11}$$

**Proposition 1.5.1: [64]**

le produit de convolution est associatif et commutatif.

*Démonstration.* Soit $f, g \in L^1(\mathbb{R}^d)$
• **commutativité** soit $N$ un négligeable tel que :

$$y \mapsto f(y)g(x-y) \quad et \quad y \mapsto g(y)f(x-y)$$

soient intégrables pour tout $x \in \mathbb{R}^d \backslash N$.
L'application

$$\phi_x : y \mapsto x - y$$

est un $\mathcal{C}^1-$difféomorphisme sur $\mathbb{R}^d$ de Jacobien $(-1)^d$, donc pour tout $x \in \mathbb{R}^d \backslash N$ on a :

$$\begin{aligned}(g \star f)(x) &= \int_{\mathbb{R}^d} g(y) f(x-y) dy \\ &= \int_{\mathbb{R}^d} g(\phi_x(y)) f(x - \phi_x(y) dy \\ &= \int_{\mathbb{R}^d} g(x-y) f(y) dy \\ &= (f \star g)(x)\end{aligned}$$

●● **<u>associativité</u>** pour presque tout $x \in \mathbb{R}^d$, on a :

$$\begin{aligned}(f \star g) \star h(x) &= \int_{\mathbb{R}^d} (f \star g)(y) h(x-y) dy \\ &= \int_{\mathbb{R}^d} \left( \int_{\mathbb{R}^d} f(z) g(y-z) dz \right) h(x-y) dy \qquad (1)\end{aligned}$$

*où* $\int_{\mathbb{R}^d} f(z) g(y-z) dz$ est définie pour presque tout $y \in \mathbb{R}^d$, de même :

$$\begin{aligned}f \star (g \star h)(x) &= \int_{\mathbb{R}^d} f(z) (g \star h(x-z)) dz \\ &= \int_{\mathbb{R}^d} f(z) \left( \int_{\mathbb{R}^d} g(y) h(x-z-y) dy \right) dz \qquad (2)\end{aligned}$$

on passe de (1) à (2) par changement de variable $y \mapsto y - z$ et théorème de Fubini.
Par le théorème de Tonelli :

$$(x, y, z) \mapsto (f \otimes g \otimes h)(x, y, z) = f(x) g(y) h(z)$$

est intégrable sur $\mathbb{R}^{3d}$.
En considérant le difféomorphisme :

$$\phi(x, y, z) = (x - y - z, y, z)$$

qui vérifie $|det(D\phi)| = 1$, $(f \otimes g \otimes h \circ \phi)$ est également intégrable. Par le théorème de Fubini cette fonction est intégrable par rapport à $(y, z)$ et pour presque tout $x \in \mathbb{R}^d$

$$f \star (g \star h)(x) = \int_{\mathbb{R}^d \times \mathbb{R}^d} (h \otimes g \otimes f) \circ \phi(x, y, z) dy dz$$

par composition avec le difféomorphisme $\Theta : (y, z) \mapsto (y - z, z)$, de Jacobien 1, on a pour presque tout $x$ :

$$\begin{aligned}f \star (g \star h)(x) &= \int_{\mathbb{R}^d \times \mathbb{R}^d} (h \otimes g \otimes f) \circ \phi \circ \Theta(x, y, z) dy dz \\ &= \int_{\mathbb{R}^d} \left( \int_{\mathbb{R}^d} (h \otimes g \otimes f) \circ \phi \circ \Theta(x, y, z) dz \right) dy \qquad (Fubini) \\ &= (f \star g) \star h(x)\end{aligned}$$

□

**Définition 1.5.2: Cas discret[64]**

Soit $(f(n))_{n\in\mathbb{Z}}$ et $(g(n))_{n\in\mathbb{Z}}$ deux suites de nombres réels. Le produit de convolution est la suite $(h(n))_{n\in\mathbb{Z}}$ dont le terme général est défini par :

$$h(n) = (f \star g)(n) = \sum_{k=-\infty}^{+\infty} f(n-k)g(k) \tag{1.12}$$

**Remarque 1.5.1**

1. Lorsque l'on suppose que les termes de $g$ sont nuls en dehors des indices appartenant à $[-K, +K]$ :
$$h(n) = (f \star g)(n) = \sum_{k=-K}^{K} f(n-k)g(k) \tag{1.13}$$
c'est le cas le plus utilisé, en effet la suite $(g(n))_{n\in\mathbb{Z}}$ représente le motif( Kernel)
2. Le produit de convolution dans le cas discret est commutatif et associatif.

### 1.5.2 Convolution pour l'imagerie

Une image est un tableau de nombre, une image gris est représentée par un tableau à deux dimension et une image couleur(RGB) est représentée par un tableau à trois dimensions

**Notations 1.5.1:**

on désigne par :
$I$ : une image.
$A$ : la matrice associé a l'image $I$.
$M$ : le motifs(kernel, filtre, masque).

**Définition 1.5.3: Convolution(deux dimensions)[29]**

La convolution en deux dimensions est une opération qui :

1. à partir d'une matrice d'entrée $A$
2. et d'une matrice d'un motif $M$

associe une matrice de sortie $A \star M$ donnée par :

$$(A \star M)(i,j) = \sum_n \sum_m A(i-n, j-m)M(n,m) \tag{1.14}$$

Tout d'abord, il faut retourner la matrice $M$ : Le calcul de $A \star M$ s'effectue coefficient par


FIGURE 1.3 – Motif retourner

coefficient :

1. On centre le motif retourné sur la position du coefficient à calculer
2. On multiplie chaque coefficient de A par le coefficient du motif retourné en face
3. La somme de ces produits donne un coefficient de $A \star M$.

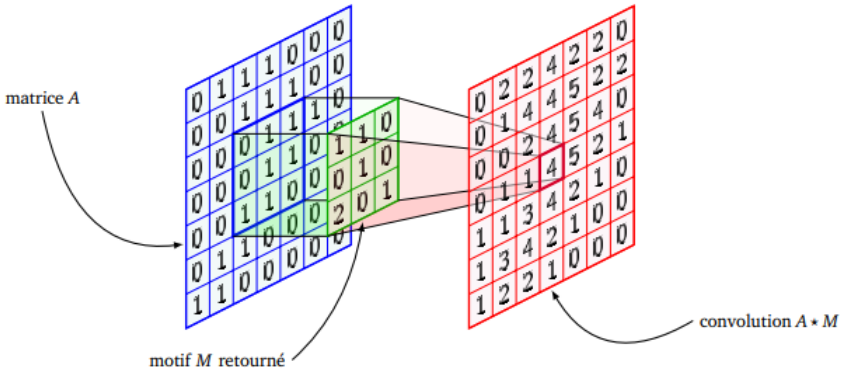



FIGURE 1.4 – Calcul d'un coefficient de $A \star M$

**Exemple 1.5.1:**

*** Translation.** Une convolution par la matrice

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

correspond à une translation des coefficients vers le bas. Par exemple :

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix} \star \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

De même pour une translation des coefficients vers la droite.
*** Moyenne.** On effectue une moyenne locale des coefficients à l'aide du motif :

$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

On obtient une image floue par application du motif moyenne :
• **Convolution de même taille.** La matrice $B = A \star M$ est de même taille que la matrice A. Pour les calculs, on peut être amené à rajouter des zéros virtuels sur les bords de la matrice A.

• **Convolution étendue**. On rajoute deux rangées de zéros virtuels autour de la matrice A. Si A est de taille $n \times p$ et M est de taille $3 \times 3$ alors pour cette convolution la matrice B est taille $(n+2) \times (p+2)$

• **Convolution restreinte**. On ne s'autorise pas à rajouter des zéros virtuels. Pour cette convolution la matrice B est donc de taille $(n-2) \times (p-2)$


FIGURE 1.5 – Application du filtre moyenne

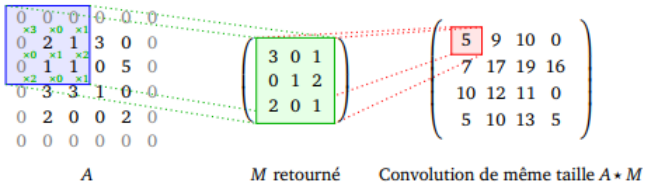



FIGURE 1.6 – Convolution de même taille

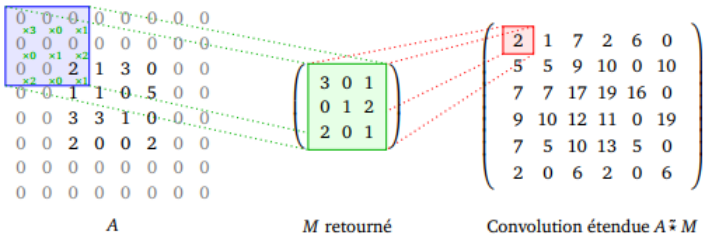



FIGURE 1.7 – Convolution étendue

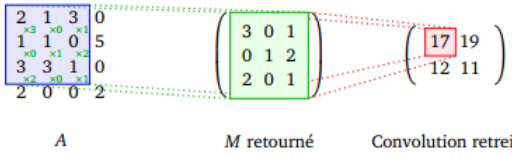



FIGURE 1.8 – Convolution restreinte

### 1.5.3 Pooling

Le pooling (regroupement de termes) consiste à transformer une matrice en une matrice plus petite tout en essayant d'en garder les caractéristiques principales.
Un pooling de taille $k$ transforme une matrice de taille $n \times p$ en une matrice de taille $k$ fois plus petite, c'est-à-dire de taille $n/k \times p/k$. Une sous-matrice de taille $k \times k$ de la matrice de départ produit un seul coefficient de la matrice d'arrivée.

On distingue deux types de pooling :

1. Le max-pooling de taille k consiste à retenir le maximum de chaque sous-matrice de taille $k \times k$
2. Le pooling en moyenne de taille k (average pooling) consiste à retenir la moyenne des

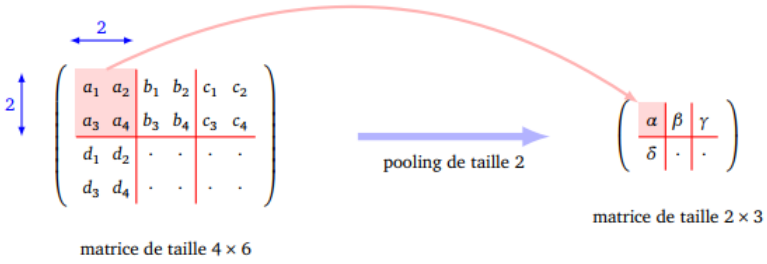



FIGURE 1.9 – Pooling de taille 2×2

termes de chaque sous-matrice de taille $k \times k$

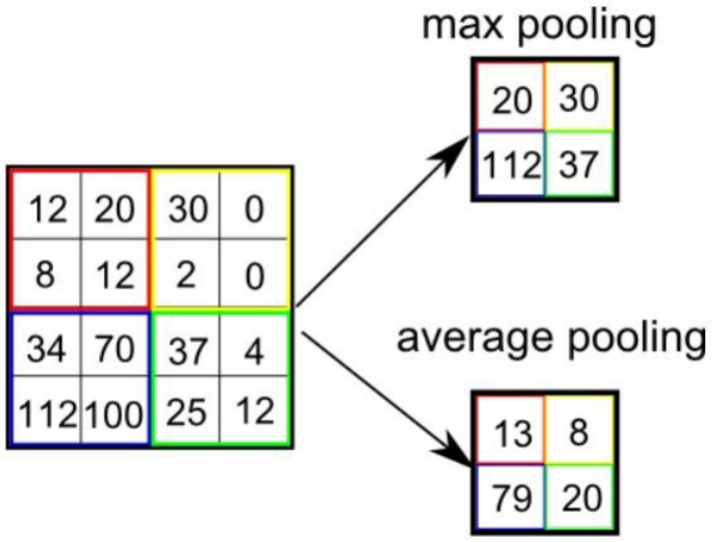



FIGURE 1.10 – Max-pooling et average-pooling de taille 2×2

Le max-pooling, qui ne retient que la valeur la plus élevée par sous-matrice, permet de détecter la présence d'une caractéristique (par exemple un pixel blanc dans une image noire). Tandis que le pooling en moyenne prend en compte tous les termes de chaque sous-matrice (par exemple avec 4 pixels d'une image de ciel, on retient la couleur moyenne).

### 1.5.4 Neurone et couche de convolution

À l'aide de la convolution, nous allons définir un nouveau type de couche de neurones : une couche de convolution. Tout d'abord un neurone de convolution de taille $3 \times 3$ est un neurone classique ayant 9 entrées.

Les poids du neurone correspondent aux coefficients d'une matrice de convolution(le motif) :

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{23} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \tag{1.15}$$

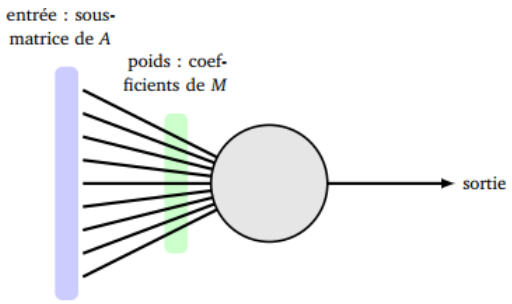



FIGURE 1.11 – Neurone de convolution

alors le neurone produit la sortie :

$$s = g(a_{11}m_{33}+a_{12}m_{32}+a_{13}m_{31}+a_{21}m_{23}+a_{22}m_{22}+a_{23}m_{21}+a_{31}m_{13}+a_{32}m_{12}+a_{33}m_{11}) \quad (1.16)$$

avec $g$ est la fonction d'activation.
Imaginons que l'entrée soit une image ou un tableau de dimension 2, pour nous ce sera une matrice A. Alors un neurone de convolution est relié à une sous-matrice $3 \times 3$ de A.

### 1.5.4.1 Couche de convolution

Considérons une entrée A représentée par une matrice de taille $n \times p$. Une couche de convolution (pour un seul motif) est la donnée d'une matrice M appelée motif (par exemple de taille $3 \times 3$) et qui renvoie en sortie les coefficients de la matrice $A \star M$ Pour une entrée de taille $n \times p$,

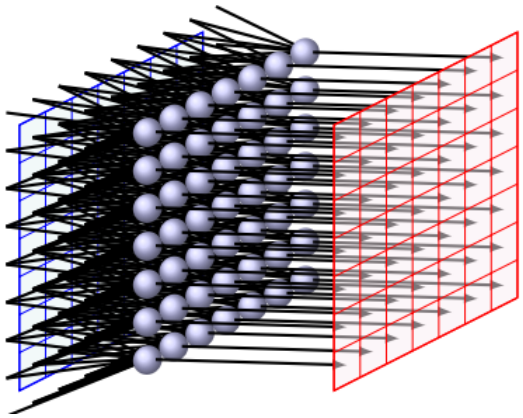


FIGURE 1.12 – Couche de convolution

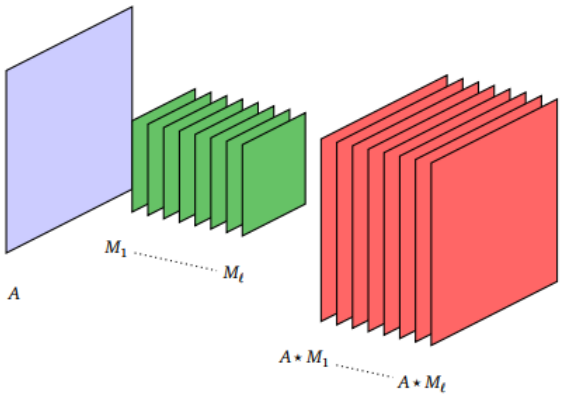
il y a donc $np$ neurones de convolutions, chacun ayant 9 arêtes (car le motif est de taille $3 \times 3$). Les poids sont communs à tous les neurones (ce sont les coefficients de M). Ainsi, pour une couche de convolution, il y a seulement 9 poids à déterminer pour définir la couche de convolution (bien que le nombre de neurones puisse être très grand).

**Remarque 1.5.2:**

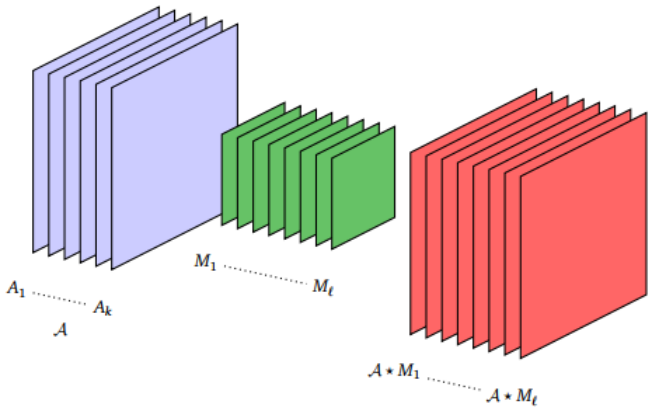
1. Combinatoire : dans le cas d'une couche complètement connectée, le nombre de poids à calculer serait énorme. En effet, une couche de $np$ neurones complètement connectée à une entrée de taille $n \times p$ amènerait à calculer $(np)^2$ poids. Pour$n = p =$ 100, cela fait 10000 neurones et 100000000 poids. Pour se rappeler, notre couche de convolution est définie par 9 poids (quels que soient n et p).
2. D'un point de vue mathématique, une couche de convolution associée au motif M est l'application :
$$\begin{array}{rccc} F & : & \mathcal{M}_{n,p}(\mathbb{R}) & \to & \mathcal{M}_{n,p}(\mathbb{R}) \\ & & A & \mapsto & A \star M \end{array}$$
3. Le motif peut être d'une taille différente de la taille $3 \times 3$ considérée dans les exemples qui est cependant la plus courante(à titre d'exemple VGG19)

### 1.5.4.2 Différentes couches d'un réseau de neurones convolutif

**Plusieurs filtres.** Dans la pratique une couche de convolution est définie avec plusieurs motifs, c'est-à-dire un nombre $l$ de matrices $M_1, M_2, ..., M_l$. Ainsi pour une entrée $A$ de taille $n \times p$, une couche de convolution à $l$ motifs renvoie une sortie de taille $n \times p \times l$, correspondant à $(A \star M_1, A \star M_2, ..., A \star M_l)$.




**Convolution à plusieurs filtres à partir de plusieurs canaux.** C'est le cas général dans la pratique. Une entrée donnée par plusieurs canaux $\mathcal{A} = (A_1, A_2, ..., A_k)$, associée à des motifs $M_1, M_2, ..., M_l$ (qui sont donc chacun des $3-$tenseurs de taille $(3, 3, k)$) produit une sortie de taille $n \times p \times l$, correspondant à $(A \star M_1, A \star M_2, ..., A \star M_l)$. Si l'entrée $\mathcal{A}$ est de taille $(n, p, k)$ alors la sortie est de taille $(n, p, l)$.



sur cette figure chaque motif $M_i$ est représenté par carré $3 \times 3$, alors qu'en fait chacun devrait être une boîte en 3 dimensions de taille $3 \times 3 \times k$.

### 1.5.5 Structure des réseaux de neurones convolutifs

Un réseau neuronal convolutif CNN est un algorithme d'apprentissage en profondeur qui peut prendre en compte une image d'entrée, attribuer une importance (poids et biais apprenables) à divers aspects/objets de l'image et être capable de les différencier les uns des autres. Le prétraitement requis dans un CNN est beaucoup plus faible par rapport aux autres algorithmes de classification. Alors que dans les méthodes primitives, les filtres sont conçus à la main, avec une formation suffisante, les CNNs ont la capacité d'apprendre ces filtres/caractéristiques.
L'architecture d'un CNN est analogue à celle du modèle de connectivité des neurones dans le cerveau humain et s'inspire de l'organisation du cortex visuel. Les neurones individuels ne répondent aux stimuli que dans une région restreinte du champ visuel appelée champ récepteur. Une collection de ces champs se chevauchent pour couvrir toute la zone visuelle.
Nous utilisons trois principaux types de couches pour créer des architectures CNNs : la couche de convolution , la couche de mise en commun(pooling) et la couche entièrement connectée.

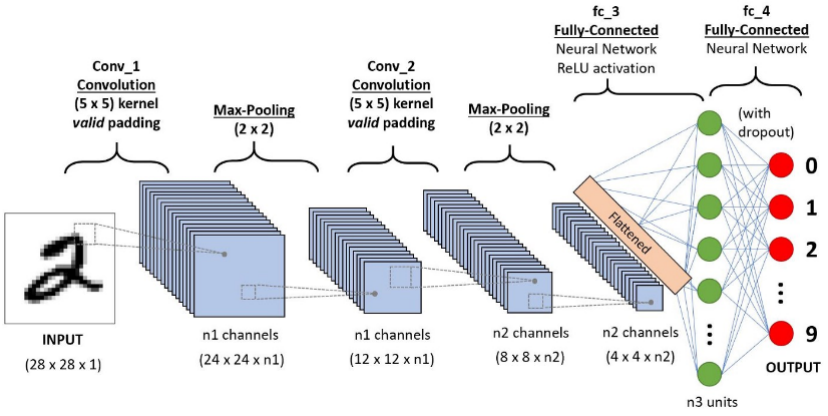



FIGURE 1.13 – Architecture d'un réseau de neurones convolutif

## 1.6 Comparaison entre les CNNs et les FCNN en traitement d'images

### Base MNIST

Un élément essentiel de l'apprentissage automatique est de disposer de données d'apprentissage nombreuses et de qualité. Une telle base est la base MNIST. chaque donnée est constituée d'une image et du chiffre attendu.

- La base est formée de 60000 données d'apprentissage et de 10000 données de test.
- Chaque donnée est de la forme : [une image, le chiffre attendu].
- Chaque image est de taille $28 \times 28$ pixels, chaque pixel contenant un des 256 niveaux de gris (numérotés de 0 à 255).

### Traitement de données

**Donnée d'entrée.** En entrée du réseau de neurones, nous devons avoir un vecteur. Au départ chaque image est un tableau de taille $28 \times 28$ ayant des entrées entre 0 et 255. nous normalisons les données(voir 2.2.6) dans l'intervalle $[0, 1]$ (en divisant par 255).
**Donnée de sortie.** Notre réseau de neurones ne va pas renvoyer le chiffre attendu, mais une liste de probabilités. Ainsi chaque chiffre doit être codé par une liste de 0 et de 1. 0 est codé par $(1, 0, 0, 0, 0, 0, 0, 0, 0, 0)$ . . . 9 est codé par $(0, 0, 0, 0, 0, 0, 0, 0, 0, 1)$.

