Les réseaux de neurones

Chapitre II : Les réseaux de neurones

II.1 Introduction

Les réseaux de neurones sont des structures cellulaires artificielles et constituent une approche permettant d'aborder sous des angles nouveaux les problèmes de perception, de mémoire, d'apprentissage et de raisonnement (en d'autres termes... d'intelligence artificielle ou abrégée "I.A.") au même titre que les algorithmes génétiques .Ils s'avèrent aussi des alternatives très prometteuses pour contourner certaines des limitations des ordinateurs classiques. Grace à leur traitement parallèle de l'information et à leurs mécanismes inspirés des cellules nerveuses (neurones), ils infèrent des Propriétés émergentes permettant de solutionner des problèmes jadis qualifiés de complexes. **[10]**

De nombreux termes sont utilisés pour désigner le domaine des réseaux de neurones artificiels, comme connexionnisme ou neuromimétique**.** Connexionnisme et neuromimétique sont tous deux des domaines de recherche à part entière, qui manipulent chacun des modèles de réseaux de neurones artificiels, mais avec des objectifs différents **[11]**

II.2 Le neurone formel

Un neurone formel est un modèle mathématique. Il est utilisé dans le cadre des recherches sur l'intelligence artificielle, principalement en association avec d'autres neurones formels pour former un réseau de neurones. **[12]**

Il contient plusieurs formules mathématiques afin de reproduire le plus fidèlement possible le fonctionnement d'un neurone avec ses différentes entrées (dendrites), leurs importances (coefficient de pondération) et sa sortie (axone) et ainsi comprendre son potentiel d'interaction avec les autres neurones.

Il est caractérise par :

- Des entrées ei et leur poids synaptique wi
- Une fonction d'activation (de transfert) f
- Une sortie y

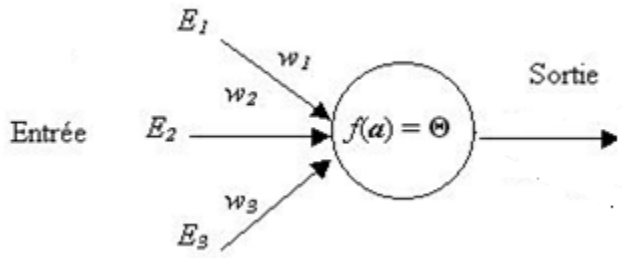


Figure II.1 : Neurone formel

- Les entrées *aj* du neurone i proviennent soit d'autres neurones soit de l'environnement.
- Les poids *Wji* associes a chaque neurone *j* déterminent l'influence de chaque entrée *i*.
- Le seuil *S* permet de contrôler l'entrée de la fonction d'activation f du neurone.
- Le potentiel d'activation est exprimé sous forme d'une combinaison linéaire des entrées *aj* pondérées par les coefficients *Wji*.
- La fonction d'activation f reçoit en entrée la somme pondérée pour produire la valeur de sortie du neurone qui sera transmise aux neurones suivants. Ainsi, à chaque signal entrant est associe un poids car les synapses n'ont pas toutes la même valeur. La valeur de la sommation est comparée à un seuil et la sortie du neurone est une fonction non linéaire du résultat:

$$in_i = \sum_{j=1}^{n} w_{ji} a_j - \theta \qquad (1)$$

$$a_i = f(in_i) \qquad (2)$$

Plusieurs fonctions d'activation f peuvent être utilisées. Toutefois dans la pratique, les deux fonctions de transfert les plus utilisées sont la fonction de Heaviside et la fonction sigmoïde.

- fonction de Heaviside: $\forall x \in R, f(x) = \begin{cases} 1 \; si \; x \geq 0 \\ 0 \quad sinon \end{cases}$ (3)
- tangente hyperbolique : $\forall x \in R, f(x) = \frac{2}{1+e^{-2x}} - 1$ (4)
- fonction gaussienne : $\forall x \in R, f(x) = e^{-\frac{x^2}{2}}$ (5)
- fonction sigmoïde : $\forall x \in R, f(x) = \frac{1}{1+e^{-x}}$ [13] (6)

II.2.1 Perceptron

Le perceptron a été introduit en 1958 par Franck Rosenblatt. Il s'agit d'un neurone artificiel inspiré par la théorie cognitive de Friedrich Hayek et celle de Donald Hebb. Dans sa version la plus simple, le perceptron n'a qu'une seule sortie y à laquelle toutes les entrées xi sont connectées (voir Figure II.2), ses entrées et sorties étant booléennes. [**14]**

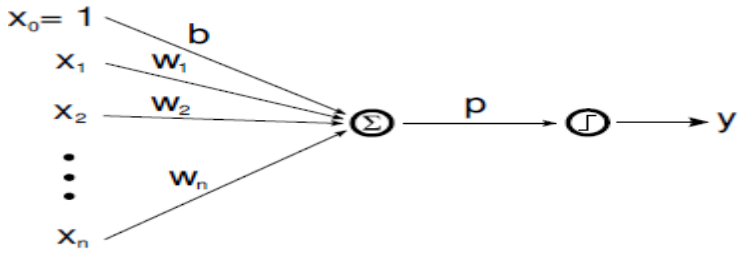


Figure II.2 : Modèle du perceptron

La somme pondérée des entrées par les poids W_i associés aux entrées est appelée potentiel, noté p.

