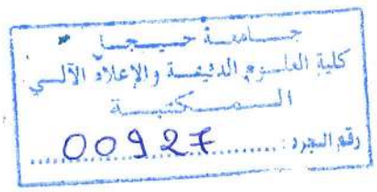
République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohammed Seddik BENYAHIA – JIJEL

Faculté des Sciences Exactes et Informatique

Département d'Informatique



Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option : Intelligence Artificielle

Thème

La reconnaissance de formes par les réseaux de neurones

Réalisé par :

Mlle. Hamdi Manal

Mme. Ayeche Radia

Encadré par :

Dr. Kara Messaoud

Promotion – 2017/2018

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohammed Seddik BENYAHIA – JIJEL

Faculté des Sciences Exactes et Informatique

Département d'Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master en Informatique

Option : Intelligence Artificielle

Thème

La reconnaissance de formes par les réseaux de neurones

Réalisé par :

Mlle. Hamdi Manal

Mme. Ayeche Radia

Encadré par :

Dr. Kara Messaoud

Promotion – 2017/2018

Remerciements

Nos premiers remerciements s'adressent à Dieu le tout puissant qui par sa bonté et sa miséricorde nous a permis d'avoir le courage, la foi et la volonté de mener à bien ce travail.

Nous tenons aussi à remercier Monsieur Kara Messaoud, notre encadreur, pour sa patience, et surtout ses judicieux conseils, qui a été présent à nos cotés à tout moment de la réalisation de ce projet.

Nos remerciements vont aussi aux membres du jury pour l'intérêt qu'ils ont porté à notre travail en acceptant d'examiner notre mémoire et de l'enrichir par leurs propositions.

Sans oublier toute personne qui nous a aidés de près ou de loin à mener à terme notre projet.

Merci à tous.

Dédicace

A nos chers parents qui nous ont soutenus et encouragés durant toute notre scolarité.

A nos frères et sœurs.

A nos enseignant(e)s.

A nos ami(e)s.

A toutes les personnes qui nous ont apportés de l'aide.

Hamdi Manal

Ayeche Radia

Résumé

Les réseaux de neurones artificiels ont réussi à prendre une place dans différents domaines tels que la reconnaissance de formes et la classification. Il existe plusieurs types de réseaux de neurones (NN– Neural Network) comme les réseaux de neurones convolutionnels (CNN– Convolutional Neural Network), et les réseaux de neurones récurrents (RNN– Recurrent Neural Network).

Dans notre travail, nous avons utilisé les CNN pour reconnaitre et classifier des images. Nous avons choisi d'appliquer deux modèles : Le premier est l'architecture du modèle VGGNet (Visual Geometry Group) connu par la profondeur de ces couches ; le deuxième que nous avons créé et qui a moins de couches.

Les résultats obtenus après plusieurs essais, montrent que le nombre d'époques, la taille de la base d'images, la profondeur du réseau influencent les performances du réseau de neurones.

Abstract

Artificial neural networks have managed to take a place in different areas such as pattern recognition and classification. Several types of Neural Networks (NN) exist such as Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).

In our work we used the CNN to recognize and classify images, so we chose to apply two models: The first one is the architecture of the VGGNet model well known for the depth of its layers and the second one we created and which has fewer layers.

The results obtained, after several experiments, show that the number of epochs, the size of the image base, the depth of the network influence the neural network performance.

ملخص

تمكنت الشبكات العصبية الاصطناعية (Neural Network) من الدخول إلى ميادين مختلفة، مثل التعرف على الأشكال و تصنيفها. هناك عدة أنواع من الشبكات العصبية الاصطناعية نخص بالذكر منها الشبكات العصبية الالتفافية (CNN) و الشبكات العصبية المتكررة (RNN) .

في عملنا هذا، استخدمنا الشبكات العصبية الالتفافية (CNN) للتعرف على الصور و تصنيفها، حيث اخترنا تطبيق نموذجين: الأول هو بنية نموذج VGGNet المعروف بعمق طبقاته و اقترحنا نموذجا آخر له عدد أقل من الطبقات.

النتائج التي تم الحصول عليها، بعد عدة تجارب، تبين أن عدد التكرارات، حجم قاعدة الصور وعمق الشبكة لها تأثير على فعالية أداء الشبكة.