Nous résumons les résultats obtenu par les deux réseau(CNN et FCNN) dans le tableau suivant :

| Résultat | Exactitude du test | Erreur du test | paramètres entrainables |
|---|---|---|---|
| Réseau FCNN | 0.9486 | 0.2930 | 16 750 |
| Réseau CNN | 0.9907 | 0.0561 | 13 714 |

TABLE 1.1 – Résultats obtenus par les réseaux CNN et FCNN

d'après le tableau ci-dessus, l'exactitude obtenue par le réseau CNN est 99.07% ce qui très bon, Cependant le résultat obtenue par le réseau FCNN est 94.86%. Le nombre de paramètre du réseau CNN est plus petit que ceux du réseau FCNNN. On conclut que en traitement d'images, les réseaux CNNs sont beaucoup mieux que les FCNN, car les CNNs permettent d'améliorer les performances et de réduire la complexité spatiale et temporelle.

## 1.7 Conclusion

Dans ce chapitre, on a étudié quelques algorithme d'apprentissage profond ainsi que leurs avantages, puis on a abordé les réseaux de neurones convolutifs et leurs avantages par rapport aux réseaux entièrement connectés en traitement d'images. Les réseaux profonds peuvent contenir des millions et des milliards de paramètres, ce qui peut engendrer le problème de sur-apprentissage(Overfitting). Dans le chapitre suivant, on va étudier quelques méthodes de régularisation qui peuvent limiter le problème de sur-apprentissage.

Chapitre 2

# Méthodes de régularisation

## 2.1 Introduction

L'un des problèmes les plus courants auxquels sont confrontés les professionnels de la science des données est d'éviter le sur-apprentissage, c'est la situation où notre modèle fonctionnait exceptionnellement bien sur les données d'apprentissage mais n'était pas en mesure de prédire les données de test. Durant ce chapitre, nous comprendrons le concept de sur-apprentissage et comment la régularisation aide à surmonter le même problème. Nous examinerons ensuite quelques techniques de régularisation différentes incluant des méthodes stochastiques et des méthodes déterministes.

**Problèmes :**

1. **Le sous-apprentissage(Underfitting).** Révèle une conception correcte de l'architecture du réseau mais une mauvaise mise en œuvre. On obtient alors des poids qui ne répondent pas correctement au problème. Cela peut être dû aux raisons suivantes :

   (a) Les données ne sont pas en nombre suffisant.

   (b) Le nombre d'itérations est insuffisant.

   (c) Le pas $\delta$ est trop grand.

   La solution : ajouter des données, augmenter le nombre d'itérations ou diminuer le pas.

2. **Le sur-apprentissage(Overfitting).** Le modèle obtenu colle parfaitement aux données d'apprentissage, mais cependant les prédictions pour de nouvelles valeurs sont mauvaises. Il s'agit donc d'un problème délicat : la fonction $F$ obtenue vérifie bien $F(X_i) \approx y_i$ pour toutes les données, mais pour une nouvelle entrée $X$, la sortie $F(X)$ n'est pas une bonne prédiction. Cela se produit lorsque l'on se concentre uniquement sur l'apprentissage à partir des données, mais que l'on a oublié que le but principal est la prédiction.

**La Régularisation** est un ensemble de méthode qui permet à la fois d'optimiser l'apprentissage d'un modèle de l'apprentissage profond et d'éviter le sur-apprentissage(manque de généralisation). Il y a plusieurs type de régularisation, dans ce rapport nous concentrons sur cinq type : Dropout, Dropconnect,Normalisation par lot(batch normalisation) qui sont des méthodes stochastiques, puis la régularisation $\ell_1$(LASSO) , régularisation $\ell_2$(Ridge) qui sont des méthodes déterministes.

## 2.2 Méthodes Stochastiques

### 2.2.1 Description du modèle

On considère une couche complètement connecter, avec une entrée $x = (x_1, x_2, ..., x_n)^t$ et la matrice des poids $W$ de taille $d \times n$. La sortie de cette couche est $r = (r_1, r_2, ..., r_d)^t$, avec :

$$r = g(u) = g(Wx) \tag{2.1}$$

*où* $g$ est une fonction d'activation non linéaire.
Nous considérons une architecture de modèle standard composée de quatre composants de base :

1. Extracteur de caractéristiques : $v = f(x, W_f)$ où $v$ sont les caractéristiques de sortie, $x$ sont les données d'entrée du modèle global et $W_f$ sont les paramètres de l'extracteur de caractéristiques. Nous choisissons $f(\cdot)$ comme un réseau de neurones convolutifs multi-couches (CNN), $W_f$ étant les filtres convolutionnels (et les biais) du CNN.

2. Couche DropConnect (respectivement de Dropout) : $r = g(u) = g((M \odot W)v)$ $(r = g(u) = m \odot g(Wv))$ où $v$ est la sortie de l'extracteur de caractéristiques, $W$ est la matrice des poids d'une couche complètement connecter(FC), $g$ est une fonction d'activation non linéaire et $M$(resp. $m$) est la matrice(resp. vecteur) de masque binaire.

3. Couche de classification Softmax : $o = s(g; W_s)$ prend comme entrée $g$ et utilise les paramètres $W_s$ pour produire une sortie $k-$dimensionnelle ($k$ étant le nombre de classes).

4. la fonction erreur entropie croisée : $E(y, o) = -\sum_{i=1}^{k} y_i log(o_i)$ prend une probabilités $o$ et les étiquettes $y$ comme entrée.
   on pose
   $$\theta = \{W_f, W, W_s\}$$

### 2.2.2 Dropout

Le Dropout [23] est une technique pour améliorer l'apprentissage d'un réseau complètement connecter(FCANN) et en particulier pour prévenir le sur-apprentissage. L'idée est de désactiver certains neurones d'une couche lors des étapes de l'apprentissage. Ces neurones sont choisis au hasard et sont désactivés temporairement pour une itération (par exemple on peut choisir à chaque itération de désactiver un neurone avec une probabilité $\frac{1}{2}$ ). Cela signifie que l'on retire toute arête entrante et toute arête sortante de ces neurones, ce qui revient à mettre les poids à zéro tant pour l'évaluation que pour la rétropropagation. Lors de l'itération suivante on choisit de nouveau au hasard les neurones à désactiver.
Appliquer un Dropout à une couche de neurones revient à désactiver chaque neurone suivant une loi de Bernoulli de paramètre p($\mathcal{B}(p)$) où $0 \leq p \leq 1$ est un réel fixé. C'est à dire que chaque élément de la sortie d'une couche est conservé avec une probabilité $p$, sinon mis à 0 avec une probabilité $q = (1 - p)$.

Lorsque Dropout est appliqué aux sorties d'une couche entièrement connectée, nous pouvons écrire :

$$r = m \odot g(Wx) \tag{2.2}$$

où $\odot$ désigne le produit élément par élément et $m$ est un vecteur de masque binaire de taille $d$ avec chaque élément $j$ tiré indépendamment de $m_j \rightsquigarrow \mathcal{B}(p)$.

**Remarque 2.2.1**

1. Lors de la phase de test, tous les neurones sont actifs.
2. De nombreuses fonctions d'activation couramment utilisées ont la propriété que $g(0) = 0$( par exemple tanh, ReLu ...) Ainsi, l'équation 2.2 précédente s'écrit :

$$r = g(m \odot Wx) \tag{2.3}$$

où Dropout est appliqué aux entrées de la fonction d'activation.

**Algorithm 2** Apprentissage SGD avec Dropout

**Entrées:** . un exemple $x$, paramètres $\theta_{k-1}$ de l'étape $k-1$,le taux d'apprentissage $\delta$.
**Sorties:** . paramètres $\theta_k$

**Propagation Avant :**
Extracteur de caractéristiques : $v \longleftarrow f(x, W_f)$
Masque $m$ échantillon aléatoire : $m_i \rightsquigarrow \mathcal{B}(p)$
calculer l'activation : $r = m \odot g(Wv)$
calculer la sortie : $o = s(r, W_s)$
**Rétro-propagation du gradient :**
calculer la différentielle de la fonction erreur $E'_\theta$ par rapport à $\theta$.
Mettre à jour la couche softmax :$W_s = W_s - \delta E'_{W_s}$
Mettre à jour la couche Dropout : $W = W - \delta(m \odot E'_{W_s})$
Mettre à jour l'extracteur de caractéristique : $W_f = W_f - \delta E'_{W_f}$

### 2.2.3 DropConnect

On va reprendre le même principe que précédemment. Mais au lieu de désactiver des neurones, on va simplement désactiver les connexions entrantes (toujours de façon aléatoire) sur une couche depuis la précédente. D'un point de vue du réseau , cela revient à instancier les valeurs des poids des connexions à 0.
DropConnect [61] est la généralisation de Dropout dans laquelle chaque connexion, plutôt que chaque unité de sortie, peut être désactivé avec une probabilité de $p$. DropConnect est similaire à Dropout car il introduit une parcimonie dynamique dans le modèle, mais diffère en ce que la parcimonie est sur les poids $W$ , plutôt que sur les vecteurs de sortie d'une couche. En d'autres termes, la couche entièrement connectée avec DropConnect devient une couche peu connectée dans laquelle les connexions sont choisies au hasard lors de la phase d'apprentissage.
Pour une couche DropConnect, la sortie est donnée par :

$$r = g((M \odot W)x) \tag{2.4}$$

où $M$ est une matrice binaire codant les informations de connexion et $M_{ij} \rightsquigarrow \mathcal{B}(p)$. Chaque élément du masque $M$ est dessiné indépendamment pour chaque exemple lors de l'apprentissage, instanciant essentiellement une connectivité différente pour chaque

exemple vu. De plus, les biais sont également masqués pendant l'apprentissage.
étant donné les paramètres $\theta = \{W_f; W; W_s\}$ et un masque tiré au hasard $M$. Le modèle global $h(x, \theta, M)$ fait correspondre les données d'entrée $x$ à une sortie $o$ à travers une séquence d'opérations. La valeur correcte de $o$ est obtenue en additionnant sur tous les masques possibles $M$ :

$$o = \mathbf{E}_M \left[ h(x, \theta, M) \right] \tag{2.5}$$

$$= \sum_M p(M) h(x, \theta, M) \tag{2.6}$$

la sortie est un mélange de $2^{|M|}$ réseau différent, chaque sortie avec une probabilité $p(M)$ si $p = \frac{1}{2}$,alors les probabilités sont égaux pour tout les $M$, et :

$$o = \frac{1}{|M|} \sum_M h(x, \theta, M) \tag{2.7}$$

$$= \frac{1}{|M|} \sum_M s(g((M \odot W)v), W_s) \tag{2.8}$$

### 2.2.4 Apprentissage

L'apprentissage du modèle décrit ci-dessus commence par la sélection d'un exemple $x$ dans l'ensemble d'apprentissage $X$ et l'extraction des caractéristiques pour cet exemple, $v$. Ces caractéristiques sont les entrées de la couche de DropConnect où une matrice de masque $M$ est d'abord tirée d'une distribution de Bernoulli de paramètre $p$ pour masquer les éléments de la matrice de pondération et les biais dans la couche DropConnect. Un élément clé pour réussie l'apprentissage avec DropConnect est la sélection d'un masque différent pour chaque exemple d'apprentissage.

**Algorithm 3** Apprentissage SGD avec DropConnect

**Entrées:** Un exemple $x$, paramètres $\theta_{k-1}$ de l'étape $k-1$,le taux d'apprentissage $\delta$.
**Sorties:** Paramètres $\theta_k$

**Propagation Avant :**
Extracteur de caractéristiques : $v \longleftarrow f(x, W_f)$
Masque $M$ échantillon aléatoire : $M_{ij} \rightsquigarrow \mathcal{B}(p)$
Calculer l'activation : $r = g((M \odot W)v)$
Calculer la sortie : $o = s(r, W_s)$
**Rétro-propagation du gradient :**
Calculer la différentielle de la fonction erreur $E'_\theta$ par rapport à $\theta$.
Mettre à jour la couche softmax :$W_s = W_s - \delta E'_{W_s}$
Mettre à jour la couche DropConnect : $W = W - \delta(M \odot E'_{W_s})$
Mettre à jour l'extracteur de caractéristique : $W_f = W_f - \delta E'_{W_f}$

### 2.2.5 Complexité du réseau DropConnect

**Définition 2.2.1: Réseau DropConnect[61]**

Étant donné une base d'apprentissage $S = \{x_1, ..., x_l\}$ avec les étiquette $\{y_1, ..., y_l\}$, nous définissons le réseau DropConnect comme un modèle mixte :

$$o = \mathbf{E}_M \left[ h(x, \theta, M) \right] = \sum_M p(M) h(x, \theta, M) \tag{2.9}$$

**Définition 2.2.2: Erreur logistique[61]**

La fonction d'erreur suivante définie sur la classification de classe $k$ est appelée la fonction d'erreur logistique :

$$E_y(o) = -\sum_i y_i ln \left( \frac{exp(o_i)}{\sum_j exp(o_j)} \right) \tag{2.10}$$

**Définition 2.2.3: Complexité de Rademacher empirique[61]**

Pour un échantillon $S = \{x_1, ..., x_l\}$ engendré par une distribution $D$ sur un ensemble $X$ et une classe de fonctions à valeurs réelles $\mathcal{F}$ dans le domaine $X$, la complexité empirique de Rademacher de $\mathcal{F}$ est la variable aléatoire :

$$hat R_\ell(\mathcal{F}) = \mathbf{E}_\sigma \left[ sup_{f \in \mathcal{F}} | \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i f(x_i) || x_1, ..., x_l \right] \tag{2.11}$$

où $\sigma_1, ..., \sigma_\ell$ sont des variables aléatoires de Rademacher uniformes indépendantes prenant les valeurs $\{+1, -1\}$.

**Définition 2.2.4: complexité de Rademacher[61]**

La complexité de Rademacher de $\mathcal{F}$ est :

$$R_\ell(\mathcal{F}) = \mathbf{E}_S \left[ \hat{R}_\ell(\mathcal{F}) \right] \tag{2.12}$$

**Théorème 2.2.1: [61]**

Considérons le réseau de neurones DropConnect défini dans la définition 2.2.5. Soit $\hat{R}_\ell(\mathcal{G})$ la complexité de Rademacher empirique de l'extracteur de caractéristiques et $\hat{R}_\ell(\mathcal{F})$ la complexité de Rademacher empirique du réseau entière. De plus, nous supposons :

1. les paramètres de la couche de Dropconnect $|W| \leq B_h$
2. les paramètres de $s$ $|W_s| < B_s$

Alors, on a :

$$\hat{R}_\ell(\mathcal{F}) \leq p(2\sqrt{k} B_s n \sqrt{d} B_h) \hat{R}_\ell(\mathcal{G}) \tag{2.13}$$

*Démonstration.* voir Annexe .1 □

**Remarque 2.2.2**

Le résultat important de ce théorème est que la complexité est une fonction linéaire.

### 2.2.6 Normalisation par lots(batch normalization)

La normalisation par lots [38] est une méthode utilisée pour rendre les réseaux de neurones plus rapides et plus stables grâce à la normalisation des entrées des couches par centrage et remise à l'échelle.
Dans un réseau de neurones, la normalisation par lots est réalisée par une étape de normalisation qui fixe les moyennes et les variances des entrées de chaque couche. La normalisation serait effectuée sur l'ensemble de l'apprentissage, est restreinte à chaque mini-lot dans le processus de l'apprentissage.
On considère un mini-lot $\mathfrak{B} = \{(x_i)/i = 1...m\}$ de taille $m$ de l'ensemble d'apprentissage complet. Pour une couche avec une entrée $d-$dimensionnelle $x = (x^{(1)}, x^{(2)}..., x^{(d)})$, La moyenne et la variance empiriques de $\mathfrak{B}$ sont données par :

$$\mu_{\mathfrak{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \sigma^2_{\mathfrak{B}} = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathfrak{B}})^2 \tag{2.14}$$

nous normaliserons chaque dimension

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_{\mathfrak{B}}}{\sqrt{\sigma^2_{\mathfrak{B}} + \epsilon}} \tag{2.15}$$

avec $k \in [1, d]$ , $i \in [1, m]$
une telle normalisation accélère la convergence, même lorsque les caractéristiques ne sont pas décorrélées.
Comme on ne peut pas être sûr que cette normalisation soit bénéfique, on introduit des nouveaux paramètres $(\gamma(k), \beta(k))$ qui permettent de rectifier la transformation :

$$y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)} \tag{2.16}$$

Formellement, l'opération qui implémente la normalisation par lots est une transformation

$$BN_{\gamma,\beta} : x^{(k)} \mapsto y^{(k)}$$

avec $i, k \in \{1, 2, ..., m\} \times \{1, 2..., d\}$ appelée transformation de normalisation par lots. La sortie $y^k = BN_{\gamma,\beta}(x^k)$ de la transformée est ensuite transmis à d'autres couches de réseau, tandis que la sortie normalisée $\hat{x}_i^{(k)}$ reste interne au calque courant.

**Remarque 2.2.3**

1. Les paramètres $(\gamma^{(k)}, \beta^{(k)})$ sont optimisés comme tous les autres paramètres du réseaux par descente de gradient.
2. $BN_{\gamma,\beta}(x)$ dépend à la fois de l'exemple d'apprentissage et des autres exemples du mini-lot.

**Algorithm 4** Transformation de normalisation par lots, appliquée à l'activation $x$ sur un mini-lot

**Entrées:** . Valeurs de $x$ sur un mini-lot : $\mathfrak{B} = \{(x_i)/i = 1...m\}$,
Paramètres à apprendre : $\gamma, \beta$.
**Sorties:** . $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$\mu_{\mathfrak{B}} \longleftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$ // moyenne du mini-lot
$\sigma^2_{\mathfrak{B}} \longleftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathfrak{B}})^2$ // variance du mini-lot
$\hat{x}_i \longleftarrow \frac{x_i - \mu_{\mathfrak{B}}}{\sqrt{\sigma^2_{\mathfrak{B}} + \epsilon}}$ // Normalisation
$y_i \longleftarrow \gamma \hat{x}^i + \beta$

Pendant l'apprentissage , nous devons rétro-propager le gradient de l'erreur par cette transformation, ainsi que calculer les gradients par rapport aux paramètres de la transformée $BN$. Nous utilisons la règle de la chaîne, comme suit :

$$\frac{\partial E}{\partial \hat{x}^i} = \frac{\partial E}{\partial y^i} \cdot \frac{\partial y^i}{\partial \hat{x}^i} = \frac{\partial E}{\partial y^i} \cdot \gamma$$
$$\frac{\partial E}{\partial \sigma^2_{\mathfrak{B}}} = \sum_{i=1}^{m} \frac{\partial E}{\partial \hat{x}^i} \cdot (x^i - \mu_{\mathfrak{B}}) \cdot (-\frac{1}{2}(\sigma^2_{\mathfrak{B}} + \epsilon)^{-\frac{3}{2}})$$

$$\frac{\partial E}{\partial \mu_{\mathfrak{B}}} = \sum_{i=1}^{m} \frac{\partial E}{\partial \hat{x}^i} \cdot \frac{-1}{\sqrt{\sigma^2_{\mathfrak{B}} + \epsilon}}$$
$$\frac{\partial E}{\partial x^i} = \frac{\partial E}{\partial \hat{x}^i} \cdot \frac{1}{\sqrt{\sigma^2_{\mathfrak{B}} + \epsilon}} + \frac{\partial E}{\partial \sigma^2_{\mathfrak{B}}} \cdot \frac{2(x^i - \mu_{\mathfrak{B}})}{m} + \frac{\partial E}{\partial \mu_{\mathfrak{B}}} \cdot \frac{1}{m}$$
$$\frac{\partial E}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial E}{\partial y^i} \cdot \hat{x}^i$$
$$\frac{\partial E}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial E}{\partial y^i}$$

Ainsi, la transformée $BN$ est une transformation différentiable qui introduit des activations normalisées dans le réseau. Cela garantit que pendant l'apprentissage du modèle, les couches peuvent continuer à apprendre sur les distributions d'entrée qui présentent moins de décalage de covariable interne, accélérant ainsi l'apprentissage. De plus, la transformée affine apprise appliquée à ces activations normalisées permet à la transformée BN de représenter la transformation d'identité et préserve la capacité du réseau.

### 2.2.7 Réseaux convolutifs normalisés par lots(batch normalized CNN)

La normalisation par lots peut être appliquée à n'importe quel ensemble d'activations dans le réseau. Ici, nous nous concentrons sur les transformées qui consistent en une transformation affine suivie d'une fonction d'activation non linéaire élément par élément :

$$z = g(Wu + b) \tag{2.17}$$

où W et b sont des paramètres appris du modèle, et $g(\cdot)$ est la fonction d'activation non linéaire(sigmoïde ou ReLU). Cette formulation couvre à la fois les couches entièrement

connectées(FC) et convolutionnelles . Nous ajoutons la transformée $BN$ immédiatement avant la non-linéarité, en normalisant $x = Wu + b$.
Pour les couches convolutives, nous souhaitons en outre que la normalisation obéisse à la propriété convolutive, de sorte que différents éléments de la même carte de caractéristique(feature map), à différents emplacements, soient normalisés de la même manière. Pour y parvenir, nous normalisons conjointement toutes les activations dans un mini-lot, sur tous les emplacements. En Alg.4, nous laissons $\mathfrak{B}$ être l'ensemble de toutes les valeurs d'une carte de caractéristique à la fois pour les éléments d'un mini-lot et les emplacements spatiaux, donc pour un mini-lot de taille $m$ et de carte de caractéristique de taille $p \times q$, on utilise le mini-lot effectif de taille $m' = |\mathfrak{B}'| = m \cdot p \cdot q$, on fait l'apprentissage d'une paire de paramètres $\gamma_k$ et $\beta_k$ par un carte de caractéristique, plutôt que par activation. Alg.5 est modifié de manière similaire, de sorte que pendant l'inférence, la transformée $BN$ applique la même transformation linéaire à chaque activation dans une carte de caractéristiques donnée.