$$p = \sum W_i X_i \qquad (7)$$

Ce potentiel est alors soumis à une fonction seuil de type Heaviside :

$$y = \begin{cases} 0 \; si \; s < 0 \\ 1 \; si \; s \geq 0 \end{cases} \qquad (8)$$

Cependant Minsky et Papert relèvent que le perceptron échoue pour des problèmes de classification simples, comme ceux où les classes ne sont pas Linéairement séparables. Le cas est la classification du XOR, où les motifs (0 ; 0) Et (1 ; 1) appartiennent à une classe tandis que les motifs (1 ; 0) et (0 ; 1) appartiennent à une autre classe. Ces problèmes suggèrent l'utilisation de plusieurs perceptrons, qui organisés en couches forment le modèle du perceptron multicouches. Ce modèle est capable de traiter des problèmes non linéaires.

II.2.2 Perceptron multi-couches

Dans le modèle du Perceptron Multicouches, les perceptrons sont organisés en couches. Les perceptrons multicouches sont capables de traiter des données qui ne sont pas linéairement séparables. Avec l'arrivée des algorithmes de rétro propagation, ils deviennent le type de réseaux

de neurones le plus utilisé. Les MLP sont généralement organisés en trois couches, la couche d'entrée, la couche intermédiaire (dite couche cachée) et la couche de sortie. L'utilité de plusieurs couches cachées n'a pas été démontrée. (La figure II.3) illustre la structure d'un MLP présentant quatre neurones en entrée, trois neurones sur la couche cachée et deux en sortie.
Lorsque tous les neurones d'une couche sont connectés aux neurones de la couche suivante, on parle alors de couches complètement connectées.**[14]**

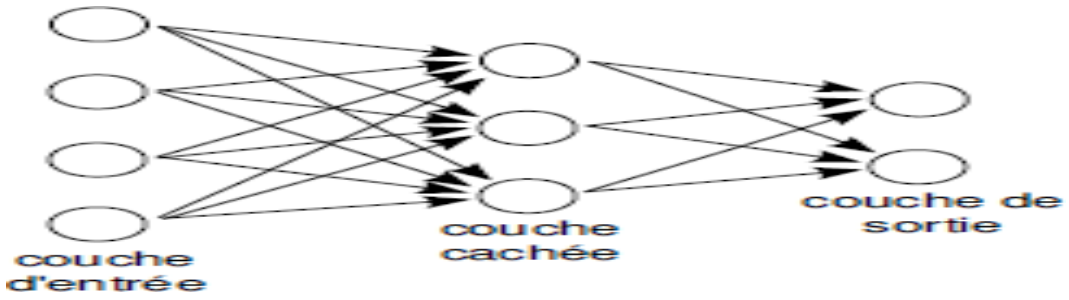


Figure II.3 Perceptron multi-couches

Les équations de Propagation décrites plus haut s'appliquent à tous les neurones. Cependant, le passage d'une couche à l'autre peut être formalisé sous forme matricielle. Soit un MLP dont le nombre de neurones sur la couche d'entrée est n0, n1 sur la couche cachée et n2 sur la couche de sortie. Les poids de la couche cachée peuvent s'écrire sous la forme d'une matrice $W^{Rn1*n0.}$ Pour un vecteur X^{Rn0} présenté en entrée, le vecteur potentiel V et le vecteur de sortie Y pour la couche cachée s'écrivent donc.

$$V = WX \qquad (9)$$

$$Y = \Phi(V) \qquad (10)$$

Avec Φ une fonction d'activation. La sortie de la couche cachée devient ensuite l'entrée de la dernière couche.

II.3 Rétro propagation du gradient (RPG)

La rétro propagation du gradient s'applique de manière récursive de la couche de sortie à la couche d'entrée tout en minimisant l'erreur quadratique $\frac{1}{2}(\mathrm{Y} - D)^2$. Elle consiste à faire évoluer les poids W dans la direction inverse du gradient $w = w - \boldsymbol{\lambda}\, dw$ où $\boldsymbol{\lambda}$ est le taux d'apprentissage.