Les mots clés : les réseaux de neurones, la reconnaissance de formes, la classification, convolution, maxpooling, couche entièrement connectée, perceptron, perceptron multi couche.

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

Acronyme	Description
RN	Réseau de Neurones
RNA	Réseau de Neurones Artificiels
RNF	Réseau de Neurones Formels
NN	Neural Network
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
MLP	Multi Layer Perceptron
PMC	Perceptron Multi Couches
FC	Fully Connected
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
RBF	Radial Basis Function

INTRODUCTION GENERALE

L'informatique est la science du traitement automatique de l'information par ordinateur. Depuis l'émergence de l'intelligence artificielle dans les années 1950, les chercheurs essaient d'injecter et d'ajouter des notions d'intelligence spécifiques aux humains à des machines pour qu'elles soient capables d'accomplir des tâches comme un être humain que ce soit pour faire des mathématiques, jouer, parler, détecter …

Le traitement automatique de l'information a connu différentes approches ; les plus utilisées sont l'approche algorithmique et l'approche basée sur la connaissance. Mais ces deux approches ne suffisent pas à répondre à tous les problèmes existants dans plusieurs domaines comme la reconnaissance de formes (image, signal), le diagnostic, le contrôle moteur, la traduction automatique. Une troisième approche est apparue où elle est inspirée du fonctionnement du cerveau humain c'est les réseaux de neurones artificiels qui sont capables de résoudre des problèmes complexes que les autres approches ne leurs trouvent pas de solution.

En 1995, et après plusieurs itérations Yann leCun et son équipe ont réussi à développer un réseau de neurones convolutionnels appelé LeNet5 qui est utilisé dans un système de lecture automatique de chèques qui a été largement déployé dans le monde. Malgré ce grand succès, les réseaux neuronaux ont été délaissés durant un bon moment depuis 1997 jusqu'à 2012.

Au début de l'année 2012, les chercheurs commencent à utiliser les réseaux de neurones de nouveau après les expériences menées simultanément chez Microsoft, Google et IBM avec l'aide du laboratoire de Geoff Hinton et qui ont montré que les réseaux profonds pouvaient diminuer de moitié les taux d'erreurs des systèmes de reconnaissance vocale. En s'inspirant du modèle LeNet5 plusieurs autres réseaux sont développés comme AlexNet en 2012, ZFNet en 2013, VGGNet et GoogleNet en 2014.

Dans notre travail, nous allons utiliser les réseaux de neurones pour reconnaitre des formes définies dans un contexte routier (bus, camion, personne, vélo, voiture, …) en utilisant deux modèles en les évaluant sur une base d'images que nous avons créée à partir notamment de la base ImageNet.

Organisation du mémoire :

Notre mémoire est structuré en quatre chapitres :

Dans le premier chapitre, nous ferons une présentation générale des réseaux de neurones et leur développement depuis leur apparition et jusqu'à nos jours ainsi que leurs applications. Une brève description de la reconnaissance de forme est également fournie.

Dans le deuxième chapitre, nous nous intéresserons aux réseaux de neurones convolutionnels et leur importance dans le domaine de la reconnaissance de formes.

Dans le troisième chapitre nous présentons quelques modèles existants et montrons les différences entre eux.

Le quatrième chapitre est consacré à la présentation des deux modèles que nous avons créés et aux discussions des résultats obtenus lors des tests que nous avons menés avec ces deux modèles.

A la fin, une conclusion générale termine ce mémoire.

CHAPITRE I : GENERALITES SUR LES RESEAUX DE NEURONES

1 Introduction

Depuis une dizaine d'années, les réseaux de neurones artificiels (RNA) sont devenus très utilisés dans de nombreux domaines pour résoudre les problèmes de classification, d'optimisation, de reconnaissance de formes etc.

Dans ce chapitre nous commençons par une brève description de la reconnaissance de forme puis nous présentons les réseaux de neurones d'une manière générale ainsi que leur développement avec le temps, en citons leurs différents types.