## 2.3 Méthodes déterministes

### 2.3.1 Régularisation $\ell_2$

La régularisation $\ell_2$ [56] est le type le plus courant de toutes les techniques de régularisation et est également connue sous le nom de décroissance du poids ou Régression Ridge.
Lors de la régularisation $\ell_2$, la fonction erreur du réseau de neurones est prolongée par un terme dit de régularisation, que l'on appelle ici $\Omega$.

$$\Omega(w) = ||w||_2^2 = \sum_i \sum_j w_{ij}^2 \tag{2.18}$$

Le terme de régularisation $\Omega$ est défini comme la norme euclidienne (ou norme $\ell_2$) de la matrices de poids, qui est la somme de tous les poids au carré d'une matrice de poids. Le terme de régularisation est pondéré par le scalaire alpha divisé par deux et ajouté à la fonction erreur régulière choisie pour le modèle . Cela conduit à une nouvelle expression de la fonction erreur :

$$E^r = E(w) + \frac{\alpha}{2}\Omega(w) \tag{2.19}$$

$$= E(w) + \frac{\alpha}{2}\sum_i \sum_j w_{ij}^2 \tag{2.20}$$

$\alpha$ est appelé taux de régularisation et est un hyperparamètre supplémentaire que nous introduisons dans le réseau neuronal. En termes simples, l'alpha détermine à quel point nous régularisons notre modèle.
Dans l'étape suivante, nous pouvons calculer le gradient de la nouvelle fonction erreur et mettre le gradient dans la règle de mise à jour des poids :

$$\nabla E^r(w^k) = \nabla E(w^k) + \alpha w^k \tag{2.21}$$

on déduit la règle de mise à jour des poids :

$$w^{k+1} = (1 - \delta\alpha)w^k - \delta\nabla E(w^k) \tag{2.22}$$

La seule différence est qu'en ajoutant le terme de régularisation, nous introduisons une soustraction supplémentaire des poids actuels (premier terme de l'équation).
En d'autres termes, indépendamment du gradient de la fonction de perte, nous réduisons un peu les poids à chaque mise à jour.

### 2.3.2 Régularisation $\ell_1$

Dans le cas de la régularisation $\ell_1$ [56] également connue sous le nom de régression LASSO, nous utilisons simplement un autre terme de régularisation $\Omega$. Ce terme est la norme $\ell_1$ c'est à dire la somme des valeurs absolues des paramètres de poids dans une matrice de poids :

$$\Omega(w) = ||w||_1 = \sum_i \sum_j |w_{ij}| \tag{2.23}$$

la nouvelle fonction erreur :

$$E^r = E(w) + \frac{\alpha}{2}\Omega(w) \tag{2.24}$$

$$= E(w) + \frac{\alpha}{2} \sum_i \sum_j |w_{ij}| \tag{2.25}$$

le gradient de la nouvelle fonction erreur :

$$\nabla E^r(w^k) = \nabla E(w^k) + \alpha sgn(w^k) \tag{2.26}$$

les équations introduites pour les régularisations $\ell_1$ et $\ell_2$ sont des fonctions de contrainte, que nous pouvons visualiser : L'image de gauche montre la fonction de contrainte (zone

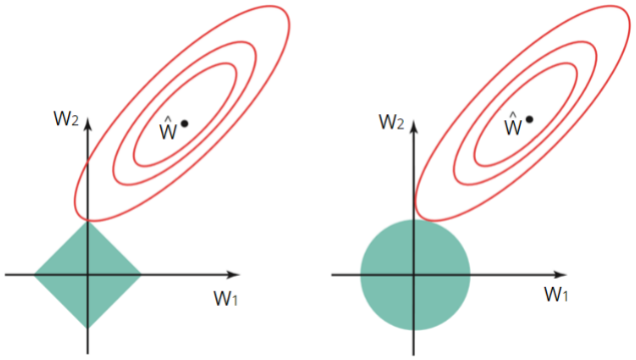



FIGURE 2.1 – Zone de contrainte de la régularisation $\ell_1$ et $\ell_2$

verte) pour la régularisation $\ell_1$ et l'image de droite montre la fonction de contrainte pour la régularisation $\ell_2$. Les ellipses rouges sont les contours de la fonction de perte utilisée lors de la descente de gradient. Au centre des contours se trouve un ensemble de poids optimaux pour lesquels la fonction de perte a un minimum global.
Dans le cas de la régularisation $\ell_1$ et $\ell_2$, les estimations de $w_1$ et $w_2$ sont données par le premier point d'intersection de l'ellipse avec la zone de contrainte verte.
la régularisation $\ell_2$ a une zone de contrainte circulaire, l'intersection ne se produira généralement pas sur un axe, et les estimations pour $w_1$ et $w_2$ seront exclusivement non nulles. Dans le cas de $\ell_1$, la zone de contraintes a une forme de losange avec des coins. Et ainsi, les contours de la fonction de perte couperont souvent la région de contrainte sur un axe. Ensuite, cela se produit, l'une des estimations ($w_1$ ou $w_2$) sera nulle, ainsi Dans un espace de grande dimension, de nombreux paramètres de poids seront égaux à zéro simultanément.

## 2.4 Conclusion

Dans ce chapitre on a abordé essentiellement le problème de sur-apprentissage, et on a décrit quelques méthodes qui permettent de limiter ce problème. On a vu que l'insuffisance de données aboutit au problème de sous-apprentissage. En général ? lorsque on a

un manque ou l'insuffisance de données, on ne peux pas construire un modèle de l'apprentissage profond performant. Dans le chapitre suivant, nous allons étudier l'apprentissage par transfert qui sert à résoudre le problème de l'insuffisance de données, et qui assouplie aussi l'hypothèse d'avoir des données indépendantes.

Chapitre 3

# Apprentissage par transfert

## 3.1 Introduction

L'apprentissage par transfert vise à améliorer les performances des apprenants cibles dans des domaines cibles en transférant les connaissances contenues dans des domaines sources différents mais liés. De cette manière, la dépendance vis-à-vis d'un grand nombre de données du domaine cible peut être réduite pour la construction des apprenants cibles. En raison des vastes perspectives d'application, l'apprentissage par transfert est devenu un domaine populaire et prometteur de l'apprentissage automatique. Durant ce chapitre nous allons étudier en détaille l'apprentissage par transfert, ainsi que les différents catégories de ce type d'apprentissage.

## 3.2 Motivation

L'étude de l'apprentissage par transfert est motivée par le fait que les gens peuvent appliquer intelligemment les connaissances acquises précédemment pour résoudre de nouveaux problèmes plus rapidement ou avec de meilleures solutions.
De nombreuses méthodes d'apprentissage automatique ne fonctionnent bien que sous une hypothèse commune : les données d'apprentissage et de test sont tirées de la même espace caractéristique $\mathcal{X}$ et la même distribution $P(X)$. Lorsque la distribution change, la plupart des modèles statistiques doivent être reconstruits à partir de zéro à l'aide de données d'apprentissage nouvellement collectées. Dans de nombreuses applications du monde réel, il est coûteux ou impossible de se souvenir des données d'apprentissage nécessaires et de reconstruire les modèles. Ce serait bien de réduire le besoin et l'effort de se souvenir des données d'apprentissage. Dans de tels cas, le transfert de connaissances ou l'apprentissage par transfert entre les domaines de tâches serait souhaitable.

## 3.3 Une brève histoire de l'apprentissage par transfert

La recherche sur l'apprentissage par transfert attire de plus en plus l'attention depuis 1995 sous différents noms : apprendre à apprendre, apprentissage tout au long de la vie, transfert de connaissances, apprentissage inductive,apprentissage multi-tâche, consolidation des connaissances, apprentissage sensible au contexte, biais inductif basé sur les connaissances, métaapprentissage...
Parmi celles-ci, une technique d'apprentissage étroitement liée à l'apprentissage par transfert est le cadre d'apprentissage multi-tâche [1], qui tente d'apprendre plusieurs tâches simultanément même lorsqu'elles sont différentes. Une approche typique de l'apprentissage

multi-tâche consiste à découvrir les caractéristiques communes qui peuvent bénéficier à chaque tâche individuelle.
l'apprentissage par transfert vise à extraire les connaissances d'un ou plusieurs tâches sources et applique les connaissances à une tâche cible. Contrairement à l'apprentissage multi-tâche, plutôt que d'apprendre toutes les tâches source et cible simultanément, l'apprentissage par transfert se soucie le plus de la tâche cible. Les rôles des tâches source et cible ne sont plus symétriques dans l'apprentissage par transfert.

## 3.4 Notations et définitions

**Définition 3.4.1: Domaine [70]**

Un domaine $D$ est la donnée d'un couple $\{\mathcal{X}, P(X)\}$ *où* $\mathcal{X}$ est un espace de caractéristique et $P(X)$ est une distribution de probabilité marginale avec $X = \{x_1, x_2, ..., x_n\} \in \mathcal{X}$ est un échantillon d'apprentissage.

**Définition 3.4.2: Tâche [70]**

Étant donné un domaine $D = \{\mathcal{X}, P(X)\}$, une tâche $\mathcal{T}$ est la donnée d'un couple $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ *où* $\mathcal{Y}$ est l'espace d'étiquette et $f(\cdot)$ est la fonction de prédiction, avec :

$$\begin{array}{rccc} f & : & \mathcal{X} & \rightarrow & \mathcal{Y} \\ & & X & \mapsto & f(X) \end{array}$$

$f(\cdot)$ est appris à partir des données d'apprentissage $\{x_i, y_i\}$, *où* $x_i \in X$, et $y_i \in \mathcal{Y}$. La fonction $f(\cdot)$ peut être utilisée pour prédire l'étiquette correspondante d'une nouvelle instance $x$. D'un point de vue probabiliste, $f(x)$ peut s'écrire comme une probabilité conditionnelle $P(y|x)$.

Dans la suite on considère un seule domaine source $D_s$ et un seule domaine cible $D_t$.

**Notations 3.4.1**

1. On note le domaine source $D_s$ par $D_s = \{(x_{s_1}, y_{s_1}), (x_{s_2}, y_{s_2}), ..., (x_{s_{n_s}}, y_{s_{n_s}})\}$ avec $x_{s_i} \in \mathcal{X}_s$ et $y_{s_i} \in \mathcal{Y}_s$, $i \in \{1, 2, ..., n_s\}$
2. De même on note $D_t$ par $D_t = \{(x_{t_1}, y_{t_1}), (x_{t_2}, y_{t_2}), ..., (x_{t_{n_t}}, y_{t_{n_t}})\}$ avec $x_{t_i} \in \mathcal{X}_t$ et $y_{t_i} \in \mathcal{Y}_t$, $i \in \{1, 2, ..., n_t\}$.

**Remarque 3.4.1**

Dans la plus part des cas $0 \leq n_t << n_s$.

**Définition 3.4.3: transfer learning [41]**

Étant donné un domaine source $D_s$ et une tâche d'apprentissage $\mathcal{T}_s$, un domaine cible $D_t$ et une tâche d'apprentissage $\mathcal{T}_t$, **l'apprentissage par transfert** consiste à améliorer l'apprentissage de la fonction prédictive cible $f_t(\cdot)$ dans $D_t$ définie par :

$$\begin{array}{rccc} f & : & \mathcal{X}_t & \rightarrow & \mathcal{Y}_t \\ & & X_t & \mapsto & f_t(X_t) \end{array}$$

en utilisant les connaissances de $D_s$ et $\mathcal{T}_s$, avec $D_s \neq D_t$, ou $\mathcal{T}_s \neq \mathcal{T}_t$.

Dans la définition ci-dessus, un domaine est un couple $D = \{\mathcal{X}, P(X)\}$, Ainsi la condition $D_s \neq D_t$ implique $\mathcal{X}_s \neq \mathcal{X}_t$ ou $P(X_s) \neq P(X_t)$. De même une tache est un couple $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$, Ainsi la condition $\mathcal{T}_s \neq \mathcal{T}_t$ implique que $\mathcal{Y}_s \neq \mathcal{Y}_t$ ou $f_s(\cdot) \neq f_t(\cdot)$. Lorsqu'il existe une relation, explicite ou implicite, entre les espaces de caractéristiques des deux domaines, on dit que les domaines source et cible sont **liés**.

## 3.5 Catégorisation des techniques d'apprentissage par transfert

Dans le cadre de l'apprentissage par transfert, nous avons trois questions de recherche principales :

1. **Que veut-on transférer ?**
2. **Comment va-t-on réaliser le transfert ?**
3. **Quand veut-on transférer ?.**

« Que veut-on transférer » demande quelle partie des connaissances peut être transférée entre les domaines ou les tâches. Certaines connaissances sont spécifiques à des domaines ou à des tâches individuels, et certaines connaissances peuvent être communes à différents domaines, de sorte qu'elles peuvent contribuer à améliorer les performances du domaine ou de la tâche cible. Après avoir découvert quelles connaissances peuvent être transférées, des algorithmes d'apprentissage doivent être développés pour transférer les connaissances, ce qui correspond à la question « Comment va-t-on réaliser le transfert »
« Quand veut-on transférer » demande dans quelles situations, le transfert doit être effec-

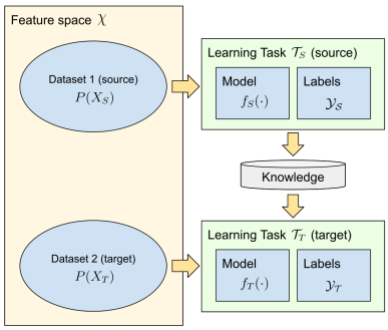



FIGURE 3.1 – Schéma d'apprentissage par transfert

tué. De même, nous sommes intéressés à savoir dans quelles situations, les connaissances ne doivent pas être transférées. Dans certaines situations, lorsque le domaine source et le domaine cible ne sont pas liés l'un à l'autre, le transfert par force brute peut échouer.

Dans le pire des cas, cela peut même nuire aux performances d'apprentissage dans le domaine cible, ce qu'on appelle le "transfert négatif". La plupart des travaux actuels sur l'apprentissage par transfert se concentrent sur « Que veut-on transférer » et « Comment va-t-on réaliser le transfert », en supposant implicitement que les domaines source et cible sont liés l'un à l'autre. Cependant, comment éviter le transfert négatif est une question ouverte importante qui attire de plus en plus l'attention à l'avenir.
En 2010 SJ PAN et al.[41] ont classer l'apprentissage par transfert en trois classes qui sont : l'apprentissage par transfert inductif, l'apprentissage par transfert transductif et l'apprentissage par transfert non supervisé.
On se basant sur la définition de l'apprentissage par transfert, nous résumons la relation entre l'apprentissage automatique traditionnel et divers cadres d'apprentissage par transfert dans le tableau 3.1 en fonction de situations différentes entre les domaines et les tâches source et cible.

| type de TL | domaines connexes | Domaine source étiquette | Domaine cible étiquette | Taches |
|---|---|---|---|---|
| TL inductif | apprentissage multi-tache | disponible | non disponible | Régression, classification |
| | apprentissage autodidacte | non disponible | disponible | Régression, classification |
| TL transdnductif | adaptation de domaine, | disponible | non disponible | Régression, classification |
| TL non supervisé | | non disponible | non disponible | clustring |

TABLE 3.1 – Relation entre l'apprentissage traditionnel et divers cadres d'apprentissage par transfert

1. Dans le cadre de l'apprentissage par transfert inductif, la tâche cible est différent de la tâche source, que les domaines source et cible soient identiques ou non. Dans ce cas, certaines données étiquetées dans le domaine cible sont nécessaires pour induire un modèle prédictif objectif à utiliser dans le domaine cible $f_t(\cdot)$. De plus, selon différentes situations de données étiquetées et non étiquetées dans le domaine source, nous pouvons encore catégoriser le cadre d'apprentissage par transfert inductif en deux cas :
   (a) De nombreuses données étiquetées dans le domaine source sont disponibles. Dans ce cas, le cadre d'apprentissage par transfert inductif est similaire au cadre d'apprentissage multi-tâche. Cependant, le cadre d'apprentissage par transfert inductif vise uniquement à atteindre des performances élevées dans la tâche cible en transférant les connaissances de la tâche source, tandis que l'apprentissage multi-tâche [1] tente d'apprendre simultanément la tâche cible et la tâche source.

   (b) Aucune donnée étiquetée dans le domaine source n'est disponible. Dans ce cas, le cadre d'apprentissage par transfert inductif est similaire à l'autodidacte(sleftaught learning) [44]. Dans le cadre de l'apprentissage autodidacte, les espaces d'étiquettes entre les domaines source et cible peuvent être différents, ce qui implique que les informations secondaires du domaine source ne peuvent pas être utilisées directement. Ainsi, il est similaire au cadre d'apprentissage par transfert inductif où les données étiquetées dans le domaine source ne sont pas disponibles.

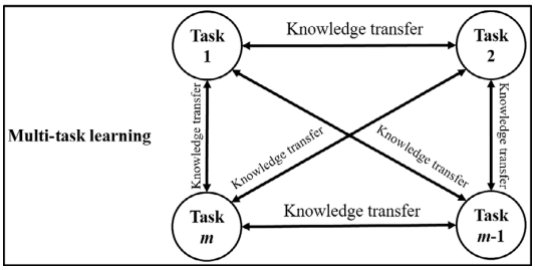



FIGURE 3.2 – Schéma d'apprentissage multi-tache

2. Dans le cadre de l'apprentissage par transfert transductif, la source et les tâches cibles sont les mêmes, tandis que les domaines source et cibles sont différents. Dans cette situation, aucune donnée étiquetée dans le domaine cible n'est disponible alors que de nombreuses données étiquetées dans le domaine source sont disponibles. De plus, selon différentes situations nous pouvons en outre classer le cadre d'apprentissage par transfert transductif en deux cas :
   (a) Les espaces de caractéristiques entre les domaines source et cible sont différents, $\mathcal{X}_s \neq \mathcal{X}_t$
   (b) Les espaces de caractéristiques entre les domaines sont les même, $\mathcal{X}_s = \mathcal{X}_t$ mais les distributions de probabilité marginales des données d'entrée sont différentes, $P(X_s) \neq P(X_t)$
3. Enfin, le cadre de l'apprentissage par transfert non supervisé est similaire au cadre de l'apprentissage par transfert inductif, la tâche cible est différente mais liée à la tâche source. Cependant, l'apprentissage par transfert non supervisé se concentre sur la résolution de tâches d'apprentissage non supervisées dans le domaine cible, tels que le clustering, la réduction de la dimension et l'estimation de la densité. Dans ce cas, il n'y a pas de données d'apprentissage étiquetées disponibles dans les domaines source et cible.

## 3.6 Les approches de l'apprentissage par transfert

Les approches de l'apprentissage par transfert dans les trois contextes différents ci-dessus peuvent être résumées en quatre cas basés sur « Quoi veut-on transférer » :

1. transfert d'instance qui suppose que certaines parties des données dans le domaine source peuvent être réutilisées pour l'apprentissage dans le domaine cible par re-pondération.
2. transfert de représentation de caractéristiques. L'idée intuitive derrière ce cas est d'apprendre une "bonne" représentation des caractéristiques pour le domaine cible. Dans ce cas, les connaissances utilisées pour le transfert entre les domaines sont codées dans la représentation des caractéristiques apprises.
3. transfert des paramètres suppose que les tâches sources et les tâches cibles partagent certains paramètres ou distributions a priori des hyper-paramètres des modèles. Les connaissances transférées sont encodées dans les paramètres partagés ou a priori.
4. transfert de connaissances relationnelles qui traite de l'apprentissage par transfert pour les domaines relationnels. L'hypothèse de base derrière ce contexte est que

certaines relations entre les données dans les domaines source et cible sont similaires. Ainsi, la connaissance à transférer est la relation entre les données.

Le tableau 3.2 montre les cas où les différentes approches sont utilisées pour chaque contexte d'apprentissage par transfert.

| | inductive TL | transductivr TL | TL non-supervisé |
|---|---|---|---|
| transfert d'instance | ✓ | ✓ | |
| transfert de représentation de caractéristique | ✓ | ✓ | ✓ |
| transfert de paramètres | ✓ | | |
| transfert de connaissances relationnel | ✓ | | |

TABLE 3.2 – Différentes approches utilisées pour les différents contextes

### 3.6.1 Apprentissage par transfert inductif

**Définition 3.6.1: [41]**

Étant donné $D_s$, $\mathcal{T}_s$, $D_t$ et $\mathcal{T}_t$, l'apprentissage par transfert inductif consiste à améliorer l'apprentissage de la fonction de prédiction cible $f_t(\cdot)$ dans $D_t$ en utilisant les connaissance de $D_s$ et $\mathcal{T}_s$, avec $\mathcal{T}_s \neq \mathcal{T}_t$.