Pour une couche dont X est le vecteur d'entrée, V le potentiel, W la matrice de poids et Y la sortie, la rétro propagation s'écrit donc :

$$\frac{\partial L}{\partial V} = \frac{\partial L}{\partial Y}\frac{\partial Y}{\partial V} \qquad (11)$$

$$= \Phi'(V)\frac{\partial L}{\partial Y}$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial V}\frac{\partial V}{\partial X} \qquad (12)$$

$$= W^T\frac{\partial L}{\partial V}$$

II.4 La Fonction d'activation

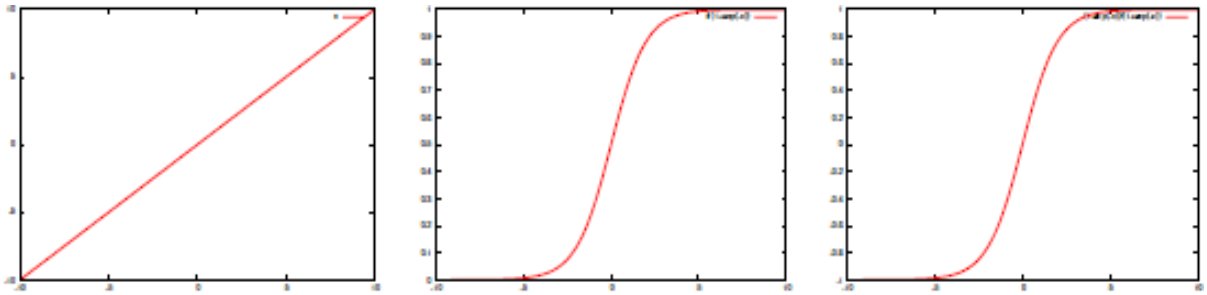
La fonction d'activation est la fonction mathématique qui permet de traiter l'information qui arrive à un neurone artificiel en machine Learning, comme le fait ceux du cerveau avec les signaux électriques qu'ils reçoivent.

Les principales fonctions d'activation Φ sont : la fonction linéaire, la fonction sigmoïde et la fonction tangente hyperbolique :

Linéaire $\Phi(p) = p$ (13)

Sigmoide $\Phi(p) = \frac{1}{1+e^{-cp}}$ $(c > 0)$ (14)

Tangente hyperbolique $\Phi(p) = \frac{1-e^{-cp}}{1+e^{-cp}}$ (15)



(a) linéaire (b) sigmoïde (c) tangente hyperbolique

Figure II.4 Fonctions d'activations classiques

La figure II.4 montre les trois fonctions d'activation classiques. Il faut remarquer que la fonction linéaire est dans] -∞ ; +∞ [, la fonction sigmoïde dans] 0 ; 1[et la fonction tangente hyperbolique dans] -1 ; 1[.

La propagation d'un vecteur d'entrée (xi) à travers un perceptron s'écrit donc :

$$p = \sum \omega i xi \qquad (16)$$

$$\mathcal{Y} = \Phi(p) \qquad (17)$$

L'apprentissage classique d'un perceptron est la régression où la fonction de Coût est de la forme:

$$L = \frac{1}{2}(o - d)^2 \qquad (18)$$

Où o est la sortie obtenue pour un échantillon et d est la sortie désirée u (label)

Le gradient de l'erreur pour le potentiel est :

$$\frac{\partial L}{\partial p} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial p} \qquad (19)$$

$$= \frac{\partial L}{\partial y}\Phi'(p)$$

Et le gradient de l'erreur pour l'entrée xi est :

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial x_i} \qquad (20)$$

$$= \frac{\partial L}{\partial p} w_i$$

Une fois la rétro propagation du gradient effectuée pour toutes les couches, les matrices de poids sont mises à jour .

L'algorithme de la rétropropagation

1. Initialiser les poids W , avec des valeurs aléatoires entre [-1, 1] .
2. Mélanger les exemples .
3. Pour tout exemple x faire :

a. Calculer les sorties de réseaux en propageant les entrées vers les sorties.
b. Corriger les poids en rétropropageant l'erreur :

$$w_{(x)ij} = w_{(x-1)ij} + \alpha\delta_{(x)j}o_{(x)i} \quad (22)$$

Avec :

$$\delta_{(x)j} = \begin{cases} e_{(x)j}o_{(x)j}[1 - o_{(x)j}] & si\ j \in la\ couche\ de\ sortie \\ o_{(x)j}[1 - o_{(x)j}]\left[\sum_{k\in c}\delta_{(x)k}W_{(x)jk}\right] & si\ j \in la\ couche\ de\ cachée \end{cases} \quad (23)$$

Ou :

x : l'exemple courant.
x - 1 : l'exemple précédent.
i : neurone i de la couche précédent.
j : neurone j de la couche suivante.
α : représente le taux ou le pas d'apprentissage.