2 La reconnaissance de formes

La reconnaissance de formes ou parfois reconnaissance de motifs est un ensemble de techniques et méthodes visant à identifier des formes informatiques à partir de données brutes afin de prendre une décision dépendant de la catégorie attribuée à cette forme. On considère que c'est une branche de l'intelligence artificielle qui fait largement appel aux techniques d'apprentissage automatique et aux statistiques.

Le mot forme est à comprendre dans un sens très général, pas seulement celui de « forme géométrique » (cercle, ellipse, losange, ...) mais plutôt de motifs qui peuvent être de natures très variées. Il peut s'agir de contenu visuel (code barre, visage, empreinte digitale...) ou sonore (reconnaissance de parole), d'images médicales (rayon X, IRM...) ou multispectrales (images satellitaires) et bien d'autres.

La reconnaissance de formes peut être effectuée au moyen de divers algorithmes d'apprentissage automatique tels :

- Un réseau de neurones.
- Une analyse statistique (méthode bayésienne, estimation paramétrique, ...).
- L'utilisation de modèles de Markov cachés.
- Une recherche d'isomorphisme de graphes ou sous-graphes.

Les formes recherchées peuvent être des formes géométriques, descriptibles par une formule mathématique, ou elles peuvent aussi être de nature plus complexe tels que : lettre, chiffre, empreinte digitale. **[2]**

2.1 Applications

- La classification des images,
- Recherche d'images par le contenu,
- La reconnaissance de l'écriture manuscrite des caractères,
- La reconnaissance optique de caractères,

- La segmentation d'images,
- La reconnaissance vocale, ...

3 Biologie du neurone

Un neurone biologique reçoit plusieurs informations (qui sont des neurotransmetteurs) au niveau de ses dendrites. Ces neurotransmetteurs sont libérés par les synapses. Lorsque la quantité d'information dépasse un certain seuil, le neurone s'active et envoi un courant électrique dans son axone ce qui lui permet d'émettre à son tour des neurotransmetteurs via ses synapses.

Pour résumer :

- Les sorties d'un neurone sont les entrées d'un autre.
- Un neurone émet lorsqu'il reçoit une quantité d'information dépassant un seuil.
- La quantité d'information émise au neurone suivant est gérée par les synapses. **[1]**

Un neurone artificiel (ou formel) est un opérateur algébrique qui effectue une somme pondérée de ses entrées, appelée potentiel. Sa sortie est une fonction de ce potentiel. **[6]**

L'image ci-dessous représente une analogie entre le neurone biologique et le neurone artificiel.

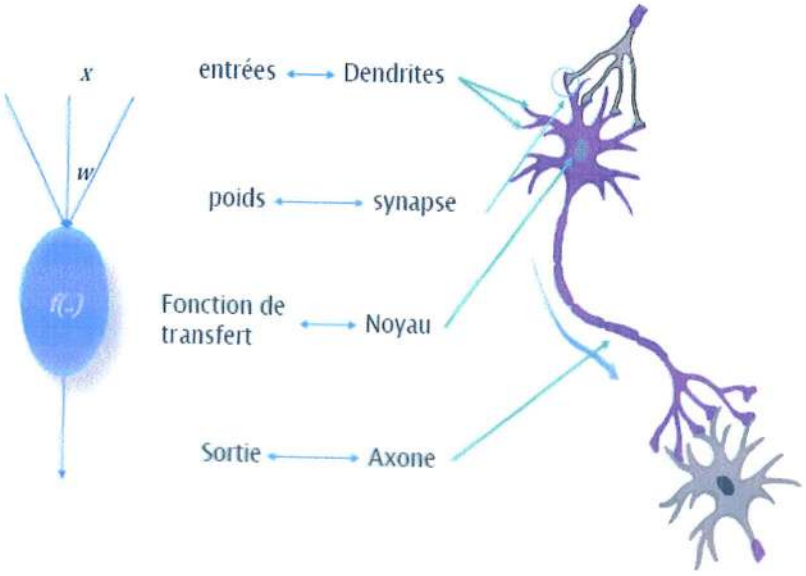


Figure I-1 : Analogie entre neurone biologique & neurone artificiel.

4 Fonctionnement général d'un réseau de neurones

4.1 Neurone artificiel

Figure I-2 : Principe d'un réseau de neurones

Le neurone artificiel, comme le réseau de neurones peut être vu, une fois entrainé, comme une fonction émettant une prédiction $\hat{y}$ par rapport à une variable d'entrée x (Voir Figure I-2).