1. Transfert des connaissance des instances . L'approche par transfert d'instance du cadre d'apprentissage par transfert inductif est intuitivement attrayante : bien que les données du domaine source ne puissent pas être réutilisées directement, certaines parties des données peuvent toujours être réutilisées avec quelques données étiquetées dans le domaine cible.
2. Transfert de représentation de caractéristiques
   L'approche de transfert de représentation de caractéristiques au problème d'apprentissage par transfert inductif vise à trouver de "bonnes" représentations de caractéristiques pour minimiser la divergence de domaine et l'erreur de modèle de classification ou de régression. Les stratégies pour trouver de « bonnes » représentations de caractéristiques sont différentes pour différents types du données de domaine source. Si de nombreuses données étiquetées dans le domaine source sont disponibles, des méthodes d'apprentissage supervisé peuvent être utilisées pour construire une représentation des caractéristiques. Ceci est similaire à l'apprentissage des caractéristiques communes dans le domaine de l'apprentissage multi-tâche. Si aucune donnée étiquetée dans le domaine source n'est disponible, des méthodes d'apprentissage non supervisées sont proposées pour construire la représentation des caractéristiques.
   (a) les méthodes de construction de caractéristiques supervisées. L'idée est d'apprendre une représentation de faible dimension qui est partagée entre les tâches connexes. De plus, la nouvelle représentation apprise peut également réduire l'erreur de modèle de classification ou de régression de chaque tâche. Dans le transfert inductif les caractéristiques communes peuvent être apprises en résolvant le problème d'optimisation, donné par :

$$\arg\min_{A,U} \sum_{t\in\{T,S\}} \sum_{i=1}^{n_t} L(y_{t_i}, \langle a_t, U^t x_{t_i} \rangle) + \gamma ||A||_{2,1}^2 \tag{3.1}$$

avec, $T, S$ est la tache cible et source respectivement, $A = [a_s, a_t] \in \mathbb{R}^{2d}$ est la matrices des paramètres, $U$ est une matrice orthogonale pour transformer les données originales de grande dimension à des représentations de faible dimension, $||A||_{r,p} := \left(\sum_{i=1}^{d} ||a^i||_r^p\right)^{\frac{1}{p}}$

(b) Construction de caractéristique non supervisée L'idée de base de cette approche consiste deux étapes :

i. Dans la première étape, les vecteurs de base de niveau supérieur $b = \{b_1, b_2, ..., b_s\}$ sont appris sur les données du domaine source en résolvant le problème d'optimisation :

$$\min_{a,b} \left|\left| x_{s_i} - \sum_j a_{s_i}^j b_j \right|\right|_2^2 + \beta ||a_{s_i}||_1 \tag{3.2}$$

$$s.t. \qquad ||b_j||_2 \leq 1, \forall j \in \{1, ..., s\} \tag{3.3}$$

*où* $a_{s_i}^j$ est la nouvelle représentation de la base $b_j$ pour l'entrée $x_{s_i}$, et $\beta$ est le paramètre de régularisation.

ii. après l'apprentissage des vecteurs de base $b$, dans la seconde étape, on résout le problème d'optimisation :

$$a_{t_i}^* = \arg\min_{a_{t_i}} \left|\left| x_{t_i} - \sum_j a_{t_i}^j b_j \right|\right|_2^2 + \beta ||a_{t_i}||_1 \tag{3.4}$$

Enfin, des algorithmes discriminatifs peuvent être appliqués a $a_{t_i}^*$ avec les étiquettes correspondantes pour former des modèles de classification ou de régression à utiliser dans le domaine cible. Un inconvénient de cette méthode est que les vecteurs de base dits de niveau supérieur appris sur le domaine source dans le problème d'optimisation peuvent ne pas convenir à une utilisation dans le domaine cible

3. Transfert de connaissances des paramètres.
La plupart des approches de transfert de paramètres pour le transfert inductif suppose que les modèles individuels pour les tâches connexes doivent partager certains paramètres ou distributions antérieures d'hyperparamètres. La plupart des approches décrites dans cette section, y compris un cadre de régularisation et un cadre bayésien hiérarchique, sont conçues pour fonctionner dans le cadre d'un apprentissage multitâche. Cependant, ils peuvent être facilement modifiés pour l'apprentissage par transfert. Ainsi, dans l'apprentissage multitâche, les poids des fonctions de perte pour les données source et cible sont les mêmes. En revanche, dans l'apprentissage par transfert, les poids des fonctions de perte pour différents domaines peuvent être différents. Intuitivement, nous pouvons attribuer un poids plus important à la fonction de perte du domaine cible pour nous assurer que nous pouvons obtenir de meilleures performances dans le domaine cible.

4. Transférer les connaissances relationnelles.
Différente des trois autres contextes, l'approche de transfert de connaissances relationnelles traite des problèmes d'apprentissage par transfert dans des domaines relationnels, où les données ne sont pas i.i.d. et peuvent être représentés par de multiples relations Cette approche ne suppose pas que les données tirées de chaque

domaine soient indépendantes et identiquement distribuées (i.i.d.) comme traditionnellement supposé. Il essaie de transférer la relation entre les données d'un domaine source vers un domaine cible. Dans ce contexte, des techniques d'apprentissage relationnel statistique sont proposées pour résoudre ces problèmes.

### 3.6.2 Apprentissage par transfert transductif

Le terme apprentissage par transfert transductif a été proposé pour la première fois par Arnold et al. [2], où ils exigeaient que les tâches source et cible soient les mêmes, bien que les domaines puissent être différents. En plus de ces conditions, ils ont en outre exigé que toutes les données non étiquetées dans le domaine cible soient disponibles au moment de l'apprentissage, cette condition peut être assouplie ; au lieu de cela, dans notre définition du cadre d'apprentissage par transfert transductif, nous exigeons seulement qu'une partie des données cibles non étiquetées soit vue au moment de l'apprentissage afin d'obtenir la probabilité marginale pour les données cibles.

**Définition 3.6.2: Apprentissage par transfert transductive [41]**

Étant donné $D_s$, $\mathcal{T}_s$, $D_t$ et $\mathcal{T}_s$, l'apprentissage par transfert transductive consiste à améliorer l'apprentissage de la fonction de prédiction cible $f_t(\cdot)$ dans $D_t$ en utilisant les connaissance de $D_s$ et $\mathcal{T}_s$, avec $D_s \neq D_t$ et $\mathcal{T}_s = \mathcal{T}_t$, de plus quelque donnée étiquette dans le domaine cible doivent être à lieu au moment de l'apprentissage.

Transférer la connaissance des instances. La plupart des approches de transfert d'instance dans le cadre de l'apprentissage par transfert transductif sont motivées par l'échantillonnage d'importance. Pour voir comment les méthodes d'échantillonnage basées sur l'importance peuvent aider dans ce contexte, nous examinons d'abord le problème de la minimisation du risque empirique (ERM). En général, nous pourrions vouloir apprendre les paramètres optimaux du modèle en minimisant le risque attendu

$$\theta^* = \arg\min_{\theta\in\Theta} \mathbb{E}_{(x,y)\in P}\left[l(x,y,\theta)\right] \tag{3.5}$$

puisque il est difficile d'estimer la distribution de probabilité $P$, on minimise le risque

$$\theta^* = \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n}\left[l(x_i,y_i,\theta)\right] \tag{3.6}$$

dans le cadre du transfert transductive, on veut faire l'apprentissage d'un modèle optimale pour le domaine cible en minimisant le risque :

$$\theta^* = \arg\min_{\theta\in\Theta} \sum_{(x,y)\in D_t} P(D_t)\left[l(x,y,\theta)\right] \tag{3.7}$$

puisque aucune donnée étiquette n'est observé dans le domaine cible, nous devons plutôt apprendre un modèle à partir des données du domaine source. Si $P(D_s) = P(D_t)$ alors nous pouvons simplement apprendre le modèle en résolvant le problème d'optimisation suivant pour une utilisation dans le domaine cible

$$\theta^* = \arg\min_{\theta\in\Theta} \sum_{(x,y)\in D_s} P(D_s)\left[l(x,y,\theta)\right] \tag{3.8}$$

### 3.6.3 Apprentissage par transfert non supervisé

**Définition 3.6.3: Apprentissage par transfert non supervisé [41]**

Étant donné $D_s$, $\mathcal{T}_s$, $D_t$ et $\mathcal{T}_s$, l'apprentissage par transfert non supervisé consiste à améliorer l'apprentissage de la fonction de prédiction cible $f_t(\cdot)$ dans $D_t$ en utilisant les connaissance de $D_s$ et $\mathcal{T}_s$, avec $\mathcal{T}_s \neq \mathcal{T}_t$ et $\mathcal{Y}_s$ , $\mathcal{Y}_t$ ne sont pas observables..

### 3.6.4 Transfert négatif

Les cas dont nous avons discuté jusqu'à présent parlent d'améliorations des tâches cibles basées sur le transfert de connaissances à partir de la tâche source. Il existe des cas où l'apprentissage par transfert peut entraîner une baisse des performances. Le transfert négatif fait référence à des scénarios où le transfert de connaissances de la source à la cible ne conduit à aucune amélioration, mais provoque plutôt une baisse de la performance globale de la tâche cible. Il peut y avoir diverses raisons à un transfert négatif, comme des cas où la tâche source n'est pas suffisamment liée à la tâche cible ou si la méthode de transfert ne peut pas très bien tirer parti de la relation entre les tâches source et cible. Éviter le transfert négatif est très important et nécessite une enquête approfondie.

## 3.7 Apprentissage par transfert profond

**Définition 3.7.1: [55]**

Etant donné une tache d'apprentissage par transfert définie par $\langle D_s, \mathcal{T}_s, D_t, \mathcal{T}_t, f_{\mathcal{T}}(\cdot)\rangle$. Il s'agit d'une tâche d'apprentissage par transfert profond lorsque $f_{\mathcal{T}}(\cdot)$ est une fonction non linéaire qui reflète un réseau de neurones profond.

L'apprentissage par transfert profond basé sur le réseau fait référence à la réutilisation du réseau partiel qui a été pré-formé dans le domaine source, y compris sa structure de réseau et ses paramètres de connexion, le transfère pour faire partie du réseau neuronal profond utilisé dans le domaine cible.

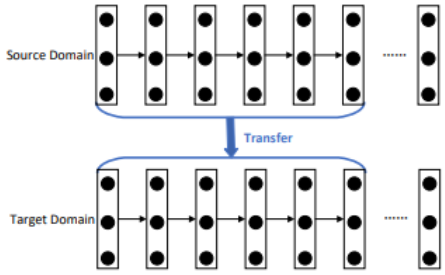



FIGURE 3.3 – Schéma d'apprentissage par transfert basé sur les réseaux

## 3.8 Apprentissage par transfert via la régularisation $\ell_1$

### 3.8.1 Préliminaire et notion de base

dans cette partie on donne quelques résultat sur les variables aléatoire sous-gaussien, ainsi une condition d'optimalité des fonction non partout différentiable qui seront utilisée dans la suite de ce chapitre.

#### 3.8.1.1 Variable aléatoire sous-gaussien

**Définition 3.8.1: sous gaussienne**

Une variable aléatoire centrée $X$ est dite sous-gaussienne de facteur variance $\sigma$ si :

$$\mathbf{E}\left(\exp(tX)\right) \leq \exp\left(\frac{\sigma^2 t^2}{2}\right), \qquad \forall t \in \mathbb{R}$$

**Théorème 3.8.1: [11]**

Soit $X_1, \ldots, X_p$ des variables aléatoire subG $(\sigma^2)$ alors le vecteur $X = (X_i) \in \mathbb{R}^p$ est subG $(\sigma^2)$.

**Lemme 3.8.1: [11]**

Soit $X_1, \ldots, X_n$ $n$ variables aléatoires sous-gaussien centrées de variance $\sigma^2$. Alors pour tout $t > 0$ on a :

$$P\left\{\max_{i=1,\ldots,n} X_i > t\right\} \leq n\mathrm{e}^{-\frac{t^2}{2\sigma^2}} \tag{3.9}$$

*Démonstration.*

$$\begin{aligned} P\left(\max_{i=1,\ldots,n} X_i > t\right) &= P\left(\bigcup_{i=1}^{n} [X_i > t]\right) \\ &\leq \sum_{i=1}^{n} P\left[X_i > t\right] \\ &= n \exp\left(-\frac{t^2}{2\sigma^2}\right) \end{aligned}$$

□

#### 3.8.1.2 Condition d'optimalité d'une fonction non partout différentiable

**Définition 3.8.2: [52]**

Le domaine d'une fonction $f$ est noté $dom(f)$. Il est défini par :

$$dom(f) = \{x \in \mathbb{R}^n : f(x) < \infty\}$$

**Définition 3.8.3: Sous-gradient et sous-différentiel[52]**

Soit $f$ une fonction convexe. Un vecteur $\eta \in \mathbb{R}^n$ est appelé sous-gradient de $f$ au point $x_0 \in \text{dom } (f)$ si

$$\forall x \in \text{dom}(f), \quad f(x) \geq f(x_0) + \langle \eta, x - x_0 \rangle$$

L'ensemble de tous les sous-gradients en $x_0$ est appelé sous-différentiel de $f$. Il est noté $\partial f(x_0)$.

**Exemple** :
Soit $f : x \in \mathbb{R} \mapsto |x|$. Calculons les sous-différentielle de $f$ en tout point $x$ de $\mathbb{R}$.

$$\partial |x| = \begin{cases} 1 & si \quad x > 0 \\ -1 & si \quad x < 0 \\ [-1, 1] & si \quad x = 0 \end{cases}$$

**Lemme 3.8.2: [52]**

Le sous-différentiel $\partial f(x_0) = \{\eta \in \mathbb{R}^n, \forall x \in \mathbb{R}^n, f(x) \geq f(x_0) + \langle \eta, x - x_0 \rangle\}$ est un ensemble convexe fermé.

*Démonstration.* Soient $g_1$ et $g_2$ des éléments de $\partial f(x_0)$. On a $\forall y \in \mathbb{R}^n$ :

$$f(y) \geq f(x) + \langle g_1, y - x \rangle$$
$$f(y) \geq f(x) + \langle g_2, y - x \rangle$$

Soit $\alpha \in [0, 1]$
En multipliant la première inégalité par $\alpha$, la deuxième par $(1 - \alpha)$ et en sommant, on voit que

$$\alpha g_1 + (1 - \alpha) g_2 \in \partial f(x_0)$$

. □

**Théorème 3.8.2: [52]**

Si $f$ est une fonction convexe et $x \in dom(f)$, alors

$$x^* \text{ est un minimum de} f \iff 0 \in \partial f(x^*)$$

.

*Démonstration.* Si $0 \in \partial f(x^*)$, alors

$$f(x) \geq f(x^*) + \langle 0, x - x^* \rangle \geq f(x^*), \forall x \in dom(f)$$

Réciproquement, si $f(x) \geq f(x^*), \forall x \in \text{dom}(f)$, alors $0 \in \partial f(*)$ par définition du sous-différentiel. □

**Théorème 3.8.3: Moreau-Rockafellar [32]**

soit $f, g : \mathbb{R}^n \to (-\infty, \infty]$ deux fonctions convexes. Alors pour tout $x_0 \in \mathbb{R}^n$

$$\partial f(x_0) + \partial g(x_0) \subset \partial(f+g)(x_0) \tag{3.10}$$

si $\mathrm{dom}(f) \cap \mathrm{dom}(g) \neq \emptyset$. Alors pour tout $x_0 \in \mathbb{R}^n$,

$$\partial f(x_0) + \partial g(x_0) = \partial(f+g)(x_0) \tag{3.11}$$

### 3.8.2 Méthode de transfert des paramètres

On considère le problème de la régression parcimonieuse(sparse regression). Les modèles parcimonieuse sont largement utilisés dans la prise de décision car ils ont peu de caractéristiques actives et donc faciles à obtenir. Cependant, les méthodes de régression parcimonieuses existantes ne sont pas nécessairement efficaces pour la prise de décision de routine, car elles peuvent modifier considérablement les paramètres même lorsque les données ne changent que légèrement.
Dans cette section on va étudié une méthodes pour transférer les connaissances depuis le modèle source vers le modèle cible. Cette méthode à été proposé par M.Takada et al.[53].

#### 3.8.2.1 Transfert LASSO

Soit $X_i \in \mathcal{X}$ et $Y_i \in \mathbb{R}$ respectivement la caractéristique et la Réponse associé, pour $i \in \{1, 2, ..., n\}$. On considère la fonction linéaire :

$$f_\beta(\cdot) = \sum_{j=1}^{p} \beta_j \psi_j(\cdot) \tag{3.12}$$

avec $\beta = (\beta_j) \in \mathbb{R}^p$ et $\psi_j(\cdot) : \mathcal{X} \to \mathbb{R}$. Soit la fonction cible :

$$f^*(\cdot) = f_{\beta^*}(\cdot) := \sum_{j=1}^{p} \beta_j^* \psi_j(\cdot) \quad et \quad \epsilon_i := Y_i - f^*(X_i) \tag{3.13}$$

notation matricielle :

$$f^* := X\beta^* \quad et \quad \epsilon := y - f^*$$

avec $f^* = (f^*(X_i)) \in \mathbb{R}^n$, $X = (\psi_j(X_i)) \in \mathbb{R}^{n \times p}$, $\beta^* = (\beta_j^*) \in \mathbb{R}^p$, $y = (Y_i) \in \mathbb{R}^n$.
on pose :

$$S := supp(\beta_j^*) = \{j \in \{1, 2, ..., p\} : \beta_j^* \neq 0\} \quad et \quad s = |S|$$

dans une grande dimension, l'approche raisonnable pour estimer $\beta^*$ consiste à supposer la parcimonieuse de $\beta^*$ c'est à dire $s << p$, et de résoudre le problème **LASSO** :

$$\min_{\beta \in \mathbb{R}^p} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (Y_i - f_\beta(X_i))^2 + \lambda ||\beta||_1 \right\}$$

**LASSO** réduit l'estimation à zero et donne une solution creuse.
Soit $\tilde{\beta}$ une estimation initiale (estimation de la source).nous employons la Régularisation $\ell_1$ de la différence entre $\beta$ et $\tilde{\beta}$ et l'ajouté au Lasso ordinaire :

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (Y_i - f_\beta(X_i))^2 + \lambda \left( \alpha ||\beta||_1 + (1 - \alpha) ||\beta - \tilde{\beta}||_1 \right) \right\} \tag{3.14}$$

On pose

$$\mathcal{L}(\beta, \tilde{\beta}) := \frac{1}{2n} \sum_{i=1}^{n} (Y_i - f_\beta(X_i))^2 + \lambda \left( \alpha ||\beta||_1 + (1 - \alpha) ||\beta - \tilde{\beta}||_1 \right) \tag{3.15}$$

avec $\lambda = \lambda_n > 0$ et $0 \leq \alpha \leq 1$ sont les paramètres de régularisation. on appelle cette méthode Transfer LASSO .

**Remarque 3.8.1**

1. Le premier terme de Régularisation réduit l'estimation à zero et donne une solution creuse
2. Le deuxième terme de régularisation réduit l'estimation de $\beta^*$ à l'estimation initiale $\tilde{\beta}$.
3. Le paramètre $\alpha$ contrôle l'équivalence entre le transfert et le rejet des connaissances de la source.

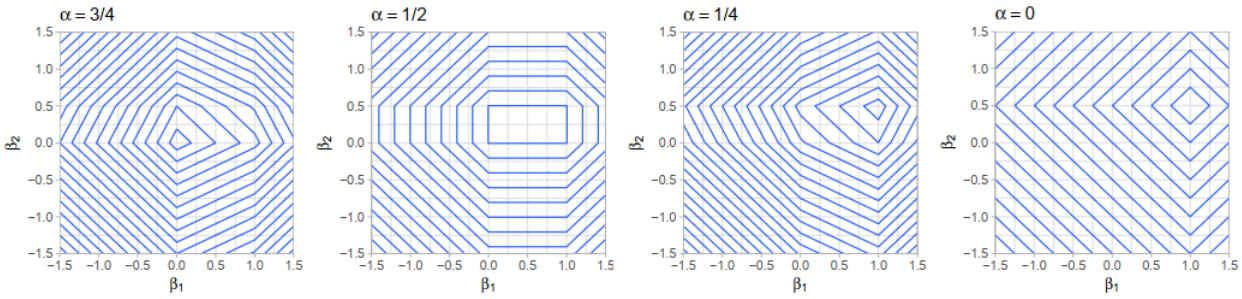



FIGURE 3.4 – Contours de la fonction $\mathcal{L}$ pour $\alpha = \frac{3}{4}, \alpha = \frac{1}{2}, \alpha = \frac{1}{4}, \alpha = 0$,avec $\tilde{\beta} = (1, \frac{1}{2})^t$

Les contours sont des polygones pointés en $\beta_j = 0$ et $\beta_j = \tilde{\beta}_j$ de sorte que cette estimation peut se réduire à zéro et à l'estimation initiale. Le paramètre de régularisation $\alpha$ ; contrôle les forces de retrait à zéro et l'estimation initiale.

#### 3.8.2.2 Algorithme de descente de coordonnées et fonction Soft-Threshold

On donne un algorithme de descente de coordonnées pour Transfert Lasso. Il est garanti de converger vers une solution optimale globale [59]. En reprenant le problème 3.14, on a :

$$\partial_{\beta_j}\mathcal{L}(\beta, \tilde{\beta}) = -\frac{1}{n}X_j^t(y - X_{-j}\beta_{-j}) + \beta_j + \lambda\alpha sgn(\beta_j) + \lambda(1-\alpha)sgn(\beta_j - \tilde{\beta}_j)$$

avec $\partial_{\beta_j}\mathcal{L}(\beta, \tilde{\beta})$ étant la sous-différentielle de $\mathcal{L}$ au point $\beta_j$ et $X_j$ et $X_{-j}$ respectivement le $j^{eme}$ colonne et $X$ sans le $j^{eme}$ colonne, $sgn(\cdot)$ est la fonction signe. on a :

$$\beta_j \longleftarrow \mathcal{T}\left(\frac{1}{n}X_j^t(y - X_{-j}\beta_{-j}), \lambda, \lambda(2\alpha - 1), \tilde{\beta}_j\right) \qquad j \in \{1, 2, ..., p\}$$

avec

$$\mathcal{T}(z, \gamma_1, \gamma_2, b) = \begin{cases} 0 & pour & -\gamma_1 \leq z \leq \gamma_2 \\ b & pour & \gamma_2 + b \leq z \leq \gamma_1 + b \\ z - \gamma_2 sgn(b) pour & & \gamma_2 \leq z \leq \gamma_2 + b \\ z - \gamma_1 sgn(z) pour & & sinon \end{cases}$$

pour $b \geq 0$.

$$\mathcal{T}(z, \gamma_1, \gamma_2, b) = \begin{cases} 0 & pour & -\gamma_2 \leq z \leq \gamma_1 \\ b & pour & -\gamma_1 + b \leq z \leq -\gamma_2 + b \\ z - \gamma_2 sgn(b) pour & & -\gamma_2 + b \leq z \leq -\gamma_2 \\ z - \gamma_1 sgn(z) pour & & sinon \end{cases}$$

pour $b \leq 0$.