4. Répéter les étapes 2 et 3 jusqu'à rempli le critère d'arrêt.

II.5 Deep learning (L'apprentissage profond)

II.5.1 Définition de l`apprentissage profond (deep learning)

L'apprentissage profond (deep learning) est une classe de méthodes dont les principes sont connus depuis la fin des années 80, mais dont l'utilisation ne s'est vraiment généralisée que depuis environ 2012 **[15]**. L'apprentissage profond peut être vu comme un réseau multicouche avec de nombreuses couches intermédiaires **[16][15]**. L'architecture profonde est capable d'extraire automatiquement des caractéristiques de l'image, par exemple les premières couches extrairont des contours que les couches suivantes forment en des concepts de plus en plus complexes et abstraits : des formes, des objets, de parties d'objets en objets, etc **[15].**

Le deep learning ou apprentissage profond est un type d'intelligence artificielle dérivé de machine learning (apprentissage automatique) où la machine est capable d'apprendre par elle-

même, contrairement à la programmation où elle se contente d'exécuter à la lettre des règles prédéterminées

II.5.2 Fonctionnement du deep Learning

Le deep Learning s'appuie sur un réseau de neurones artificiels s'inspirant du cerveau humain. Ce réseau est composé de dizaines voire de centaines de « couches » de neurones, chacune recevant et interprétant les informations de la couche précédente. Le système apprendra par exemple à reconnaître les lettres avant de s'attaquer aux mots dans un texte, ou détermine s'il y a un visage sur une photo avant de découvrir de quelle personne il s'agit a chaque étape, les « mauvaises » réponses sont éliminées et renvoyées vers les niveaux en amont pour ajuster le modèle mathématique. Au fur et à mesure, le programme réorganise les informations en blocs plus complexes. Lorsque ce modèle est par la suite appliqué à d'autres cas, il est normalement capable de reconnaître un chat sans que personne ne lui ait jamais indiqué qu'il n'ai jamais appris le concept de chat. Les données de départ sont essentielles : plus le système accumule d'expériences différentes, plus il sera performant.

II.5.3 Applications du deep Learning

Le deep Learning est utilisé dans de nombreux domaines :

- reconnaissance d'image.
- traduction automatique .
- voiture autonome.
- diagnostic médical.
- recommandations personnalisées.
- modération automatique des réseaux sociaux.
- prédiction financière et trading automatisé.
- identification de pièces défectueuses.
- détection de malwares ou de fraudes .
- chatbots (agents conversationnels).
- exploration spatiale.
- robots intelligents.

II.5.4 Quelques algorithmes de Deep Learning

Il existe différents algorithmes d"apprentissage profond qui utilise les méthodes:

- **Les réseaux de neurones profonds :** (Deep Neural Networks). Ces réseaux sont similaires aux réseaux MLP mais avec plus de couches cachées. L'augmentation du nombre de couches, permet à un réseau de neurones de détecter de légères variations du modèle d'apprentissage, favorisant le sur-apprentissage ou sur-ajustement (« overfitting »).
- **Les réseaux de neurones convolutionnels :** (CNN ou Convolutional Neural Networks). Le problème est divisé en sous parties, et pour chaque partie, un «cluster» de neurones sera créer afin d'étudier cette portion spécifique. Par exemple, pour une image en couleur, il est possible de diviser l'image sur la largeur, la hauteur et la profondeur (les couleurs).
- **La machine de Boltzmann profonde :** (Deep Belief Network): Ces algorithmes fonctionnent suivant une première phase non supervisée, suivi de l'entraînement classique supervisé. Cette étape d'apprentissage non-supervisée, permet, en outre, de faciliter l'apprentissage supervisé.

II.5.5 Les réseaux neuronaux convolutifs (CNN)

Les réseaux de neurones convolutifs (CNN) sont des variantes des réseaux profonds et permettent d'extraire les caractéristiques d'une image **[17]** . Leur fonctionnement est inspiré par les processus biologiques. Les réseaux neuronaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation.

II.5.5.1Architecture de réseaux de neurone convolutionnel

Une architecture CNN est formée par un empilement de couches de traitement indépendantes :

- La couche de convolution (CONV) qui traite les données d'un champ récepteur
- La couche de pooling (POOL), qui permet de compresser l'information en réduisant la taille de l'image intermédiaire (souvent par sous-échantillonnage)

- La couche de correction (ReLU), souvent appelée par abus 'ReLU' en référence à la fonction d'activation (Unité de rectification linéaire)
- La couche "entièrement connectée" (FC), qui est une couche de type perceptron.
- La couche de perte (LOSS)

II.5.5.1.1 Couche de convolution(CONV)

La couche de convolution est le bloc de construction de base du réseau convolutif. Ces couches se composent d'une grille rectangulaire de neurones qui ont un petit champ réceptif, mais s'étend à travers toute la profondeur du volume d'entrée. Ainsi, la couche de convolution est juste une convolution d'image de la couche précédente, où les poids spécifient le filtre de convolution **[15]**. La Figure II.5 montre un exemple de conv2d.