Plus techniquement, le neurone artificiel est une combinaison d'une opération affine avec une fonction d'activation f, comme on peut le voir sur la figure 1-3. Pour faire une analogie avec le neurone biologique, les poids w_i représentent les synapses, le z les dendrites et le y est l'axone. Le seuil à dépasser est modélisé par la fonction f. Pour simplifier les opérations, nous mettrons toujours le terme de biais (b) dans la matrice de poids W et en augmentant la dimension de notre entrée x par 1 **[1]**.

L'équation ci-dessous, modélise un neurone qui a deux entrées x_1 et x_2.

$$y = f(z) = f\left(\sum_{i=1}^{2} w_i x_i - b\right) = f\left(\sum_{i=0}^{2} w'_i x'_i\right), \boldsymbol{W}^T = (-b \quad w_1 \quad w_2), \boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \qquad (1)$$

Il n'y a pas de définition universellement acceptée de réseaux de neurones. On considère généralement qu'un réseau de neurones est constitué d'un grand ensemble d'unités (ou neurones), ayant chacune une petite mémoire locale. Ces unités sont reliées par des canaux de communication (les connexions, aussi appelées synapses d'après le terme biologique correspondant), qui transportent des données numériques. Les unités peuvent uniquement agir sur leurs données locales et sur les entrées qu'elles reçoivent par leurs connexions.

La plupart des réseaux de neurones ont une certaine capacité d'apprentissage. Cela signifie qu'ils apprennent à partir d'exemples, de même que les enfants apprennent à distinguer les chiens des chats à partir d'exemples de chiens et de chats. Le réseau peut ensuite dans une certaine mesure être capable de généraliser, c'est-à-dire de produire des résultats corrects sur des nouveaux cas qui ne lui avaient pas été présentés au cours de l'apprentissage. **[3]**

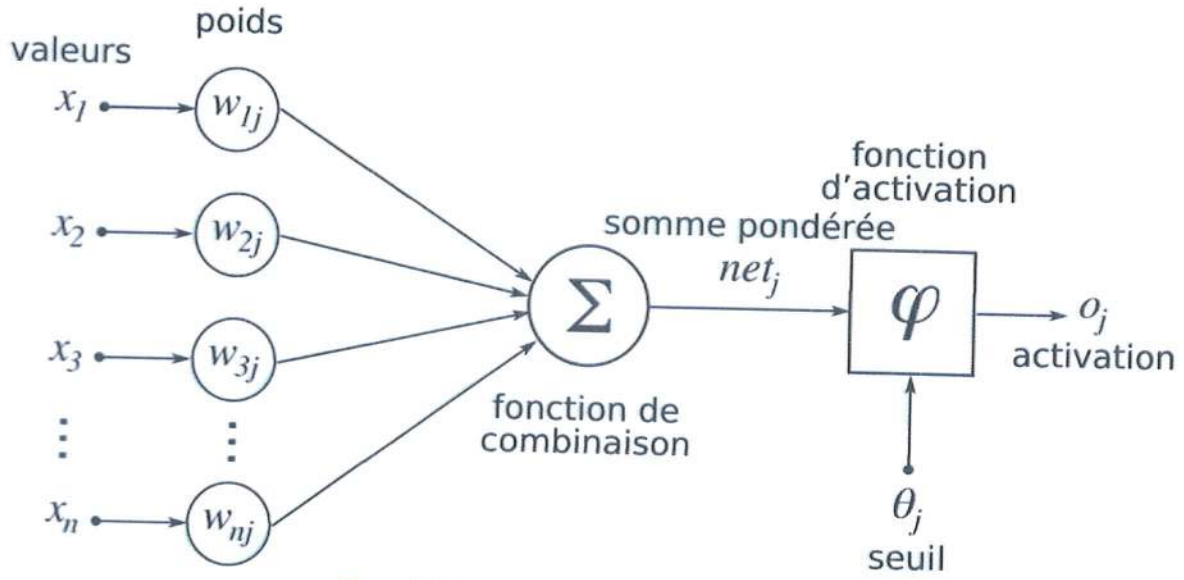


Figure I-3 : Représentation du neurone artificiel.