*Démonstration.* La fonction $\mathcal{L}(\beta, \tilde{\beta})$ est convexe comme somme de fonctions convexes, par contre elle n'est pas différentiable car

$$x \mapsto |x|$$

ne l'est pas. En utilisant la condition d'optimalité donnée par le théorème 3.8.1.2

$$x^* \text{ est un minimum de f avec f convexe } \iff 0 \in \partial f(x^*)$$

avec $\partial f$ est la sous-différentielle de $f$.
on a :

$$\begin{aligned}\mathcal{L}(\beta, \tilde{\beta}) &= \frac{1}{2n}\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{n}\beta_j\psi_j(X_i))^2 + \lambda\left(\alpha\sum_{j=1}^{p}|\beta_j| + (1-\alpha)\sum_{j=1}^{p}|\beta_j - \tilde{\beta}_j|\right) \\ &= \frac{1}{2n}\sum_{i=1}^{n}\left(Y_i - \sum_{k=1,k\neq j}^{n}\beta_k\psi_k(X_i) - \beta_j\psi_j(X_i)\right)^2 + \\ &\lambda\left(\alpha\sum_{k=1,k,\neq j}^{p}|\beta_k| + \alpha|\beta_j| + (1-\alpha)\sum_{k=1}^{p}|\beta_k - \tilde{\beta}_k| + (1-\alpha)|\beta_j - \tilde{\beta}_j|\right)\end{aligned}$$

on pose

$$g(\beta_j) = \frac{1}{2n}\sum_{i=1}^{n}\left(Y_i - \sum_{k=1,k\neq j}^{n}\beta_k\psi_k(X_i) - \beta_j\psi_j(X_i)\right)^2$$

on a alors :

$$\partial_{\beta_j} g(\beta_j) = -\frac{1}{n}X_j^t(y - X_{-j}\beta_{-j}) + \beta_j\frac{\sum_{i=1}^{n}\psi_j^2(X_i)}{n}$$

avec

$$\frac{\sum_{i=1}^{n}\psi_j^2(X_i)}{n} = 1 \qquad \text{pour chaque } j \in \{1, 2, ..., p\}$$

d'autre part on a :

$$\partial_{\beta_j}(\lambda\alpha|\beta_j|) = \begin{cases} \lambda\alpha \quad si \quad \beta_j > 0 \\ [-\lambda\alpha, \lambda\alpha] \quad si \quad \beta_j = 0 \\ -\lambda\alpha \quad si \quad \beta_j < 0 \end{cases}$$

et

$$\partial_{\beta_j}\left(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j|\right) = \begin{cases} \lambda(1-\alpha) \quad si \quad \beta_j - \tilde{\beta}_j > 0 \\ [-\lambda(1-\alpha), \lambda(1+\alpha)] \quad si \quad \beta_j - \tilde{\beta}_j = 0 \\ -\lambda(1-\alpha) \quad si \quad \beta_j - \tilde{\beta}_j < 0 \end{cases}$$

on pose :

$$z = \frac{1}{n}X_j^t(y - X_{-j}\beta_{-j}), \qquad \gamma_1 = \lambda, \qquad \gamma_2 = \lambda(2\alpha - 1), \qquad b = \beta_j$$

ainsi il y a 9 cas à distingué :
<u>cas 1</u> : $\beta_j = 0 \Longrightarrow \partial_{\beta_j}(\lambda\alpha|\beta_j|) = [-\lambda\alpha, \lambda\alpha]$

1. sous cas 1 :$\tilde{\beta}_j > 0 \Longrightarrow \partial_{\beta_j}(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j|) = -\lambda(1-\alpha)$
2. sous cas 2 :$\tilde{\beta}_j < 0 \Longrightarrow \partial_{\beta_j}(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j| = \lambda(1-\alpha)$
3. sous cas 3 :$\tilde{\beta}_j = 0 \Longrightarrow \partial_{\beta_j}(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j| = [-\lambda(1-\alpha), \lambda(1-\alpha)]$

cas 2 : $\beta_j > 0 \Longrightarrow \partial_{\beta_j} \lambda\alpha|\beta_j| = \lambda\alpha$

1. sous cas 1 :$\beta_j > \tilde{\beta}_j \Longrightarrow \partial_{\beta_j}(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j|) = \lambda(1-\alpha)$
2. sous cas 2 :$\tilde{\beta}_j > \beta_j \Longrightarrow \partial_{\beta_j}(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j|) = -\lambda(1-\alpha)$
3. sous cas 3 :$\tilde{\beta}_j = \beta_j \Longrightarrow \partial(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j|) = [-\lambda(1-\alpha), \lambda(1-\alpha)]$

cas 3 : $\beta_j < 0 \Longrightarrow \partial_{\beta_j}(\lambda\alpha|\beta_j|) = -\lambda\alpha$

1. sous cas 1 :$\beta_j > \tilde{\beta}_j \Longrightarrow \partial_{\beta_j}(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j|) = \lambda(1-\alpha)$
2. sous cas 2 :$\tilde{\beta}_j > \beta_j \Longrightarrow \partial_{\beta_j}(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j|) = -\lambda(1-\alpha)$
3. sous cas 3 :$\tilde{\beta}_j = \beta_j \Longrightarrow \partial(\lambda(1-\alpha)|\beta_j - \tilde{\beta}_j|) = [-\lambda(1-\alpha), \lambda(1-\alpha)]$

En discutant toutes les cas possible on trouve que :

$$\beta_j \longleftarrow \mathcal{T}(z, \gamma_1, \gamma_2, b) \text{ pour chaque } \quad j \in \{1, 2, ..., p\}$$

*où* $\mathcal{T}$ est la fonction "Soft-Threshold" . □

### 3.8.3 Propriétés théoriques du transfert LASSO

#### 3.8.3.1 Estimation d'erreur

**Hypothèse 1.** *On suppose que la suite* $\{\epsilon_i\}_{i=1}^n$ *de bruit, est une suite de variable aléatoire sous-gaussien i.i.d de variance* $\sigma$*, i.e*

$$\mathbf{E}(\exp(t\epsilon)) \leq \exp\left(\frac{\sigma^2 t^2}{2}\right), \quad \forall t \in \mathbb{R}$$

**Définition 3.8.4: Restriction de valeur propre généralisé(GRE)[53]**

On dit que la condition GRE est vérifie pour un sous-ensemble $B \subset \mathbb{R}^p$ si :

$$\phi = \phi(B) := \inf_{v \in B} \frac{v^t \frac{1}{n} X^t X v}{||v||_2^2} > 0$$

La condition GRE est la généralisation de la condition de restriction de valeur propre [7].
On pose $\Delta = \tilde{\beta} - \beta^*$, et $v_s = v_{|S}$ la restriction de $v$ a l'ensemble des indices $S$.

**Théorème 3.8.4: Estimation d'erreur[53]**

Supposons que l'hypothèse 1 est vérifiée , et que la condition GRE est vérifiée pour $B = B(\alpha, c, \Delta)$, avec :

$$B := \{v \in \mathbb{R}^p : (\alpha - c)||v_{s^c}||_1 + (1-\alpha)||v - \Delta||_1 \leq (\alpha + c)||v_s||_1 + (1-\alpha)||\Delta||_1\}$$

pour une certaine constante $c > 0$. Alors, on a :

$$||\hat{\beta} - \beta^*||_2^2 \leq \frac{(\alpha+c)^2\lambda_n^2 s}{\phi^2}\left[1 + \sqrt{1 + \frac{2(1-\alpha)\phi||\Delta||_1}{(\alpha+c)^2\lambda_n s}}\right]^2$$

avec une probabilité au moins $1 - \nu_{n,v}$, *où* $\nu_{n,v} = \exp\left(\frac{-nc^2\lambda_n^2}{2\sigma^2} + log(2p)\right)$

*Démonstration.*

$$\begin{aligned}
&\mathcal{L}(\hat{\beta}, \tilde{\beta}) \leq \mathcal{L}(\beta^*, \tilde{\beta}) \\
&\qquad \Longleftrightarrow \frac{1}{2n}||X\beta^* + \epsilon - X\hat{\beta}||_2^2 + \lambda_n\alpha||\hat{\beta}||_1 + \lambda_n(1-\alpha)||\hat{\beta} - \tilde{\beta}||_1 \\
&\qquad\quad \leq \frac{1}{2n}||\epsilon||_2^2 + \lambda_n\alpha||\beta^*||_1 + \lambda_n(1-\alpha)||\tilde{\beta} - \beta^*||_1 \\
&\qquad\quad \Longleftrightarrow \frac{1}{2n}||X(\hat{\beta} - \beta^*)||_2^2 + \lambda_n\alpha||\hat{\beta}||_1 + \lambda_n(1-\alpha)||\hat{\beta} - \tilde{\beta}||_1 \\
&\qquad\quad \leq \frac{1}{n}\epsilon^t X(\hat{\beta} - \beta^*) + \lambda_n\alpha||\beta^*||_1 + \lambda_n(1-\alpha)||\tilde{\beta} - \beta^*||_1 \\
&\qquad\quad \Longrightarrow \frac{1}{2n}||X(\hat{\beta} - \beta^*)||_2^2 + \lambda_n\alpha||\hat{\beta}||_1 + \lambda_n(1-\alpha)||\hat{\beta} - \tilde{\beta}||_1 \\
&\qquad\quad \leq \left|\left|\frac{1}{n}X^t\epsilon\right|\right|_\infty ||\hat{\beta} - \beta^*||_1 + \lambda_n\alpha||\beta^*||_1 + \lambda_n(1-\alpha)||\tilde{\beta} - \beta^*||_1
\end{aligned}$$

d'après l'hypothèse $\epsilon$ est une variable aléatoire sous-gaussien de variance $\sigma$, on a :

$$P\left(\left|\left|\frac{1}{n}X^t\epsilon\right|\right|_\infty \leq \gamma_n\right) \geq 1 - \exp\left(-\frac{n\gamma_n^2}{2\sigma^2} + log(2p)\right)$$

on prend $\gamma_n = c\lambda_n$, avec une probabilité au moins

$$1 - \exp\left(-\frac{n\gamma_n^2}{2\sigma^2} + log(2p)\right)$$

on a :

$$\begin{aligned}
&\tfrac{1}{2n}||X(\hat{\beta} - \beta^*)||_2^2 + \lambda_n\alpha||\hat{\beta}||_1 + \lambda_n(1-\alpha)||\hat{\beta} - \tilde{\beta}||_1 \\
&\leq c\lambda_n||\hat{\beta} - \beta^*||_1 + \lambda_n\alpha||\beta^*||_1 + \lambda_n(1-\alpha)||\tilde{\beta} - \beta^*||_1
\end{aligned}$$

$\Longleftrightarrow$

$$\begin{aligned}
&\tfrac{1}{2n}||X(\hat{\beta} - \beta^*)||_2^2 + \lambda_n\alpha||\hat{\beta}_s||_1 + \lambda_n\alpha||\hat{\beta}_{s^c}||_1 + \lambda_n(1-\alpha)||\hat{\beta}_s - \tilde{\beta}_s||_1 + \lambda_n(1-\alpha)||\hat{\beta}_{s^c} - \tilde{\beta}_{s^c}||_1 \\
&\quad \leq c\lambda_n||\hat{\beta}_s - \beta_s^*||_1 + c\lambda_n||\hat{\beta}_{s^c}||_1 + \lambda_n\alpha||\beta_s^*||_1 + \lambda_n(1-\alpha)||\tilde{\beta}_s - \beta_s^*||_1 + \lambda_n(1-\alpha)||\hat{\beta}_{s^c}||_1
\end{aligned}$$

car on a : $\forall k \in S^c = (supp(\beta^*))^c$, $\beta^*_k = 0 \Longrightarrow \beta^*_{s^c} = 0$
$\Longrightarrow \frac{1}{2n}||X(\hat{\beta} - \beta^*)||_2^2 + \lambda_n(\alpha - c)||\hat{\beta}_s||_1 + \lambda_n(1-\alpha)||\hat{\beta}_s - \tilde{\beta}_s||_1 + \lambda_n(1-\alpha)||\hat{\beta}_{s^c} - \tilde{\beta}_{s^c}||_1$
$\leq \lambda_n(\alpha + c)||\hat{\beta}_s - \beta^*_s||_1 + \lambda_n(1-\alpha)||\tilde{\beta}_s - \beta^*_s||_1 + \lambda_n(1-\alpha)||\tilde{\beta}_{s^c}||_1$
on a donc :

$$(\alpha - c)||\hat{\beta}_s||_1 + (1-\alpha)||\hat{\beta} - \tilde{\beta}||_1 \leq (\alpha + c)||\hat{\beta}_s - \beta^*_s||_1 + (1-\alpha)||\tilde{\beta} - \beta^*||_1$$

cela nous donne

$$\hat{\beta} - \beta^* \in B := \{v \in \mathbb{R}^p : (\alpha - c)||v_{s^c}|| + (1-\alpha)||v - \Delta||_1 \leq (\alpha + c)||v_s||_1 + (1-\alpha)||\Delta||_1\}$$

avec $\Delta := \tilde{\beta} - \beta^*$ D'autre part, on a :

$$\frac{1}{2n}||X(\hat{\beta} - \beta^*)||_2^2 \leq \lambda_n(\alpha + c)||\hat{\beta}_s - \beta^*_s||_1 + \lambda_n(1-\alpha)||\Delta||_1$$

. en utilisant la condition GRE, on a :

$$\begin{aligned}\frac{1}{2n}||X(\hat{\beta} - \beta^*)||_2^2 &= \frac{1}{2}(\hat{\beta} - \beta^*)^t(\frac{1}{n}X^tX)(\hat{\beta} - \beta^*) \\ &\geq \frac{\phi}{2}||\hat{\beta} - \beta^*||_2^2.\end{aligned}$$

puisque

$$||\hat{\beta}_s - \beta^*_s||_1 \leq \sqrt{s}||\hat{\beta}_s - \beta^*||_2 \leq \sqrt{s}||\hat{\beta} - \beta^*||_2$$

on a

$$\frac{\phi}{2}||\hat{\beta} - \beta^*||_2^2 \leq (\alpha + c)\lambda_n\sqrt{s}||\hat{\beta} - \beta^*||_2 + \lambda_n(1-\alpha)||\Delta||_1$$

$$\Longrightarrow ||\hat{\beta} - \beta^*||_2 \leq \frac{(\alpha + c)\lambda_n\sqrt{s} + \sqrt{(\alpha + c)^2\lambda_n^2 s + 2\phi\lambda_n(1-\alpha)||\Delta||_1}}{\phi}$$

$$\Longrightarrow ||\hat{\beta} - \beta^*||_2^2 \leq \left[\frac{(\alpha + c)\lambda_n\sqrt{s} + \sqrt{(\alpha + c)^2\lambda_n^2 s + 2\phi\lambda_n(1-\alpha)||\Delta||_1}}{\phi}\right]^2$$

$$\Longrightarrow ||\hat{\beta} - \beta^*||_2^2 \leq \frac{(\alpha + c)^2\lambda_n^2 s}{\phi^2}\left[1 + \sqrt{1 + \frac{2\phi(1-\alpha)||\Delta||_1}{(\alpha + c)^2\lambda_n s}}\right]^2$$

□

#### 3.8.3.2 Taux de convergence

**Théorème 3.8.5: Taux de convergence [53]**

On suppose l'hypothèse 1 est vérifiée et que la condition GRE 3.8.3.1 soit aussi vérifiée pour le même ensemble $B$. Alors, avec une probabilité au moins $1 - \nu_{n,c}$, on a :

$$||\hat{\beta} - \beta^*||_2^2 = O((\alpha + c)^2\lambda_n^2 s + (1-\alpha)\lambda_n||\Delta||_1)$$

lorsque $\lambda_n \to 0$.

*Démonstration.* d'après le théorème 3.8.3.1 on a :

$$||\hat{\beta} - \beta^*||_2^2 \leq \frac{(\alpha + c)^2 \lambda_n^2 s}{\phi^2} \left[1 + \sqrt{1 + \frac{2\phi(1-\alpha)||\Delta||_1}{(\alpha + c)^2 \lambda_n s}}\right]^2$$

puisque

$$\sqrt{1 + \frac{2\phi(1-\alpha)||\Delta||_1}{(\alpha + c)^2 \lambda_n s}} > 1$$

on a

$$\begin{aligned} ||\hat{\beta} - \beta^*||_2^2 &\leq \frac{(\alpha + c)^2 \lambda_n^2 s}{\phi^2} \left[2\sqrt{1 + \frac{2\phi(1-\alpha)||\Delta||_1}{(\alpha + c)^2 \lambda_n s}}\right]^2 \\ &= \frac{4(\alpha + c)^2 \lambda_n^2 s}{\phi^2} + \frac{8(1-\alpha)\phi||\Delta||_1}{\phi} \\ &= O((\alpha + c)^2 \lambda_n^2 s + (1-\alpha)\lambda_n ||\Delta||_1). \end{aligned}$$

□

**Théorème 3.8.6: [53]**

On suppose les meme condition que le théorème 3.8.3.1, on suppose de plus que la condition dite **beta-min**

$$|\beta_s^*| > \frac{(\alpha + c)^2 \lambda_n^2 s}{\phi^2} \left[1 + \sqrt{1 + \frac{2\phi(1-\alpha)||\Delta||_1}{(\alpha + c)^2 \lambda_n s}}\right]^2$$

est vérifie alors, on a

$$S \subset supp(\hat{\beta})$$

avec un probabilité au monis $1 - \nu_{n,c}$.

#### 3.8.3.3 unchanging condition

Le théorème suivant montre que l'estimation reste inchangée sous une certaines conditions.

**Théorème 3.8.7: unchanging condition [53]**

Soit $r(\beta) := y - X\beta$. il existe une solution inchangé $\hat{\beta} = \tilde{\beta}$ si, et seulement si :

$$\left|\frac{1}{n}X_j^t r(\tilde{\beta})\right| \leq \lambda \quad \forall j \quad s.t. \quad \tilde{\beta}_j = 0$$

et

$$-\lambda\Big((1-\alpha) - \alpha sgn(\tilde{\beta}_j)\Big) \leq \frac{1}{n}X_j^t r(\tilde{\beta}) \leq \lambda\Big((1-\alpha) + \alpha sgn(\tilde{\beta})\Big) \quad \forall j \quad \tilde{\beta}_j \neq 0$$

de plus, il existe une solution nulle $\hat{\beta} = 0$ si et seulement si :

$$\left|\frac{1}{n}X_j^t r(0)\right| \leq \lambda \quad \forall j \quad s.t. \quad \tilde{\beta}_j = 0$$

et

$$-\lambda\Big((1-\alpha) - \alpha sgn(\tilde{\beta}_j)\Big) \leq \frac{1}{n}X_j^t r(0) \leq \lambda\Big(\alpha - (1-\alpha) sgn(\tilde{\beta})\Big) \quad \forall j \quad \tilde{\beta}_j \neq 0$$

*Démonstration.* En utilisant le théorème 3.8.1.2, il existe une solution nulle $\hat{\beta} = 0$ si et seulement si

$$\left|\frac{1}{n}X_j^t y\right| \leq \lambda \quad \forall j \quad s.t. \quad \tilde{\beta}_j = 0$$

et

$$\left|\frac{1}{n}X_j^t y + \lambda(1-\alpha)sgn(\tilde{\beta}_j)\right| \leq \lambda\alpha \quad \forall j \quad s.t. \quad \tilde{\beta}_j \neq 0 \qquad (*)$$

$(*)$ est équivalent à

$$-\lambda\alpha - \lambda(1-\alpha)sgn(\tilde{\beta}_j) \leq \frac{1}{n}X_j^t y \leq \lambda\alpha - \lambda(1-\alpha)sgn(\tilde{\beta}_j) \quad \forall j \quad s.t. \quad \tilde{\beta}_j \neq 0$$

soit $r := y - X\tilde{\beta}$. par le théorème 3.8.1.2, il existe un solution inchangé $\hat{\beta} = \tilde{\beta}$ si et seulement si

$$\left|\frac{1}{n}X_j^t r\right| \leq \lambda \quad \forall j \quad s.t. \quad \tilde{\beta}_j = 0$$

et

$$\left|\frac{1}{n}X_j^t r - \lambda\alpha sgn(\tilde{\beta}_j)\right| \leq \lambda(1-\alpha) \quad pour \quad s.t. \quad \tilde{\beta}_j = 0 \qquad (**)$$

$(**)$ est équivalent à :

$$-\lambda(1-\alpha) + \lambda\alpha sgn(\tilde{\beta}_j) \leq \frac{1}{n}X_j^t r \leq \lambda(1-\alpha) + \lambda\alpha sgn(\tilde{\beta}_j) \quad \forall j \quad s.t. \quad \tilde{\beta}_j \neq 0$$

□

L'estimation initiale $\tilde{\beta}$ est arbitraire. Nous étudions le comportement de Transfert Lasso comme une estimation en deux étapes. Nous supposons que l'estimation initiale est une solution Lasso utilisant un autre jeu de données $X' \in \mathbb{R}^{m\times p}$, $Y' \in \mathbb{R}^m$ et le vrai paramètre $\tilde{\beta}^*$. on définie

$$S' = supp(\tilde{\beta}^*), s' = |S'|, \Delta' := \tilde{\beta}^* - \beta^*$$

Alors on a le corollaire suivante

**Corollaire 3.8.1: [53]**

On suppose que l'hypothèse 1 est vérifiée et que la condition GRE est vérifiée pour $\phi' = \phi'(B')$ et $B' = B'(1, c', 0)$ on $X', y$, et $\tilde{\beta}^*$. on suppose les même hypothèses que le théorème 3.8.3.1. Alors avec une probabilité au moins $1 - \nu_{n,c} - \nu_{m,c'}$

$$||\hat{\beta} - \beta^*||_2^2 \leq \frac{(\alpha + c)^2 \lambda_n^2 s}{\phi^2} \left(1 + \sqrt{1 + \frac{4(1-\alpha)(1+c')\phi\lambda_m s'}{(\alpha+c)^2\phi'\lambda_n s} + \frac{2(1-\alpha)\phi||\Delta^*||_1}{(\alpha+c)^2\lambda_n s}}\right)^2$$

*Démonstration.* Part le théorème 3.8.3.1 avec $\alpha = 1$, on a pour une certain $c' > 0$

$$||\tilde{\beta} - \tilde{\beta}^*||_2^2 \leq \frac{4(1+c')^2\lambda_m^2 s'}{\phi'^2}, \quad et \quad ||\tilde{\beta} - \tilde{\beta}^*||_1 \leq \frac{2(1+c')\lambda_m s'}{\phi'}$$

avec

$$\phi' = \phi'(B') := \inf_{v \in B'} \frac{v^t \frac{1}{n} X'^t X' v}{||v||_2^2} > 0$$

$$B' = B'(c') := \{v \in \mathbb{R}^p : (1-c')||v_{s'_c}||_1 \leq (1+c')||v_{s'}||_1\}$$

avec une probabilité au moins $1 - \nu_{m,c'}$, donc on a :

$$||\Delta||_1 = ||\tilde{\beta} - \beta^*||_1 \leq ||\tilde{\beta} - \tilde{\beta}^*||_1 + ||\tilde{\beta}^* - \beta^*||_1 \leq \frac{2(1+c')\lambda_m \tilde{s}}{\phi'} + ||\Delta^*||_1$$

avec $\Delta := \tilde{\beta}^* - \beta^*$. En combinant ça avec le théorème 3.8.3.1, on obtient le corollaire. □

Chapitre 4

# Application à la classification des images

## 4.1 Introduction

Les CNNs sont utilisées pour des problèmes de classification majeurs dans le cadre desquels les caractéristiques sont automatiquement acquises du niveau inférieur au niveau supérieur sur la croissance successive des couches du réseau. L'apprentissage par transfert est le processus qui consiste à transférer les connaissances apprises en résolvant un problème pour en résoudre un autre qui est connexe. L'apprentissage par transfert est bénéfique lorsqu'il n'y a pas suffisamment de données relatives à l'apprentissage et qu'il réduit également la complexité des calculs. Il fournit une bonne précision de classification sur une base de données réduite. Dans notre application, l'architecture VGG16 qui est pré-formée sur une énorme base de données ImageNet avec plus d'un million d'images appartenant à 1000 catégories différentes est utilisée pour classifier les images d'entrée. Ensuite, la classification se fait au moyen de la couche entièrement connectée et de l'activation Softmax.
Dans le présent chapitre, nous illustrerons l'importance de l'apprentissage par transfert dans les réseaux neuronaux convolutifs. Nous emploierons VGG16 pour la classification des images.
Notre application se décline en deux volets :

1. La première partie est consacrée à la comparaison entre le VGG16 brute, c'est-à-dire sans transfert, puis le VGG16 préformé c'est-à-dire en utilisant le transfert des connaissances, à l'aide des bases de données "CIFAR10", " MNIST " et la base "Flowers recognition".
2. Afin de mieux comprendre l'importance du transfert des connaissances, nous nous penchons sur la question des lacunes dans les données, Il s'agit du cas où il n'y a pas suffisamment de données pour construire un modèle performant à zéros. Nous appliquerons l'apprentissage de transfert pour dépister la "pneumonie", c'est-à-dire classer les personnes qui ont une pneumonie et celles qui n'en ont pas au moyen de la base de données Chest X_Ray. Celle-ci contient 5863 au total, ce qui ne suffit pas à construire un modèle à zéro.