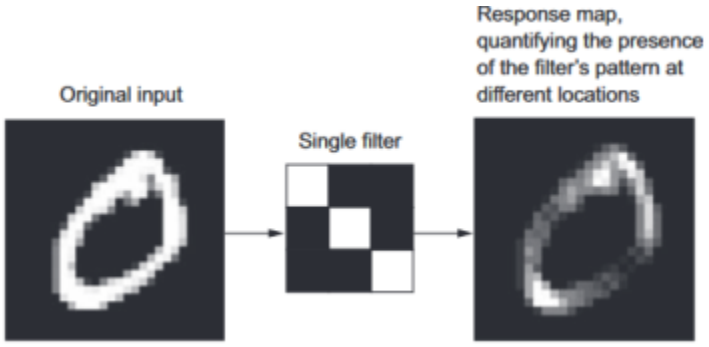


Figure II.5 montre exemple de conv2d

Les trois hyperparamètres permettent de dimensionner le volume de la couche de convolution :

- Profondeur : nombre de noyaux de convolution.
- Le pas : contrôle le chevauchement des champs récepteurs.
- Zéro padding : mettre des zéros à la frontière du volume d'entrée.

II.5.5.1.2 Couche de pooling (POOL)

Après chaque couche de convolution, il peut y avoir une couche de pooling. La couche de pooling est une forme de sous-échantillonnage de l'image entrée. (La Figure II.6) montre la mise en commun couramment utilisés, à savoir, pooling moyen et max pooling **[18] [19].** Le pooling réduit la taille spatiale d'une image intermédiaire, réduisant ainsi la quantité de paramètres et de calcul dans le réseau.

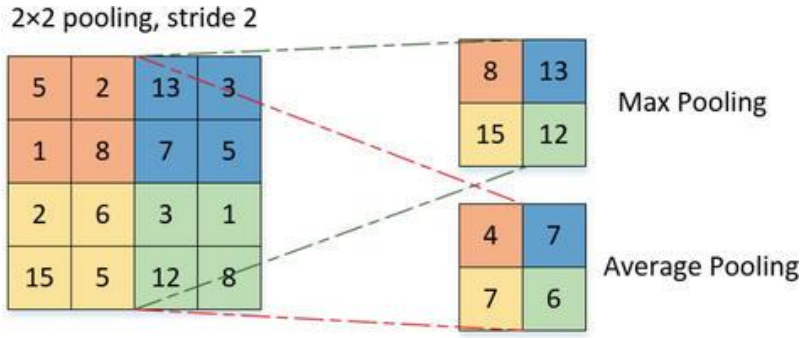


Figure II.6 Exemples de max et moyenne pooling sur une fenêtre 2 x 2 et pas 2

II.5.5.1.3 Couche de correction (RELU)

Souvent, il est possible d'améliorer l'efficacité du traitement en intercalant entre les couches de traitement une couche qui va opérer une fonction mathématique (fonction d'activation) sur les signaux de sortie. On a notamment :

- La correction ReLU (abréviation de Unités Rectifié linéaires)**:**f(x)=max (0,x).Cette fonction, appelée aussi« fonction d'activation non saturante»,augmente les propriétés non linéaires de la fonction de décision et de l'ensemble du réseau sans affecter les champs récepteurs de la couche de convolution.
- La correction par tangente hyperbolique f(x)=tanh(x).
- La correction par la tangente hyperbolique saturante : f(x)=|tanh(x)|.
- La correction par la fonction sigmoïde.

Souvent, la correction Relu est préférable, car il en résulte la formation de réseau neuronal plusieurs fois plus rapide, sans faire une différence significative à la généralisation de précision.

II.5.5.1.4 Couche entièrement connectée (FC)

Après plusieurs couches de convolution et de max dans le réseau neuronal se fait via des couches entièrement connectées. Les neurones dans une couche entièrement connectée ont des connexions vers toutes les sorties de la couche précédente (comme on le voit régulièrement dans les réseaux réguliers de neurones). Leurs fonctions d'activations peuvent donc être calculées avec une multiplication matricielle suivie d'un décalage de polarisation.

II.5.5.1.5 Couche de perte (LOSS)

La couche de perte spécifie comment l'entrainement du réseau pénalise l'écart entre le signal prévu et réel. Elle est normalement la dernière couche dans le réseau. Diverses fonctions de perte adaptées à différentes tâches peuvent y être utilisées. La perte « Soft max » est utilisée pour prédire une seule classe parmi K classes mutuellement exclusives. La perte par entropie croisée sigmoïde est utilisée pour prédire K valeurs de probabilité indépendante dans [0,1] . La perte euclidienne est utilisée pour régresser vers des valeurs réelles.

II.5.5.1.6 Couche Dropout

Dropout est l'une des techniques de régularisation des réseaux de neurones les plus efficaces et les plus couramment utilisées **[20]**, développé par Srivastava et al. **[21]**. Dropout, appliqué à une couche, consiste à abandonner de manière aléatoire (mettre à zéro) un certain nombre (précisé par une probabilité donner à Dropout) d'entités en sortie de la couche pendant l'entraînement **[20]** (voir figure II.7).