4.2 Modélisation d'un réseau de neurone

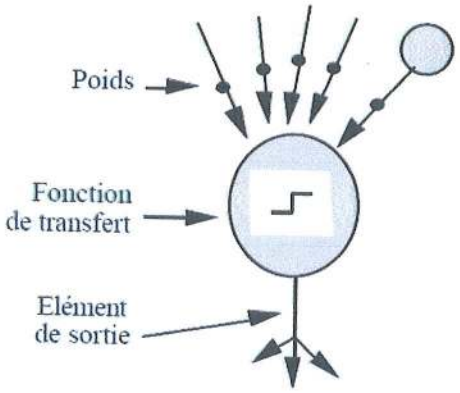


Figure I-4 : Modélisation d'un réseau de neurone.

Chaque neurone a : des entrées, comme le montre le schéma ci-dessus et chacune des entrées est pondérée par un poids, et une sortie du neurone. Une fois l'entrée est connue le neurone applique une fonction d'activation qui détermine la valeur de la sortie. **[3]**

Voici quelques fonctions couramment utilisées comme fonction d'activation :

La fonction de Heaviside :

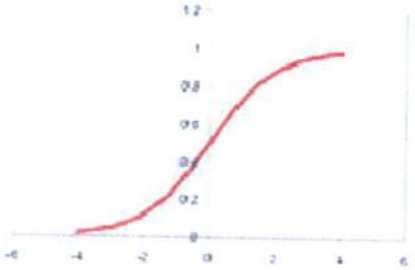
$$\varphi(v) = \begin{cases} 1 \text{ si } v \geq 0 \\ 0 \text{ si } v < 0 \end{cases}$$

si $s \in \{0, 1\}$, ou

$$\varphi(v) = \begin{cases} 1 \text{ si } v \geq 0 \\ -1 \text{ si } v < 0 \end{cases}$$

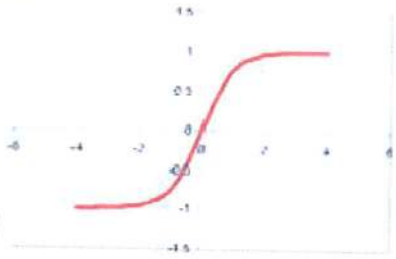
si $s \in \{-1, 1\}$.

La fonction sigmoïde (ou logistique) :



$$\varphi(v) = \frac{1}{1 + e^{-av}}$$

La fonction hyperbolique :



$$\varphi(v) = \tanh(v)$$

Et d'autres fonctions d'activation comme la fonction gaussienne, ReLU ...

4.3 Connectivité

La connectivité des réseaux est la manière dont les neurones sont connectés entre eux. Elle peut être totale (tous les neurones sont connectés entre eux) ou organisée par couche (les neurones d'une couche ne sont connectés qu'à la couche précédente en entrée et à la couche suivante en sortie). Il existe des réseaux monocouches ou multicouches. **[3]**

4.4 Calcul des poids synaptiques

La rétropropagation est une méthode de calcul des poids (aussi appelés poids synaptiques. Du nom des synapses, terme désignant la connexion biologique entre deux neurones) pour un réseau à apprentissage supervisé qui consiste à minimiser l'erreur quadratique de sortie (somme des carrés de l'erreur de chaque composante entre la sortie réelle et la sortie désirée).

D'autres méthodes de modification des poids sont plus "locales", chaque neurone modifie ses poids en fonction de l'activation ou non des neurones proches. **[3]**

4.5 Apprentissage

On appelle « apprentissage » des réseaux de neurones la procédure qui consiste à estimer les paramètres des neurones du réseau, afin que celui-ci remplisse au mieux la tâche qui lui est affectée.

En fonction du type du problème traité, on peut avoir à mettre en place différents types d'apprentissage : apprentissage supervisé, apprentissage par renforcement, apprentissage semi-supervisé ou apprentissage non-supervisé.