## 4.2 Description des bases de données

### CIFAR10

L' ensemble de données CIFAR-10 ( Institut Canadien de recherches avancées ) est une collection d'images couramment utilisées pour former des algorithmes d'apprentissage

automatique et de vision par ordinateur. C'est l'un des ensembles de données les plus largement utilisés pour la recherche en apprentissage automatique. L'ensemble de données CIFAR-10 contient 60 000 images couleur $32 \times 32 \times 3$ dans 10 classes différentes répartie de la manière suivante.

1. 50 000 données d'apprentissage.
2. 10 000 données de tests

Les 10 classes différentes représentent les avions, les voitures, les oiseaux, les chats, les cerfs, les chiens, les grenouilles, les chevaux, les bateaux et les camions. Il y a 6000 images de chaque classe.

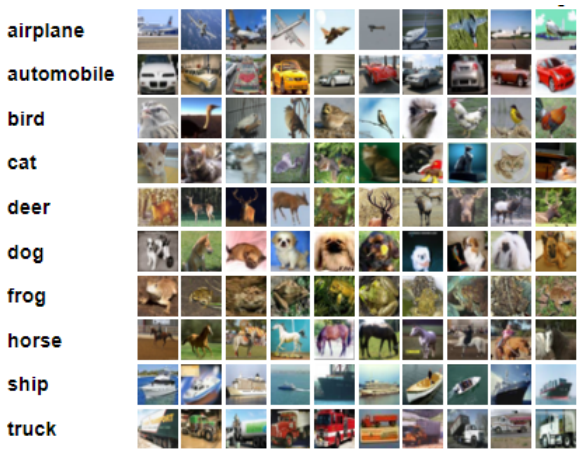



FIGURE 4.1 – Échantillon de la base CIFAR10

### chest X_Ray

Pour les problèmes de classification biomédicale de l'image, il est difficile d'obtenir une quantité aussi importante de données étiquetées, car il faut des médecins experts pour classer chaque image, ce qui est une tâche coûteuse et chronophage. L'apprentissage par transfert est un moyen détourné pour surmonter cet obstacle.Dans cette technique, pour résoudre un problème concernant un ensemble de données restreint ; un modèle fondé sur un ensemble de données volumineux est réutilisé et les poids du réseau déterminés dans ce modèle sont appliqués. Les Modèles CNNs formés sur un grand ensemble de données tel que ImageNet , qui inclut plus de 14 millions d'images, sont fréquemment utilisées pour la classification biomédicale des images.
Le jeu de données est organisé en 3 dossiers (train, test, val) et contient des sous-dossiers pour chaque catégorie d'image (Pneumonia/Normal). Il y a 5 863 images radiographiques (JPEG) et 2 catégories (Pneumonie/Normal).

## 4.3 Matrice de confusion

La matrice de confusion, également appelée matrice d'erreurs, est une disposition de tableau spécifique qui permet de visualiser les performances d'un algorithme. Chaque ligne de la matrice représente les instances d'une classe réelle tandis que chaque colonne représente les instances d'une classe prédite, ou vice versa.

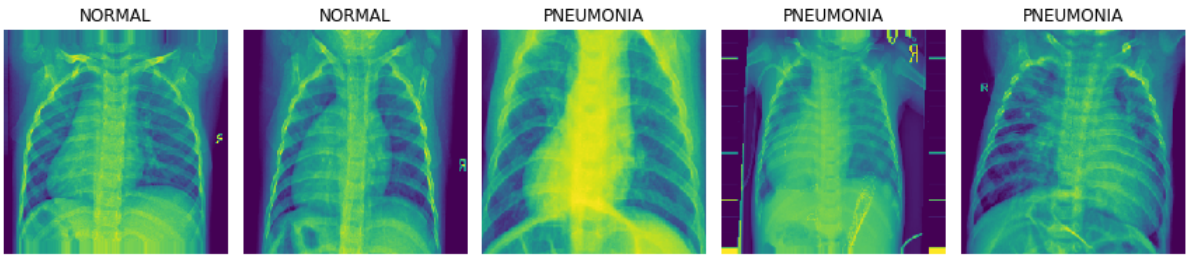



FIGURE 4.2 – Échantillon de la base Chest X_Ray

| | | État prévu | |
|---|---|---|---|
| | Population totale = P + N | Positif (PP) | Négatif (PN) |
| État réel | Positif (P) | Vrai positif (TP) | Faux négatif (FN) |
| | Négatif (N) | Faux positif (FP) | Vrai négatif (TN) |

FIGURE 4.3 – Matrice de confusion

Vrai positif (TP) : mesure dans quelle mesure le modèle prédit correctement la classe positive.
Faux positif (FP) : Des faux positifs se produisent lorsque le modèle prédit qu'une instance appartient à une classe qu'elle n'appartient pas en réalité.
Vrai négatif (TN) : Les vrais négatifs sont les résultats que le modèle prédit correctement comme étant négatifs.
Faux négatif (FN) : Un faux négatif se produit lorsqu'un modèle prédit qu'une instance est négative alors qu'elle est en réalité positive.

Les faux négatifs peuvent s'avérer très coûteux, particulièrement en médecine. Par exemple, si un test de dépistage du cancer prévoit qu'un patient n'a pas le cancer, il pourrait conduire à une progression de la maladie sans traitement. Les faux négatifs sont souvent plus graves que les faux positifs, de sorte qu'il est important d'en tenir compte dans l'évaluation du rendement d'un modèle de classification.

### 4.3.1 Exemple : Classification binaire

Étant donné un échantillon de 12 individus, 8 ayant reçu un diagnostic de cancer et 4 non cancéreux, où les individus atteints de cancer appartiennent à la classe 1 (positif) et les individus non cancéreux appartiennent à la classe 0 (négatif) :

| Numéro individuel | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Classement réel | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

Supposons que nous ayons un classificateur qui distingue d'une façon ou d'une autre les personnes atteintes ou non du cancer, nous pouvons prendre les 12 personnes et les soumettre à l'évaluation. Le classificateur procède alors à 9 prédictions précises et au taux 3 : Selon les prévisions, deux personnes atteintes de cancer seraient exemptes de cancer (échantillons 1 et 2) et une personne sans cancer serait atteinte de cancer (échantillon 9).

| Numéro individuel | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Classement réel | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Classement prévu | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Si l'on compare l'ensemble de classification actuel à l'ensemble de classification prévu, il existe quatre résultats différents qui pourraient donner lieu à une colonne particulière. Tout d'abord, si la classification effective est positive et que la classification prévue est positive (1.1), c'est ce qu'on appelle un vrai résultat positif puisque l'échantillon positif a été correctement identifié par le classificateur. Deuxièmement, si la classification effective est positive et la classification estimée négative (1,0), il s'agit d'un faux négatif parce que l'échantillon positif est identifié par erreur par le classificateur comme négatif. Troisièmement, si la classification effective est négative et que la classification anticipée est positive (0,1), il s'agit d'un faux positif parce que l'échantillon négatif est identifié à tort par le classificateur comme positif. Quatrièmement, lorsque la classification réelle est négative et que la classification prévue est négative (0,0), il s'agit d'un vrai résultat négatif, car l'échantillon négatif est correctement identifié par le classificateur comme négatif.

| Numéro individuel | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Classement réel | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Classement prévu | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| Résultat | FN | FN | TP | TP | TP | TP | TP | TP | FP | TN | TN | TN |

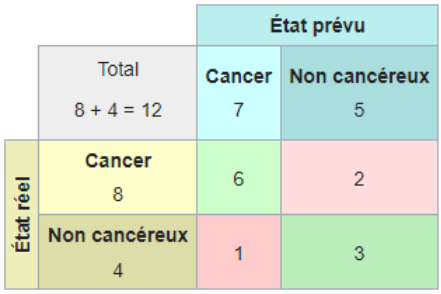



FIGURE 4.4 – Matrice de confusion pour la classification binaire

### 4.3.2 Exemple : Classification Multi-classes

Supposons qu' on a 3 classe :Versicolor, Virginia, Setosa de la base IRIS. Dans le problème de classification multi-classes, nous n'obtiendrons pas directement les valeurs TP, TN, FP, FN comme dans le problème de classification binaire. Nous devons calculer pour chaque classe :
FN : la valeur Faux-négatif pour une classe sera la somme des valeurs des lignes correspondantes à l'exception de la valeur TP.
FP : La valeur Faux-positif pour une classe sera la somme des valeurs de la colonne correspondante à l'exception de la valeur TP .
TN : La valeur True Négative pour une classe sera la somme des valeurs de toutes les colonnes et lignes à l'exception des valeurs de cette classe pour laquelle nous calculons les valeurs.
TP : La valeur vraie positive est celle où la valeur réelle et la valeur prédite sont identiques.
La matrice de confusion pour l'ensemble de données IRIS est la suivante : Calculons les valeurs TP, TN, FP, FN pour la classe Setosa en utilisant les astuces ci-dessus :

| | setosa | versicolor | virginica |
|---|---|---|---|
| setosa | 16 | 0 | 0 |
| versicolor | 0 | 17 | 1 |
| virginica | 0 | 0 | 11 |

TABLE 4.1 – Matrice de confusion pour la base IRIS

TP : La valeur réelle et la valeur prédite doivent être identiques. Donc concernant la classe Setosa, la valeur de la cellule 1 est la valeur TP.
FN : la somme des valeurs des lignes correspondantes à l'exception de la valeur TP,
FN = (cellule 2+cellule3) = $(0 + 0) = 0$
FP : La somme des valeurs de la colonne correspondante à l'exception de la valeur TP.
FP = (cellule 4+cellule 7) = $(0 + 0) = 0$
TN : la somme des valeurs de toutes les colonnes et lignes, à l'exception des valeurs de cette classe pour laquelle nous calculons les valeurs.
TN = (cellule 5 + cellule 6 + cellule 8 + cellule 9)= $17 + 1 + 0 + 11 = 29$
De même, pour la classe Versicolor , virginica.
Pourquoi la matrice de confusion ?
La matrice de confusion nous permet d'évaluer le rappel, la précision, l'exactitude et la courbe AUC-ROC sont les mesures de mesure de la performance d'un modèle de deep learning, et cela grâce aux paramètres que nous allons définir ci-après.

**QUESTION :Quelle mesure devrait être employée pour évaluer les performances d'un modèle d'apprentissage profond ?**

Il est important d'évaluer le rendement du modèle de classification afin de pouvoir utiliser ces modèles de manière fiable en production pour résoudre des problèmes concrets. Les paramètres de rendement dans les modèles de classification d'apprentissage automatique servent à évaluer le rendement des modèles de classification d'apprentissage automatique dans un contexte donné. Ces mesures de rendement comprennent l'exactitude, la précision, le rappel et le score F1, parce qu'ils nous aident à comprendre les points forts et les limites de ces modèles lors de l'établissement de prévisions dans de nouvelles situations. La performance d'un modèle est cruciale pour l'apprentissage automatique.

## 4.4 Métriques

### 4.4.1 Exactitude

L'exactitude (accuracy) du modèle est un indicateur de rendement du modèle d'apprentissage automatique, ce qui correspond au rapport entre les vrais positifs et les vrais négatifs et toutes les observations positives et négatives. Autrement dit, la précision indique la fréquence à laquelle il faut s'attendre à ce que notre modèle de machine learning prédit avec précision un résultat sur le nombre total de fois où il a fait des prévisions.

$$accuracy = \frac{(TP + TN)}{(TP + FN + TN + FP)}$$

pour les exemples donnés ci-dessus :

1. classification binaire : exactitude= $\frac{6+3}{6+2+1+3} = 75\%$

2. classification multi-classes : exactitude= $\frac{16+29}{16+29+0+0} = 100\%$ pour la classe setosa.

**Remarque 4.4.1: Importante**

Lorsque l'ensemble de données n'est pas équilibré, une autre mesure de l'AUC (la région sous la courbe ROC) est plus robuste que la mesure de précision. L'AUC tient compte de la distribution des classes dans un ensemble de données asymétriques.

### 4.4.2 Précision

Le score de précision du modèle mesure la proportion d'étiquettes positives qui sont en fait correctes. Autrement dit, pour tous les cas déclarés positifs, quel pourcentage était exact ? La précision est aussi reconnue comme valeur prédictive positive. La précision est principalement utilisée quand on a besoin de prédire la classe positive et que le coût des faux positifs est plus élevé que celui des faux négatifs comme dans le diagnostic médical. Le score de précision est une mesure utile de la réussite prédictive quand les classes sont fortement déséquilibrées. D'un point de vue mathématique, il représente la relation entre le vrai positif et la somme du vrai positif et du faux positif.

$$precision = \frac{TP}{(FP + TP)}$$

pour les exemples donnés ci-dessus :

1. classification binaire : précision= $\frac{6}{6+1} = 85.7\%$
2. classification multi-classes : précision= $\frac{16}{16+0} = 100\%$ pour la classe setosa.

### 4.4.3 Rappel

La score de rappel (Racall) du modèle représente la capacité du modèle à prédire avec précision les résultats positifs parmi les résultats positifs réels. Autrement dit, « dans tous les cas vraiment positifs, quel pourcentage a été classifié correctement ?». Cela diffère de la précision qui mesure le nombre de prédictions faites par les modèles qui sont réellement positives sur toutes les prédictions positives faites. En d'autres mots, il mesure la capacité de notre modèle d'apprentissage automatique d'identifier tous les vrais positifs parmi tous les positifs présents dans un ensemble de données. Plus le taux de rappel est élevé, plus le modèle de machine learning est efficace pour repérer les exemples positifs et négatifs. Le rappel est aussi désigné sous le nom de sensibilité ou de taux positifs réels. Un taux de mémorisation élevé indique que la tendance est bonne pour repérer les exemples positifs.

Le rappel est souvent utilisé conjointement avec d'autres mesures du rendement, comme l'exactitude et la précision, afin de brosser un tableau complet du rendement du modèle.

$$rappel = \frac{TP}{FN + TP}$$

pour les exemples donnés ci-dessus :

1. classification binaire :rappel= $\frac{6}{6+2} = 75\%$
2. classification multi-classes : rappel= $\frac{16}{16+0} = 100\%$ pour la classe setosa.

### 4.4.4 F1-score

Le score F1 du modèle représente la note du modèle en fonction de la note de précision et de mémorisation. Le score F1 est un moyen harmonique pondéré de précision et de mémoire de sorte que le meilleur score est 1 et le pire est 0. En général, le score F1 est inférieur aux mesures de précision car celles-ci incluent la précision et le rappel dans leur calcul. En règle générale, il convient d'utiliser la moyenne pondérée F1 pour comparer les modèles de classification, et non la précision globale. Le F1-score est une métrique de performance du modèle d'apprentissage automatique qui donne un poids égal à la fois à la précision et au rappel pour mesurer ses performances en termes de précision, il s'agit donc d'une solution de rechange aux mesures de précision (nous n'avons pas besoin de connaître le nombre total d'observations). Il est souvent utilisé en tant que valeur unique qui fournit des informations de haut niveau sur la qualité des résultats du modèle. Il s'agit d'une mesure utile du modèle dans les scénarios où des tentatives sont faites pour optimiser la précision ou la note de rappel, ce qui nuit au rendement du modèle.

$$\begin{aligned} F1 &= 2.\frac{prcision \times rappel}{prcision + rappel} \\ &= \frac{TP}{TP + \frac{1}{2}(FP + FN)} \end{aligned}$$

pour les exemples donnés ci-dessus :

1. classification binaire : F1-score= $2.\frac{0.85\times0.75}{0.85+0.75} = 75\%$
2. classification multi-classes : F1-score= $2.\frac{1\times1}{1+1} = 100\%$ pour la classe setosa.

**Remarque 4.4.2**

les différentes combinaisons de rappel et de précision ont les significations suivantes :

- haut rappel + haute précision : la classe est parfaitement gérée par le modèle.
- faible rappel + haute précision : le modèle ne peut pas bien détecter la classe mais est très fiable lorsqu'il le fait.
- rappel élevé + faible précision : la classe est bien détectée mais le modèle inclut également des points d'autres classes.
- faible rappel + faible précision : la classe est mal gérée par le modèle.

## 4.5 Modèle d'apprentissage par transfert : VGG16

VGG16 est une architecture de réseau neuronal à convolution (CNN) qui a été utilisée pour remporter le concours ILSVR (Imagenet) en 2014. Elle est considérée comme l'une des meilleures architectures de modèle de vision jusqu'à ce jour. La chose la plus unique à propos de VGG16 est qu'au lieu d'avoir un grand nombre d'hyper-paramètres, ils se sont concentrés sur des couches de convolution de filtre $3 \times 3$ avec une foulée 1 et ont toujours utilisé le même rembourrage et la même couche maxpool de filtre $2 \times 2$ de foulée 2. Il suit cet arrangement de couches de convolution et de Maxpoling de manière cohérente dans toute l'architecture. Au final, il a 2 FC (couches entièrement connectées) suivies d'un softmax pour la sortie. Le nombre 16 dans VGG16 fait référence à 16 couches qui ont des poids. Ce réseau est un réseau assez grand et il compte environ 138 millions (environ) de paramètres.

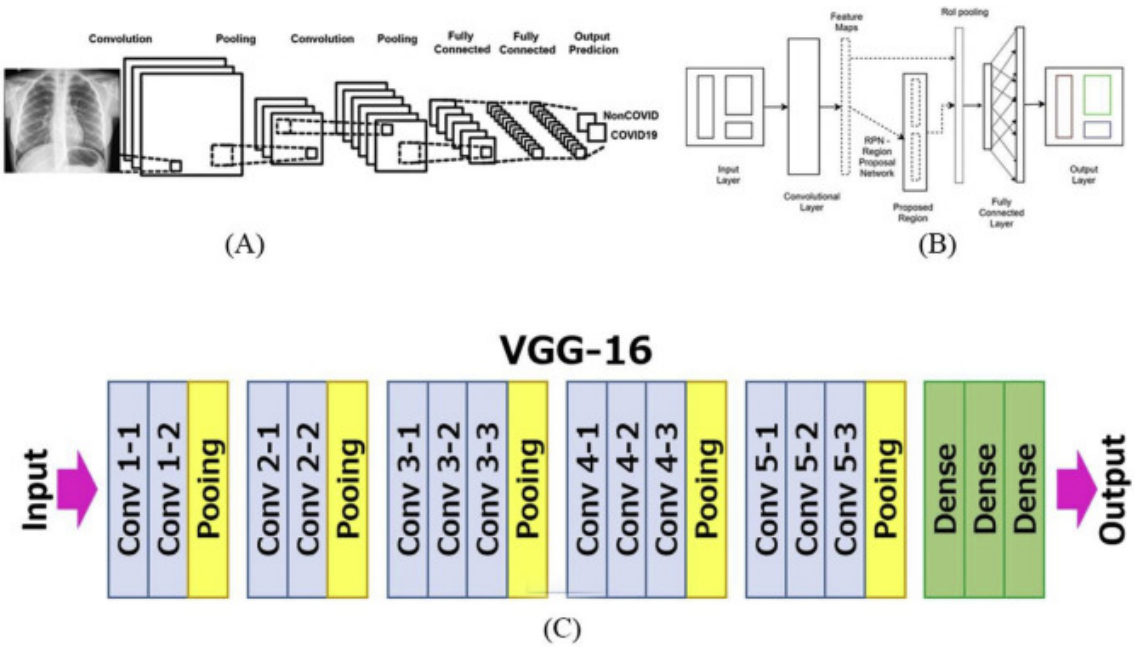



FIGURE 4.5 – Architecture de $VGG16$

## 4.6 Expérimentations et discutions

Avant la phase d'apprentissage toutes les images sont redimensionnés pour le modèle VGG16 (avec et sans transfert), En outre, tous les pixels de l'image sont normalisés dans $[0, 1]$. Nous avons formé les deux modèles Avec et Sans Transfert en utilisant les mêmes paramètres, ceux-ci sont :
le nombre d'époque est réglé à 100, l'entropie croisé catégorique sélectionnée comme fonction d'erreur, "Adam" utilisé comme algorithme d'optimisation, le taux d'apprentissage définie sur $3.22e-05$.
On a utilisé plusieurs méthodes de Régularisation pour éviter le sur-apprentissage, premièrement la normalisation par lot(Batch normalisation), deuxièmement la méthode Dropout après les couches entièrement connectée avec un taux de 0.2.