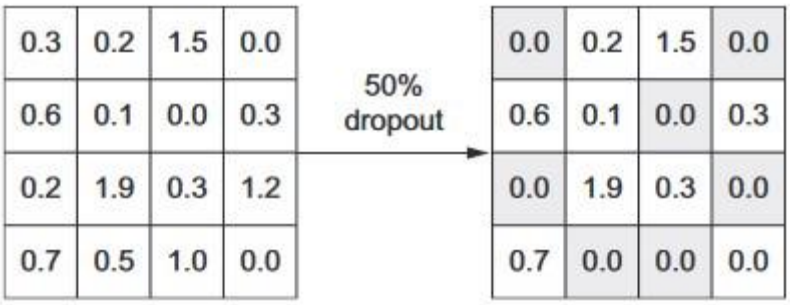


Fig. II.7 Exemple d'Utilisation Dropout avec une probabilité 0.5 (50%)[20]

II.5.5.1.7 Exemples de modèles de CNN

La forme la plus commune d'une architecture de réseau de neurones convolutifs empile quelques couches Conv-ReLU, les suit avec des couches Pool, et répète ce schéma jusqu'à ce que l'entrée soit réduite dans un espace d'une taille suffisamment petite. À un moment, il est fréquent de placer des couches entièrement connectées (FC). La dernière couche entièrement connectée est reliée vers la sortie. Voici quelques architectures communes de réseau de neurones convolutifs qui suivent ce modèle :

- INPUT -> CONV -> RELU -> FC
- INPUT -> [CONV -> RELU -> POOL] * 2 -> FC -> RELU -> FC Ici, il y a une couche de CONV unique entre chaque couche POOL
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL] * 3 -> [FC -> RELU] * 2 -> FC Ici, il y a deux couches CONV empilées avant chaque couche POOL.

L'empilage des couches CONV avec de petits filtres de pooling (plutôt un grand filtre) permet un traitement plus puissant, avec moins de paramètres.

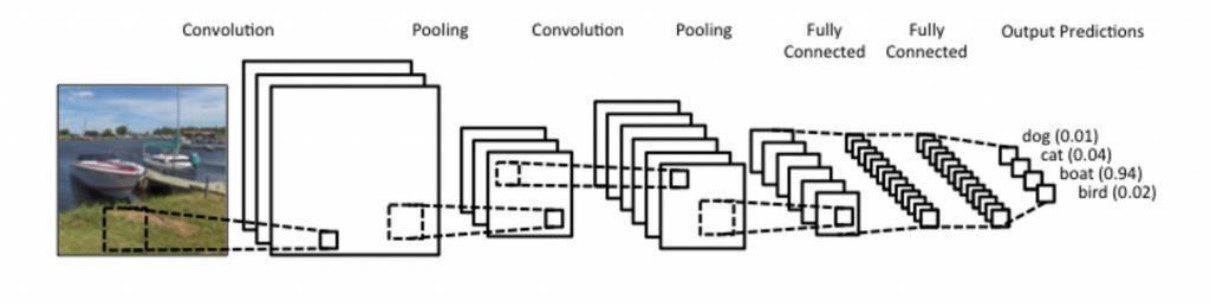


Figure II.8 : Exemples de modèles de CNN

II.5.5.1.8 Choix des paramètres

Les réseaux de neurones convolutifs utilisent plus d'hyper paramètres qu'un perceptron multicouche standard. Même si les règles habituelles pour les taux d'apprentissage et des constantes de régularisation s'appliquent toujours, il faut prendre en considération les notions de nombre de filtres, leur forme et la forme du max pooling. **[22]**

Nombre de filtres : Comme la taille des images intermédiaires diminue avec la profondeur du traitement, les couches proches de l'entrée ont tendance à avoir moins de filtres tandis que les couches plus proches de la sortie peuvent en avoir davantage. Pour égaliser le calcul à chaque couche, le produit du nombre de caractéristiques et le nombre de pixels traités est généralement choisi pour être à peu près constant à travers les couches. Pour préserver l'information en entrée, il faudrait maintenir le nombre de sorties intermédiaires (nombre d'images intermédiaires multiplié par le nombre de positions de pixel) pour être croissante (au sens large) d'une couche à l'autre .Le nombre d'images intermédiaires contrôle directement la puissance du système, dépend du nombre d'exemples disponibles et la complexité du traitement

Forme du filtre : Les formes de filtre varient grandement dans la littérature. Ils sont généralement choisis en fonction de l'ensemble de données. Les meilleurs résultats sur les images de MNIST (28x28) sont habituellement dans la gamme de 5x5 sur la première couche, tandis que les ensembles de données d'images naturelles (souvent avec des centaines de pixels dans chaque dimension) ont tendance à utiliser de plus grands filtres de première couche de 12x12, voire 15x15. Le défi est donc de trouver le bon niveau de granularité de manière à créer des abstractions à l'échelle appropriée et adaptée à chaque cas

Forme du Max Pooling : Les valeurs typiques sont 2x2. De très grands volumes d'entrée peuvent justifier un pooling 4x4 dans les premières couches. Cependant, le choix de formes plus grandes va considérablement réduire la dimension du signal, et peut entraîner la perte de trop d'information.