4.5.1 L'apprentissage supervisé

Cette approche a pour objectif la conception d'un modèle reliant des données d'apprentissage à un ensemble de valeurs de sortie (un comportement). **[4]**

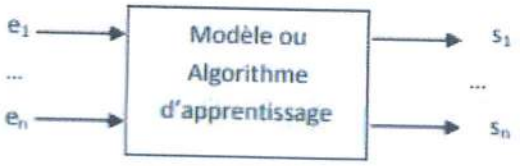


Figure I-5 : Schéma d'un modèle supervisé.

4.5.2 Apprentissage par renforcement

Les données en entrée sont les mêmes que pour l'apprentissage supervisé, cependant l'apprentissage est guidé par l'environnement sous la forme de récompenses ou de pénalités données en fonction de l'erreur commise lors de l'apprentissage. **[4]**

4.5.3 L'apprentissage semi supervisé

Les données d'entrée sont constituées d'exemples étiquetés et non étiquetés. Ce qui peut être très utile quand on a deux types de données, car cela permet de ne pas en laisser de côté et d'utiliser toute l'information. **[4]**

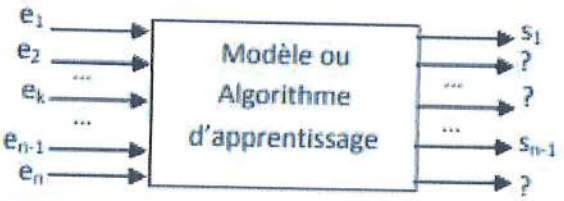


Figure I-6 : Schéma d'un modèle semi supervisé ou incrémental.

4.5.4 *L'apprentissage non supervisé*

Il vise à concevoir un modèle structurant l'information. La différence ici est que les comportements (ou catégories ou encore les classes) des données d'apprentissage ne sont pas connus, c'est ce que l'on cherche à trouver. **[4]**

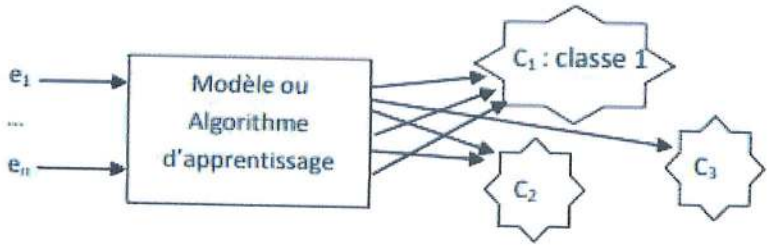


Figure I-7 : Schéma d'un modèle non supervisé.

5 *Quelques réseaux célèbres*

Il y a de très nombreuses sortes de réseaux de neurones actuellement. On présente ici les plus classiques.

5.1 Le perceptron

C'est un des premiers réseaux de neurones, conçu en 1958 par Rosenblatt. Il est linéaire et monocouche. Il est inspiré du système visuel. La première couche (d'entrée) représente la rétine. Les neurones de la couche suivante (unique, d'où le qualificatif de monocouche) sont les cellules d'association, et la couche finale les cellules de décision. Les sorties des neurones ne peuvent prendre que deux états (-1 et 1 ou 0 et 1).

Seuls les poids des liaisons entre la couche d'association et la couche finale peuvent être modifiés. La règle de modification des poids utilisée est la règle de Widrow-Hoff : si la sortie du réseau (donc celle d'une cellule de décision) est égale à la sortie désirée, le poids de la connexion entre ce neurone et le neurone d'association qui lui est connecté n'est pas modifié.

Dans le cas contraire le poids est modifié proportionnellement à la différence entre la sortie obtenue et la sortie désirée :

$$w \leftarrow w + k\,(d - s)$$

Où s est la sortie obtenue, d est la sortie désirée et k est une constante positive.

En 1969, Papert et Minsky ont démontré les limites du perceptron classique, incapable, par exemple de simuler la fonction ou exclusif (XOR). **[3]**

5.2 Le perceptron multi-couches

Le perceptron multi-couches (PMC) est une amélioration du perceptron comprenant une ou plusieurs couches intermédiaires dites couches cachées, dans le sens où elles n'ont qu'une utilité intrinsèque pour le réseau de neurones et pas de contact direct avec l'extérieur. Chaque neurone n'est relié qu'aux neurones des couches directement précédente et suivante, mais à tous les neurones de ces couches.