### 4.6.1 CIFAR10

Prédiction sur un échantillon de 10 000 Données, constitue par 1 000 pour chaque classe

| | paramètres entrainables | paramètres non entrainables | temps d'exécution |
|---|---|---|---|
| Sans Transfert | 15 111 242 | 0 | 1h 45min |
| Avec Transfert | 396 554 | 14 714 688 | 40 min |
| total de paramètres | 15 111 242 | 15 111 242 | |

TABLE 4.2 – Nombre de paramètres et temps d'exécution avec et sans transfert

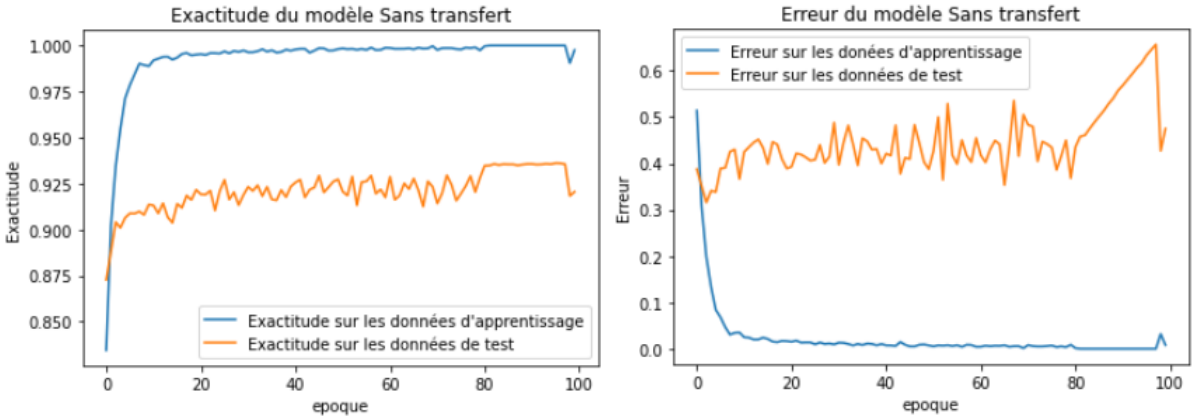



FIGURE 4.6 – Exactitude et erreur du modèle VGG16 sans transfert

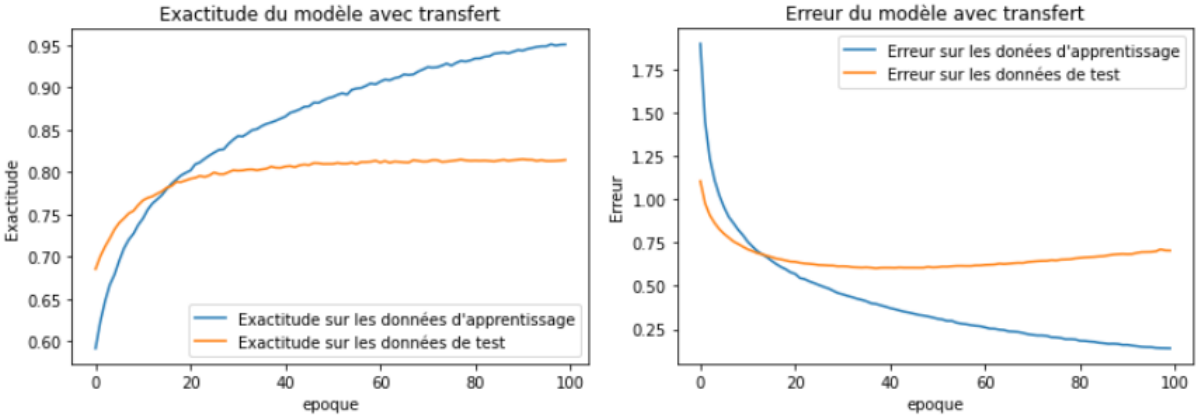



FIGURE 4.7 – Exactitude et erreur du modèle VGG16 avec transfert

| | Sans Transfert | | | Avec Transfert | | | |
|---|---|---|---|---|---|---|---|
| classes | précision | Rappel | F1-score | précision | Rappel | F1-score | support |
| avion | 0.93 | 0.94 | 0.93 | 0.84 | 0.83 | 0.84 | 1000 |
| automobile | 0.94 | 0.97 | 0.96 | 0.89 | 0.89 | 0.89 | 1000 |
| oiseau | 0.95 | 0.87 | 0.91 | 0.78 | 0.76 | 0.77 | 1000 |
| chat | 0.85 | 0.84 | 0.85 | 0.68 | 0.66 | 0.67 | 1000 |
| cerf | 0.93 | 0.90 | 0.92 | 0.77 | 0.76 | 0.76 | 1000 |
| chien | 0.90 | 0.86 | 0.88 | 0.78 | 0.74 | 0.76 | 1000 |
| grenouille | 0.97 | 0.91 | 0.94 | 0.81 | 0.84 | 0.82 | 1000 |
| cheval | 0.86 | 0.98 | 0.92 | 0.83 | 0.85 | 0.84 | 1000 |
| bateau | 0.94 | 0.98 | 0.96 | 0.87 | 0.91 | 0.89 | 1000 |
| camion | 0.96 | 0.95 | 0.95 | 0.89 | 0.89 | 0.89 | 1000 |
| exactitude | | | 0.92 | | | 0.81 | 1000 |
| Moyenne macro | 0.92 | 0.92 | 0.92 | 0.81 | 0.81 | 0.81 | 1000 |
| Moyenne pondérée | 0.92 | 0.92 | 0.92 | 0.81 | 0.81 | 0.81 | 1000 |

TABLE 4.3 – Rapport de classification avec et sans transfert

### 4.6.2 MNIST

Prédiction sur un échantillon de 10 000 Données.

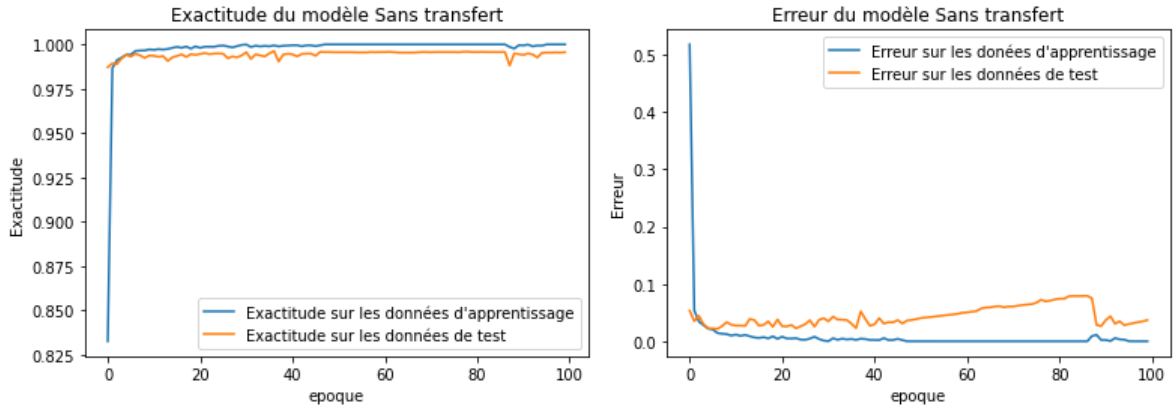



FIGURE 4.8 – Exactitude et erreur du modèle VGG16 sans transfert

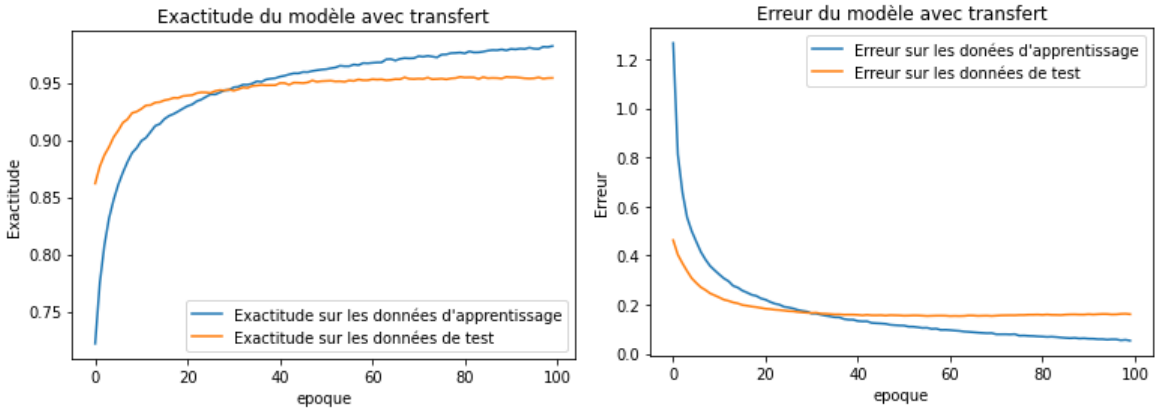



FIGURE 4.9 – Exactitude et erreur du modèle VGG16 avec transfert

| | paramètres entrainables | paramètres non entrainables | temps d'exécution |
|---|---|---|---|
| Sans Transfert | 15 111 242 | 0 | 1h 52min |
| Avec Transfert | 396 554 | 14 714 688 | 45 min |
| total de paramètres | 15 111 242 | 15 111 242 | |

TABLE 4.4 – Nombre de paramètres et temps d'exécution avec et sans transfert

| | Sans Transfert | | | Avec Transfert | | | |
|---|---|---|---|---|---|---|---|
| classes | précision | Rappel | F1-score | précision | Rappel | F1-score | support |
| 0 | 0.99 | 1.00 | 1.00 | 0.98 | 0.98 | 0.98 | 980 |
| 1 | 1.00 | 1.00 | 1.00 | 0.99 | 0.99 | 0.99 | 1135 |
| 2 | 1.00 | 1.00 | 1.00 | 0.95 | 0.93 | 0.94 | 1032 |
| 3 | 0.99 | 1.00 | 1.00 | 0.91 | 0.93 | 0.92 | 1010 |
| 4 | 0.99 | 1.00 | 1.00 | 0.97 | 0.97 | 0.97 | 982 |
| 5 | 1.00 | 0.99 | 0.99 | 0.94 | 0.93 | 0.93 | 892 |
| 6 | 1.00 | 0.99 | 1.00 | 0.98 | 0.97 | 0.97 | 958 |
| 7 | 0.99 | 1.00 | 0.99 | 0.96 | 0.97 | 0.96 | 1028 |
| 8 | 1.00 | 0.99 | 1.00 | 0.91 | 0.93 | 0.92 | 974 |
| 9 | 0.99 | 0.99 | 0.99 | 0.95 | 0.94 | 0.95 | 1009 |
| exactitude | | | 1.00 | | | 0.95 | 10000 |
| Moyenne macro | 1.00 | 1.00 | 1.00 | 0.95 | 0.95 | 0.95 | 10000 |
| Moyenne pondérée | 1.00 | 1.00 | 1.00 | 0.95 | 0.95 | 0.95 | 10000 |

TABLE 4.5 – Rapport de classification avec et sans transfert

### 4.6.3 Flowers Recognition

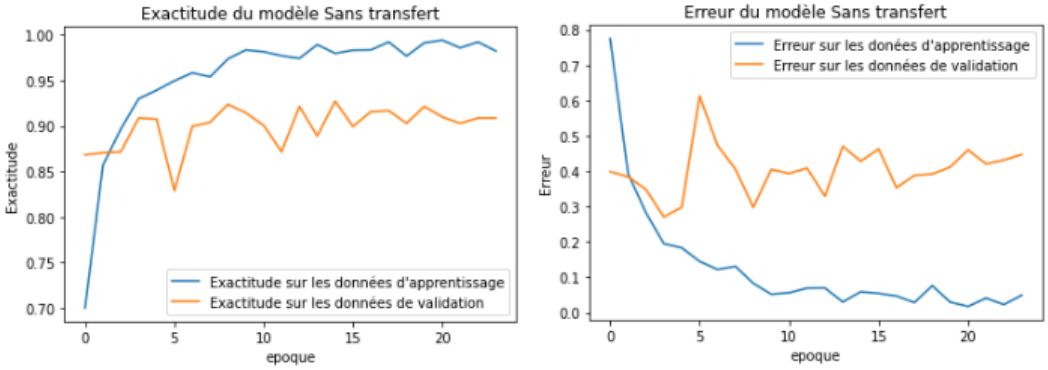



FIGURE 4.10 – Exactitude et erreur du modèle sans transfert

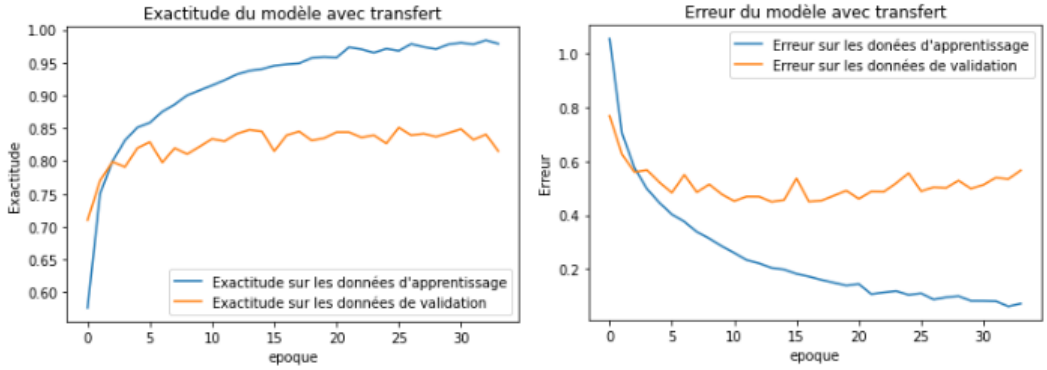



FIGURE 4.11 – Exactitude et erreur du modèle avec transfert

| | Sans Transfert | | | Avec Transfert | | | |
|---|---|---|---|---|---|---|---|
| classes | précision | Rappel | F1-score | précision | Rappel | F1-score | support |
| 0 | 0.93 | 0.90 | 0.85 | 0.78 | 0.81 | 153 | |
| 1 | 0.95 | 0.93 | 0.94 | 0.85 | 0.86 | 0.85 | 211 |
| 2 | 0.87 | 0.91 | 0.89 | 0.73 | 0.84 | 0.78 | 157 |
| 3 | 0.88 | 0.94 | 0.91 | 0.96 | 0.66 | 0.78 | 147 |
| 4 | 0.90 | 0.87 | 0.88 | 0.77 | 0.88 | 0.82 | 197 |
| exactitude | | | 0.91 | | | 0.82 | 865 |
| Moyenne macro | 0.91 | 0.91 | 0.91 | 0.83 | 0.81 | 0.81 | 865 |
| Moyenne pondérée | 0.91 | 0.91 | 0.91 | 0.83 | 0.82 | 0.81 | 865 |

TABLE 4.6 – Rapport de classification avec et sans transfert

| | paramètres entrainables | paramètres non entrainables | temps d'exécution |
|---|---|---|---|
| Sans Transfert | 33 073 477 | 0 | 40min/24 époque |
| Avec Transfert | 18 358 789 | 14 714 688 | 41 min / 35 époque |
| total de paramètres | 33 073 477 | 33 073 477 | |

TABLE 4.7 – Nombre de paramètres et temps d'exécution avec et sans transfert

Trois expériences ont été réalisées pour la comparaison entre l'apprentissage traditionnel et l'apprentissage par transfert, la première sur la base du CIFAR10, la deuxième sur le MNIST et la troisième sur la reconnaissance des fleurs. Dans les trois cas on voit que l'apprentissage par transfert diminué un peut l'exactitude, par exemple la première expérimentation sans transfert a obtenu une exactitude de 92% tab.4.6.1 sur les données de test cependant lorsqu'on a utilisé le transfert cette exactitude diminue à 81%, même chose pour la base MNIST, sans transfert a obtenu une exactitude de 100% tab.4.6.2 et avec transfert cette exactitude est diminuée à 95%. Les paramètres obtenus avec le transfert sont légèrement moins élevés que ceux sans transfert, mais deux choses très importantes ont été gagnées avec le transfert :

1. Complexité réduite dans le temps et la mémoire :les première et seconde expériences montrent que sans transfert, il y a plus de 15 millions de paramètres à régler tab.4.2,tab.4.4, on rappel que la taille de l'image est $32 \times 32 \times 3$,et le délai d'exécution est de 2 heures (avec GPU). Toutefois avec le transfert il y a seulement 396 554 paramètres à ajuster, et le temps d'exécution est de 40 min, dans le troisième essai avec le transfert il y a seulement 18 millions de paramètres pour se conformer tab.4.7. Il existe néanmoins plus de 33 millions de paramètres à moduler sans transfert, ce qui est donc très grand, l'apprentissage d'un tel réseau prend beaucoup de temps, même sur les appareils disposant de multiples unités de traitement graphique (GPU) et de multiples unités de traitement centrales (CPU).
2. Dans le cas du manque de données en tant qu'exemple l'émergence d'une nouvelle épidémie ou nous voulons construire un modèle qui prédit si quelqu'un est contaminé ou non. Dans ce cas, le transfert de connaissances est très utile, nous le verrons plus en détail dans la prochaine section avec la base de données sur le thorax X_Ray.

### 4.6.4 Chest X_Ray

L'un des principaux motifs du recours au transfert de connaissances réside dans le manque de données, nous sommes aux prises avec ce genre de problème, particulièrement en imagerie médicale. Pour ce faire, nous utiliserons la base de données Chest X_Ray qui ne contient pas suffisamment de données (5863) pour établir un réseau à zéro. Nous construirons un réseau de convolution pour la détection de la pneumonie, en utilisant le transfert de connaissances de VGG16 et la base Chest X_Ray, et nous comparerons notre résultat à d'autres résultats publiés.

#### 4.6.4.1 Modèle proposé

Nous utiliserons l'apprentissage par transfert au moyen du modèle préformé VGG16. Nous allons prendre le VGG16 et enlever le dernier niveau de classification ; puis nous ajouterons deux calques complètement connectés à l'activation de ReLu.
La première couche correspond à 128 neurones, suivie d'une couche Dropout.
La seconde couche consiste en 64 neurones suivis d'une couche de Dropout.
Une couche de classification finale constituée d'un seul neurone à activation sigmoïde.

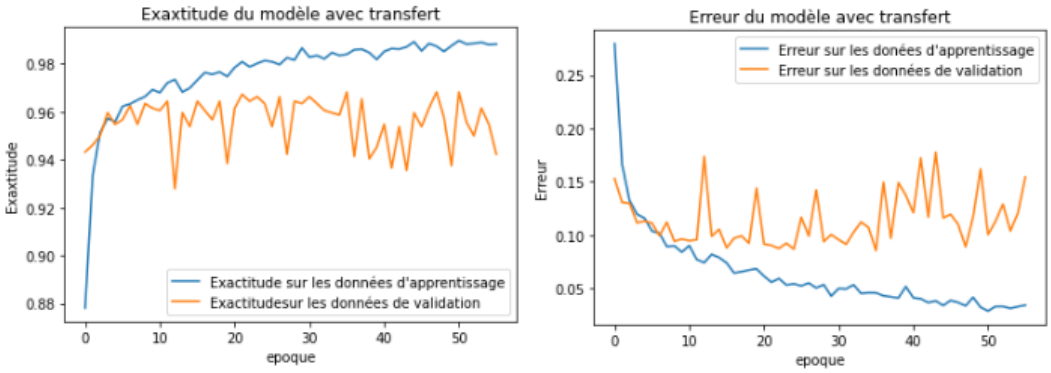



FIGURE 4.12 – Exactitude et erreur du modèle

Nous formons notre réseau au cours de 50 époque et nous obtenons l'exactitude et l'erreur sur les données d'apprentissage et les données de test. L'exactitude des données d'apprentissage a atteint $98,83\%$, tandis que celle des données de test a atteint $94,23\%$.

### 4.6.5 Résultats

Nous avons évalué notre modèle en utilisant 624 images radiographiques frontales du thorax. L'ensemble de tests contient 234 cas normaux et 390 cas de pneumonie. On trouvera au tableau 4.6.5 le rapport classification. La figure.4.6.5 montre la matrice de confusion de notre réseau.

La mesure du performance d'un modèle d'apprentissage en profondeur varie d'un modèle à l'autre. Dans notre cas et en général dans le domaine de l'imagerie médicale on veut que l'erreur " une personne infectée réellement soit prédite comme normale par le réseau" soit très faible et cela correspond au score de rappel. Pour minimiser cette erreur il faut avoir un score de rappel très proche de 1. D'après le tableau 4.6.5, ce qui est très bon parce-que même on n'a pas un nombreux suffisant de données pour construire un modèle à zéro. On a utilisé le transfert des connaissances en utilisant le VGG16, on a obtenu un

| | précision | rappel | F1-score | support |
|---|---|---|---|---|
| Personnes Normales [classe 0] | 0.93 | 0.85 | 0.89 | 234 |
| Personnes infectées [classe 1] | 0.92 | 0.96 | 0.94 | 390 |
| Exactitude | | | 0.92 | 624 |
| Moyenne macro | 0.93 | 0.91 | 0.92 | 624 |
| Moyenne pondérée | 0.92 | 0.92 | 0.92 | 624 |

TABLE 4.8 – Rapport de classification

modèle très performant, de plus on a gagné beaucoup de temps et de mémoire(réduire la complexité).