II.6 Conclusion

Dans ce chapitre on a présenté les notions importantes qui sont en relation avec l'apprentissage profond (définition, Architectures…etc.). Aussi qu'une vision générale sur l'apprentissage profond, toute on donnant en détail la méthode choisie dans notre travail de recherche qui est le CNNs. Le prochain chapitre, traite les détails de la conception, ainsi que la méthode et les outils utilisés pour la réalisation de notre application.

Chapitre III

Implémentation et réalisation

Chapitre III: Implémentation et réalisation

III.1 Introduction

Dans ce chapitre, nous allons présenter les expérimentations effectuées sur notre modèle afin de démontrer son efficacité à détecter covid 19 a partir de Scanner thoracique. Nous détaillerons d'abord les environnements matériels et logiciels que nous avons utilisés pour développer notre application, puis nous comparerons les résultats.

III. 2 Environnement de développement logiciel

III. 2.1 TensorFlow

TensorFlow[1] est un framework de programmation pour le calcul numérique qui a été rendu Open Source par Google en Novembre 2015. Depuis son release, TensorFlow n'a cessé de gagner en popularité, pour devenir très rapidement l'un des frameworks les plus utilisés pour le Deep Learning et donc les réseaux de neurones. Son nom est notamment inspiré du fait que les opérations courantes sur des réseaux de neurones sont principalement faites via des tables de données multi-dimensionnelles, appelées Tenseurs (Tensor). Un Tensor à deux dimensions est l'équivalent d'une matrice. Aujourd'hui, les principaux produits de Google sont basés sur TensorFlow: Gmail, Google Photos, Reconnaissance de voix.

III. 2.2 Keras

Keras[2] est une API de réseaux de neurones de haut niveau, écrite en Python et capable de fonctionner sur TensorFlow ou Theano. Il a été développé en mettant l'accent sur l'expérimentation rapide. Être capable d'aller de l'idée à un résultat avec le moins de délai possible est la clé pour faire de bonnes recherches. Il a été développé dans le cadre de l'effort de recherche du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), et son principal auteur et mainteneur est François Chollet, un ingénieur Google.

En 2017, l'équipe TensorFlow de Google a décidé de soutenir Keras dans la bibliothèque principale de TensorFlow. Chollet a expliqué que Keras a été conçue comme une interface plutôt

1. https://www.tensorflow.org/
2. https://keras.io/

que comme un cadre d'apprentissage end to end. Il présente un ensemble d'abstractions de niveau supérieur et plus intuitif qui facilitent la configuration des réseaux neuronaux indépendamment de la bibliothèque informatique de backend. Microsoft travaille également à ajouter un backend CNTK à Keras aussi.

III. 2.3 Python

Python[3] est un langage de programmation de haut niveau interprété (il n'y a pas d'étape de compilation) et orienté objet avec une sémantique dynamique. Il est très sollicité par une large communauté de développeurs et de programmeurs. Python est un langage simple, facile à apprendre et permet une bonne réduction du cout de la maintenance des codes. Les bibliothèques (packages) python encouragent la modularité et la réutilisabilité des codes.

III. 2.4 Scikit-learn

Scikit-learn est une bibliothèque libre Python dédiée à l'apprentissage automatique. Elle est développée par de nombreux contributeurs notamment dans le monde académique par des instituts français d'enseignement supérieur et de recherche comme Inria et Télécom ParisTech. Elle comprend notamment des fonctions pour estimer des forêts aléatoires, des régressions logistiques, des algorithmes de classification, et les machines à vecteurs de support. Elle est conçue pour s'harmoniser avec des autres bibliothèques libre Python, notamment NumPy et SciPy.

III. 2.5 Anaconda

Anaconda[4] est une distribution libre et open source des langages de programmation Python et R appliquée au développement d'applications dédiées à la fouille de données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique), qui vise à simplifier la gestion des paquets et de déploiement. Les versions dépaquetages sont gérées par le système de gestion de paquets conda. La distribution Anaconda est utilisée par plus de 6 millions d'utilisateurs et comprend plus de 250 paquets populaires fouille de des données adaptés pour Windows, Linux et MacOs.

3. https://www.python.org/
4. https://www.anaconda.com/

III. 2.6 Pycharm IDE

PyCharm est un environnement de développement intégré utilisé pour programmer en Python. Il permet l'analyse de code et contient un débogueur graphique. Il permet également la gestion des tests unitaires, l'intégration de logiciel de gestion de versions, et supporte le développement web avec Django. Développé par l'entreprise tchèque JetBrains, c'est un logiciel multi-plate forme qui fonctionne sous Windows, Mac OS X et Linux. Il est décliné enédition professionnelle, diffusé sous licence propriétaire, et en édition communautaire diffusé sous licence Apache.