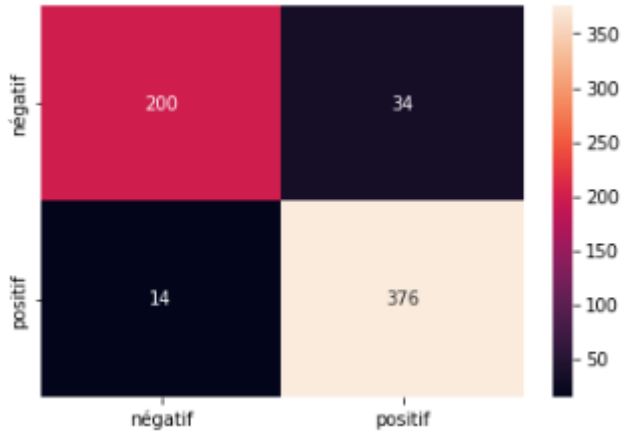



FIGURE 4.13 – Matrice de confusion

D'après la matrice de confusion 4.6.5, on voit que parmi 390 personnes infectées, notre réseau à prédire correctement 376 personnes, autrement dit un pourcentage de 96.4%, et parmi 234 personnes normales notre réseau a prédit correctement 200 personnes.

### 4.6.6 Comparaison avec des travaux antérieurs

| Travaux antérieurs | Exactitude | Rappel | F1-score |
|---|---|---|---|
| Enes Ayan et al .[3] | 87% | 89% | 90% |
| Rachna Jain et al .[28] | 87.18% | 96% | 90% |
| Gaobo Liang et al.[35] | 90.5% | 95.1% | 92.7% |
| Modèle proposé | 93% | 96% | 94% |

TABLE 4.9 – Comparaison avec des travaux antérieurs

D'après le tableau 4.6.5, on voit que l'exactitude de notre réseau est 92%, le score de Rappel est 96% . En comparant avec d'autres résultats de la littérature, Enes Ayan et al .[3] ils ont obtenue une exactitude de 87% et un score de Rappel 89%. Rachna Jain et al .[28] ont obtenue une exactitude de 87.17% et un score de Rappel 96% . Gaobo Liang et al.[35] ont obtenue une exactitude de 90.5% et un score de Rappel 95.1% . Cette différence de performance est une conséquence des couches qu'on a ajoutées au VGG16 préformé, et la régularisation qu'on a utilisée, Dropout pour les couches entièrement connectées,

la normalisation par lot pour les couches de convolution, ainsi que l'augmentation des données.

## 4.7 Conclusion

Dans le présent chapitre, une comparaison a été faite entre l'apprentissage classique et l'apprentissage par transfert ; la première partie utilise des bases de données standard et constate seulement que l'apprentissage par transfert réduit légèrement les performances, mais notre objectif d'exploité un modèle de l'apprentissage profond préformé et l'utilise dans un modèle cible, et on a remarqué que l'apprentissage par transfert réduit considérablement la complexité(spatiale et temporelle), de plus dans le cas du problème du manque des données qui est un problème très connue surtout dans le domaine de l'imagerie médicale, nous ne pouvons pas construire un modèle d'apprentissage profond à zéro parce qu'il engendrera le problème du sous-apprentissage (under-fitting). Dans ce cas, l'utilisation de l'apprentissage par transfert résout ce problème et donne de bons résultats de performance, dont il est question dans la deuxième partie de notre application.

# Conclusion générale

Tout au long de ce projet nous avons été amenés à concevoir et traité deux grands problèmes souvent rencontrés en deep learning qui sont :

1. Le sur-apprentissage(Overfitting)
2. Le sous-apprentissage(Underfitting) et plus précisément le problème de manque de données.

Initialement, nous étudions quelques algorithmes d'apprentissage et leurs bénéfices. Les réseaux de neurones convolutifs et leurs propres avantages dans le traitement d'images ont également fait l'objet de discussions. Nous avons ensuite vu des méthodes visant à limiter le problème de sur-apprentissage. Dans le troisième chapitre, le concept d'apprentissage par transfert a été introduit, ce qui résout essentiellement le problème du sous-apprentissage et plus spécifiquement le problème de l'absence de données. Enfin on a mené une expérience qui comporte deux volets :

1. Nous avons comparé l'apprentissage par transfert et l'apprentissage traditionnel à l'aide de trois bases de données.
2. On a utilisé la base de données radiographiques thoraciques contenant des images médicales afin de comprendre l'importance de l'apprentissage par transfert.

## .1 Complexité de Readmacher

pour démontrer le théorème 3.8.3.1, on aura besoins des résultat suivant :

**Lemme .1.1: [61]**

pour toutes les activations : sigmoïde, tanh et relu, on a :

$$\hat{R}_\ell(a \circ \mathcal{F}) \leq 2\hat{R}_\ell(\mathcal{F}) \tag{1}$$

**Lemme .1.2: [61]**

Soit $\hat{R}_\ell(\mathcal{G})$la classe des fonctions réelles $\mathbb{R}^d \to \mathbb{R}$ de dimension d'entrée $\mathcal{F}$, c'est-à-dire que $\mathcal{G} = \{\mathcal{F}\}_{j=1}^d$ et $\mathcal{H}_B$ est une fonction de transformation linéaire paramétré par $W$ avec $||W|| \leq B$ alors :

$$\hat{R}_\ell(\mathcal{H} \circ \mathcal{G}) \leq \sqrt{d}B\hat{R}_\ell(\mathcal{F} \tag{2}$$

*Démonstration.*

$$\begin{aligned}
\hat{R}_\ell(\mathcal{H} \circ \mathcal{G}) &= \mathbf{E}_\sigma \left[ \sup_{h\in\mathcal{H}, g\in\mathcal{G}} \left| \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i h \circ g(x_i) \right| \right] \\
&= \mathbf{E}_\sigma \left[ \sup_{h\in\mathcal{H}, g\in\mathcal{G}} \left| \left\langle W, \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i g(x_i) \right\rangle \right| \right] \\
&\leq B\mathbf{E}_\sigma \left[ \sup_{f^j\in\mathcal{F}} \left| \left| \left[ \frac{2}{\ell} \sum_{i=1}^{\ell} \sigma_i^j f^j(x_i) \right]_{j=1}^n \right| \right| \right] \\
&= B\sqrt{d}\mathbf{E}_\sigma \left[ \sup_{f\in\mathcal{F}} \left[ \frac{2}{\ell} \left| \sum_{i=1}^{\ell} \sigma_i f(x_i) \right]_{j=1}^n \right| \right] \\
&= \sqrt{d}B\hat{R}_\ell(\mathcal{F})
\end{aligned}$$

□

**Lemme .1.3: [61]**

soit $\mathcal{F}_M$ une classe de fonction dépend de $M$, Alors :

$$\hat{R}_\ell(\mathbf{E}_M[\mathcal{F}_M]) \leq \mathbf{E}_M[\hat{R}_\ell(\mathcal{F}_M)] \tag{3}$$

*Démonstration.* **( du théorème 3.8.3.1)**

$$\begin{aligned}
\hat{R}_\ell(\mathcal{F}) &= \hat{R}_\ell(\mathbf{E}_M\left[f(x,\theta,M)\right]) \\
&\leq \mathbf{E}_M(\hat{R}_\ell\left[f(x,\theta,M)\right]) \\
&= \mathbf{E}_M(\hat{R}_\ell\left[s \circ a \circ h_M \circ g\right]) \\
&\leq (\sqrt{dk}B_s)\sqrt{d}\mathbf{E}_M(\hat{R}_\ell\left[a \circ h_M \circ g\right]) \\
&= 2(\sqrt{k}dB_s)\mathbf{E}_M(\hat{R}_\ell\left[h_M \circ g\right])
\end{aligned}$$

avec :

$$h_M = (M \star W)v$$

on a :

$$\begin{aligned}
\mathbf{E}_M(\hat{R}_\ell\left[h_M \circ g\right]) &= 2(\sqrt{k}dB_s)\mathbf{E}_{M,\sigma}\left[\sup_{h\in\mathcal{H},g\in\mathcal{G}}\left|\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i W^T D_M g(x_i)\right|\right] \\
&= 2(\sqrt{k}dB_s)\mathbf{E}_{M,\sigma}\left[\sup_{h\in\mathcal{H},g\in\mathcal{G}}\left|\left\langle D_M W, \frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i g(x_i)\right\rangle\right|\right] \\
&\leq 2(\sqrt{k}dB_s)\mathbf{E}_M\left[\max_W ||D_M W||\right]\mathbf{E}_\sigma\left[\sup_{g^j\in\mathcal{G}}\left|\left|\frac{2}{\ell}\sum_{i=1}^{\ell}\sigma_i g^j(x_i)\right|\right|_{j=1}^{n}\right] \\
&\leq B_h p\sqrt{nd}(\sqrt{n}\hat{R}_\ell(\mathcal{G})) \\
&= pn\sqrt{d}B_h\hat{R}_\ell(\mathcal{G})
\end{aligned}$$

où $D_M$ dans l'équation est une matrice diagonale avec des éléments diagonaux égaux à $m$. d'ou

$$\hat{R}_\ell(\mathcal{F}) \leq p(2\sqrt{k}B_s n\sqrt{d}B_h)\hat{R}_\ell(\mathcal{G})$$

□

## .2 Apprentissage et Inférence avec les réseaux normalisés par lots

Pour normaliser par lots un réseau, nous spécifions un sous-ensemble d'activations et insérons la transformée $BN$ pour chacune d'elles selon Algorithme4. Toute couche qui recevait auparavant $x$ comme entrée reçoit maintenant $BN(x)$. Un modèle utilisant la normalisation par lots peut être formé à l'aide d'une descente de gradient par lots ou d'une descente de gradient stochastique avec un mini-lot de taille $m > 1$ ou avec l'une de ses variantes (Adagrad,Adam...). La normalisation des activations qui dépend du mini-batch permet un apprentissage efficace, mais n'est ni nécessaire ni souhaitable lors de l'inférence, nous voulons que la sortie dépende uniquement de l'entrée, de manière déterministe. Pour cela, une fois le réseau entraîné, on utilise la normalisation :

$$\hat{x} = \frac{x - \mathbf{E}(x)}{\sqrt{Var(x) + \epsilon}} \tag{4}$$

en utilisant la population plutôt que des mini-lots des statistiques. En négligeant $\epsilon$, ces activations normalisées ont la même moyenne 0 et la même variance 1 que pendant l'entraînement. Nous utilisons l'estimation de variance non biaisée $Var(x) = \frac{m}{m-1}\cdot\mathbf{E}_{\mathfrak{B}}(\sigma^2_{\mathfrak{B}})$, où l'espérance est sur la formation de mini-lots de taille $m$ et $\sigma^2_{\mathfrak{B}}$ sont leurs variances d'échantillon. En utilisant plutôt des moyennes mobiles, nous pouvons suivre la précision d'un modèle pendant qu'il s'entraîne. Étant donné que les moyennes et les variances sont fixes lors de l'inférence, la normalisation est simplement une transformée linéaire appliquée à chaque activation. Il peut en outre être composé avec la mise à l'échelle par $\gamma$ et décalage de $\beta$, pour donner une seule transformée linéaire qui remplace $BN(x)$. L'algorithme5 résume la procédure d'apprentissage des réseaux normalisés par lots.

**Algorithm 5** Apprentissage d'un réseau normalisé par lots

**Entrées:** . un Réseau $\mathcal{N}$ avec paramètres entraînables $\theta$ ;
sous-ensemble d'activations $\{x_k\}_{k=1}^{K}$

**Sorties:** . Réseau normalisé par lots pour l'inférence, $\mathcal{N}_{BN}^{inf}$.

1 : $\mathcal{N}_{BN}^{tr} \longleftarrow \mathcal{N}$ // Apprentissage du réseau $BN$

**pour** $k = 1...K$ **faire**

2 : Ajouter la transformation $y_k = BN_{\gamma_k,\beta_k}(x_k)$ à $\mathcal{N}_{BN}^{tr}$

3 : Modifier chaque couche dans $\mathcal{N}_{BN}^{tr}$ avec l'entrée $x_k$ pour prendre $y_k$.

**fin pour**

4 : Apprentissage de $\mathcal{N}_{BN}^{tr}$ pour optimiser les paramètres $\theta \bigcup \{\gamma_k, \beta_k\}_{k=1}^{K}$

5 : $\mathcal{N}_{BN}^{inf} \longleftarrow \mathcal{N}_{BN}^{tr}$

**pour** k=1...K **faire**

6 : Traiter plusieurs mini-lots d'apprentissage $\mathfrak{B}$, chacun taille $m$ :

$\mathbf{E}(x) \longleftarrow \mathbf{E}_{\mathfrak{B}}(\mu_{\mathfrak{B}})$

$Var(x) \longleftarrow \frac{m}{m-1}\mathbf{E}_{\mathfrak{B}}(\sigma_{\mathfrak{B}}^2)$

9 :dans $\mathcal{N}_{BN}^{inf}$ remplacer la transformation $y = BN_{\gamma,\beta}(x)$ par :

$y = \frac{\gamma}{\sqrt{Var(x)+\epsilon}} \cdot x + (\beta - \frac{\gamma \mathbf{E}(x)}{\sqrt{Var(x)+\epsilon}})$

**fin pour**

# Bibliographie


[1] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. *Advances in neural information processing systems*, 19, 2006.

[2] Andrew Arnold, Ramesh Nallapati, and William W Cohen. A comparative study of methods for transductive transfer learning. In *Seventh IEEE international conference on data mining workshops (ICDMW 2007)*, pages 77–82. IEEE, 2007.

[3] Enes Ayan and Halil Murat Ünver. Diagnosis of pneumonia from chest x-ray images using deep learning. In *2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT)*, pages 1–5. Ieee, 2019.

[4] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1) :151–175, 2010.

[5] Arthur Benton. Facial recognition 1990. *Cortex*, 26(4) :491–499, 1990.

[6] Himanshu Sharad Bhatt, Shourya Roy, Arun Rajkumar, and Sriranjani Ramakrishnan. Learning transferable feature representations using neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4124–4134, 2019.

[7] Peter J Bickel, Ya'acov Ritov, and Alexandre B Tsybakov. Simultaneous analysis of lasso and dantzig selector. *The Annals of statistics*, 37(4) :1705–1732, 2009.

[8] Mario Bkassiny, Yang Li, and Sudharman K Jayaweera. A survey on machine-learning techniques in cognitive radios. *IEEE Communications Surveys & Tutorials*, 15(3) :1136–1159, 2012.

[9] Sylvie Boldo, François Clément, Vincent Martin, Micaela Mayero, and Houda Mouhcine. Lebesgue induction and tonelli's theorem in coq. *arXiv preprint arXiv :2202.05040*, 2022.

[10] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

[11] Benjamin M Case, Colin Gallagher, and Shuhong Gao. A note on sub-gaussian random variables. *Cryptology ePrint Archive*, 2019.

[12] Jonghyun Choi, Abhishek Sharma, David W Jacobs, and Larry S Davis. Data insufficiency in sketch versus photo face recognition. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2012.

[13] Tyler Cody and Peter Beling. A systems theory of transfer learning. 07 2021.

[14] Corinna Cortes, Marius Kloft, and Mehryar Mohri. Learning kernels using local rademacher complexity. *Advances in neural information processing systems*, 26, 2013.

[15] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. volume 227, pages 193–200, 01 2007.

[16] Gwendoline De Bie, Gabriel Peyré, and Marco Cuturi. Stochastic deep networks. In *International Conference on Machine Learning*, pages 1556–1565. PMLR, 2019.

[17] Stephanie Dick. Artificial intelligence. 2019.

[18] Ahmed Fawzy Gad, Ahmed Fawzy Gad, and Suresh John. *Practical computer vision applications using deep learning with CNNs*. Springer, 2018.

[19] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77 :354–377, 2018.

[20] Kevin Gurney. *An introduction to neural networks*. CRC press, 2018.

[21] Misgina Tsighe Hagos and Shri Kant. Transfer learning based detection of diabetic retinopathy from small dataset. *arXiv preprint arXiv :1905.07203*, 2019.

[22] William Grant Hatcher and Wei Yu. A survey of deep learning : Platforms, applications and emerging research trends. *IEEE Access*, 6 :24411–24432, 2018.

[23] David P Helmbold and Philip M Long. Surprising properties of dropout in deep networks. In *Conference on Learning Theory*, pages 1123–1146. PMLR, 2017.

[24] El Houssaine Hssayni, Nour-Eddine Joudar, and Mohamed Ettaouil. Convolutional neural networks : Architecture optimization and regularization. In *International Conference on Digital Technologies and Applications*, pages 180–189. Springer, 2022.

[25] Mohammed Amine Janati Idrissi, Hassan Ramchoun, Youssef Ghanou, and Mohamed Ettaouil. Genetic algorithm for neural network architecture optimization. In *2016 3rd International conference on logistics operations management (GOL)*, pages 1–4. IEEE, 2016.

[26] Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[27] Anil K Jain, Jianchang Mao, and K Moidin Mohiuddin. Artificial neural networks : A tutorial. *Computer*, 29(3) :31–44, 1996.

[28] Rachna Jain, Preeti Nagrath, Gaurav Kataria, V Sirish Kaushik, and D Jude Hemanth. Pneumonia detection in chest x-ray images using convolutional neural networks and transfer learning. *Measurement*, 165 :108046, 2020.

[29] Nour-Eddine Joudar, Mohamed Ettaouil, et al. Krr-cnn : kernels redundancy reduction in convolutional neural networks. *Neural Computing and Applications*, 34(3) :2443–2454, 2022.

[30] Ben Krose and Patrick van der Smagt. *An introduction to neural networks*. 2011.

[31] Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, 22(2) :712–733, 2020.

[32] Hang-Chin Lai and Lai-Jiu Lin. Moreau-rockafellar type theorem for convex set functions. *Journal of mathematical analysis and applications*, 132(2) :558–571, 1988.

[33] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553) :436–444, 2015.

[34] Gen Li, Yuantao Gu, and Jie Ding. L1 regularization in two-layer neural networks. *IEEE Signal Processing Letters*, PP :1–1, 11 2021.

[35] Gaobo Liang and Lixin Zheng. A transfer learning method with deep residual network for pediatric pneumonia diagnosis. *Computer methods and programs in biomedicine*, 187 :104964, 2020.

[36] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42 :60–88, 2017.

[37] Jie Lu, Vahid Behbood, Peng Hao, Hua Zuo, Shan Xue, and Guangquan Zhang. Transfer learning using computational intelligence : A survey. *Knowledge-Based Systems*, 80 :14–23, 2015.

[38] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization. *arXiv preprint arXiv :1809.00846*, 2018.

[39] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Understanding regularization in batch normalization. 09 2018.

[40] Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[41] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10) :1345–1359, 2009.

[42] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and Sundaraja S Iyengar. A survey on deep learning : Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5) :1–36, 2018.

[43] R Meena Prakash, N Thenmoezhi, and M Gayathri. Face recognition with convolutional neural network and transfer learning. In *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 861–864. IEEE, 2019.

[44] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning : transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766, 2007.

[45] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. Multilayer perceptron : Architecture optimization and training. 2016.

[46] Morten Arendt Rasmussen and Rasmus Bro. A tutorial on the lasso approach to sparse modeling. *Chemometrics and Intelligent Laboratory Systems*, 119 :21–31, 2012.

[47] Ricardo Ribani and Mauricio Marengoni. A survey of transfer learning for convolutional neural networks. In *2019 32nd SIBGRAPI conference on graphics, patterns and images tutorials (SIBGRAPI-T)*, pages 47–57. IEEE, 2019.

[48] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv :1609.04747*, 2016.

[49] Linda G Shapiro, George C Stockman, et al. *Computer vision*, volume 3. Prentice Hall New Jersey, 2001.

[50] Piyush Sharma, Akiko Ueno, and Russel Kingshott. Self-service technology in supermarkets–do frontline staff still matter ? *Journal of Retailing and Consumer Services*, 59 :102356, 2021.

[51] Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19 :221, 2017.

[52] Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*, volume 3. Springer Science & Business Media, 2012.

[53] Masaaki Takada and Hironori Fujisawa. Transfer learning via $\ell_1$ regularization. 06 2020.

[54] Srikanth Tammina. Transfer learning using vgg-16 with deep convolutional neural network for classifying images. *International Journal of Scientific and Research Publications (IJSRP)*, 9(10) :143–150, 2019.

[55] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[56] Rob Tibshirani, T Hastie, and M Wainwright. Statistical learning and sparsity, 2019.

[57] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society : Series B (Methodological)*, 58(1) :267–288, 1996.

[58] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends : algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[59] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3) :475–494, 2001.

[60] Dimpy Varshni, Kartik Thakral, Lucky Agarwal, Rahul Nijhawan, and Ankush Mittal. Pneumonia detection using cnn based feature extraction. In *2019 IEEE international conference on electrical, computer and communication technologies (ICECCT)*, pages 1–7. IEEE, 2019.

[61] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[62] Donghui Wang, Yanan Li, Yuetan Lin, and Yueting Zhuang. Relational knowledge transfer for zero-shot learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[63] H Wang, Z Lei, X Zhang, B Zhou, and J Peng. Machine learning basics. *Deep learning*, pages 98–164, 2016.

[64] Eric W Weisstein. Convolution. *https ://mathworld. wolfram. com/*, 2003.

[65] Xuetong Wu, Jonathan H Manton, Uwe Aickelin, and Jingge Zhu. A bayesian approach to (online) transfer learning : Theory and algorithms. *arXiv preprint arXiv :2109.01377*, 2021.

[66] LI Xuhong, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *International Conference on Machine Learning*, pages 2825–2834. PMLR, 2018.

[67] Yuki Yoshida and Masato Okada. Data-dependence of plateau phenomenon in learning with neural network—statistical mechanical analysis. *Advances in Neural Information Processing Systems*, 32, 2019.

[68] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. A survey on deep learning for big data. *Information Fusion*, 42 :146–157, 2018.

[69] Xiaojin Jerry Zhu. Semi-supervised learning literature survey. 2005.

[70] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1) :43–76, 2020.