III. 2.7 PyQt

PyQt[5] est l'une des liaisons Python les plus populaires pour le framework C ++ multiplateforme Qt. PyQt a été développé par Riverbank Computing Limited. Qt lui-même est développé dans le cadre du projet Qt. PyQt fournit des liaisons pour Qt 4 et Qt 5. PyQt est distribué sous un choix de licences: GPL version 3 ou une licence commerciale.

PyQt est disponible en deux éditions: PyQt4 qui compilera contre Qt 4.x et 5.x et PyQt5 qui ne compilera que contre 5.x. Les deux éditions peuvent être construites pour Python 2 et 3. PyQt contient plus de 620 classes qui couvrent les interfaces utilisateur graphiques, la gestion XML, la communication réseau, les bases de données SQL, la navigation Web et d'autres technologies disponibles dans Qt.

III. 2.8 OpenCV

OpenCV[6] (Open source Computer Vision) est une bibliothèque libre de vision par ordinateur, qui possède plus de 2500 algorithmes optimisés, La bibliothèque comprend un ensemble complet d'algorithmes de vision par ordinateur et d'algorithmes d'apprentissage. Elle existe depuis une décennie et est publiée sous la licence Berkeley Software Distribution (BSD), ce qui facilite l'utilisation et la modification du code par les utilisateurs.

Les modules intégrés d'OpenCV sont suffisamment puissants et polyvalents pour résoudre la plupart des problèmes de vision informatique pour lesquels des solutions bien établies sont

5. https://pypi.org/project/PyQt5/
6. https://opencv.org/

disponibles. Ils sont destinés à des applications en temps réel et conçus pour être exécutés très rapidement.

OpenCV a été officiellement lancé en tant que projet de recherche au sein d'Intel Research afin de faire progresser les technologies dans les applications gourmandes en ressources. Parmi les principaux contributeurs au projet figuraient des membres d'"Intel Research Russia" et de l'équipe "Performance Library d'Intel".

III. 3 Configuration matérielle

III.3.1 Configuration matérielle distante (Google Colab)

Google Collaboratory[7] ou Colab1, un outil Google simple et gratuit pour vous initier au Réseaux profonds ou collaborer avec vos collègues sur des projets en science des données .Colab permet d'améliorer vos compétences de codage en langage de programmation Python, de développer des applications en Réseaux Profonds en utilisant des bibliothèques populaires telles que Keras, TensorFlow, PyTorch et OpenCV sans installation ainsi que d'utiliser un environnement de développement (Jupyter Notebook) qui ne nécessite aucune configuration. Cependant, chaque 12 heures, la machine virtuelle mise à disposition par Google est réinitialisée nécessitant un mécanisme de sauvegarde des données en cours .De plus, les documents Colab (Jupyter Notebook) sont enregistrés directement votre compte Google Drive**.**

L'infrastructure distante mise à disposition par Google Colab et utilisée pour l'entraî-nement possède cette configuration :

- Processeur Intel Core Xeon CPU @2.3Ghz, 45MB Cache.
- Processeur graphique NVIDIA Tesla K80, ayant 2496 cœurs CUDA, Compute 3,7, 12 Go (11.439 Go utilisable) GDDR5 VRAM.
- Mémoire vive de 12.6 Go.
- Disque dur de capacité 320 Go.
- Jupyter Notebook
- Système d'exploitation Linux x64 bits.

7. https://colab.research.google.com

III.3.2 Configuration matérielle local

Pour la reconnaissance, nous avons utilisé un pc portable personnel possédant cette configuration:

- Processeur Intel Core i5-9400f CPU @3.5Ghz,5 .0 GhZ.
- Processeur graphique NVIDIA GEFORCE GTX 1060 3Go VRAM
- Mémoire vive de 8 Go
- Disque dur hybride SSD de capacité 256 Go et HDD de capacité 1 To
- Système d'exploitation Windows 10 x64 bits

III.3.3 Les base d'images

Kaggle[8] est une collection de données scientifiques. Kaggle permet aux utilisateurs de rechercher et de publier des ensembles de données et des explorateurs, de créer des modèles dans un environnement de science des données basé sur le Web, de travailler avec d'autres données scientifiques et des moteurs de correspondance automatique, et de participer à des concours pour résoudre les failles de données.

III.3.4 Architecture de notre réseau

Au cours de nos expérimentations, nous avons créé trois modèles (modèle 1, modèle 2 et modèle 3) avec différentes architectures, où on a appliqué les trois modèles sur la base d'images
Dans ce qui suit on présente l'architecture des trois modèles :

Architecture du modèle 01

Le premier modèle que nous présentons dans la figure III.1 est composé de cinq couches de convolution et deux couches de maxpooling et trois couches de fullyconnected.
L'image en entrée est de taille 244*244, l'image passe d'abord à la première couche de convolution. Cette couche est composée de 32 filtres de taille 3*3, Chacune de nos couches de convolution est suivie d'une fonction d'activation ReLU cette fonction force les neurones à retourner des valeurs positives, après cette convolution 32 featuresmaps de taille 242*242 seront créés.

8. https://www.kaggle.com/