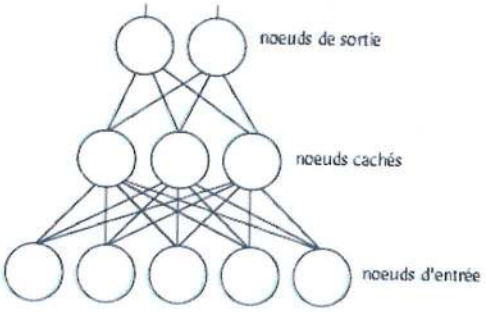


Figure I-8 : Le perceptron multi-couches.

Les PMC utilisent, pour modifier leurs poids, un algorithme de rétro propagation du gradient qui est une généralisation de la règle de Widrow-Hoff. Il s'agit toujours de minimiser l'erreur quadratique. On propage la modification des poids de la couche de sortie jusqu'à la couche d'entrée. **[3]**

5.3 Les réseaux de Hopfield

Il s'agit d'un réseau constitué de neurones à deux états (-1 et 1, ou 0 et 1), dont la loi d'apprentissage est la règle de Hebb (1949), qui veut qu'une synapse améliore son activité si et seulement si l'activité de ses deux neurones est corrélée. C'est-à-dire que le poids d'une connexion entre deux neurones augmente quand les deux neurones sont activés au même temps. **[3]**

5.4 Les réseaux de Kohonen

Contrairement aux réseaux de Hopfield où les neurones sont modélisés de la façon la plus simple possible, on recherche ici un modèle de neurones plus proches de la réalité. Ces réseaux sont inspirés des observations biologiques du fonctionnement des systèmes nerveux de perception des mammifères. Une loi de Hebb modifiée (tenant compte de l'oubli) est utilisée pour l'apprentissage. La connexion est renforcée dans le cas ou les neurones reliés ont une activité simultanée, et diminuée dans le cas contraire (alors qu'il ne se passait précédemment rien dans ce cas).

Les réseaux de Kohonen ont des applications dans la classification, le traitement d'images, l'aide à la décision et l'optimisation. **[3]**

5.5 Les réseaux de RBF (Radial Basis Fonction)

Le réseau RBF est un réseau de neurones supervisé. Il s'agit d'une spécialisation d'un PMC. Un RBF est constitué uniquement de 3 couches (voir Figure II-3)

- La couche d'entrée : Elle retransmet les entrées sans distorsion.
- La couche RBF : Couche cachée qui contient les neurones RBF.
- La couche de sortie : Simple couche qui contient une fonction linéaire.

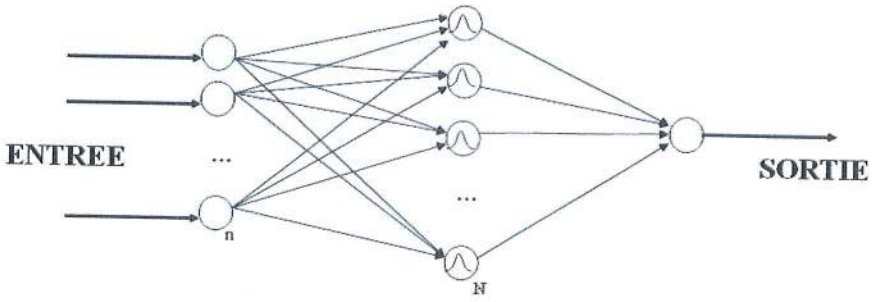


Figure I-9 : Schéma d'un réseau RBF.

Chaque neurone RBF contient une gaussienne qui est centrée sur un point de l'espace d'entrée. Pour une entrée donnée, la sortie du neurone RBF est la hauteur de la gaussienne en ce point. La fonction gaussienne permet aux neurones de ne répondre qu'à une petite région de l'espace d'entrée, région sur laquelle la gaussienne est centrée. Donc il y a quatre paramètres principaux à régler dans un réseau RBF.

- Le nombre de neurones RBF (nombre de neurones dans l'unique couche cachée).
- La position des centres des gaussiennes de chacun des neurones.
- La largeur de ces gaussiennes.
- Le poids des connexions entre les neurones RBF et le(s) neurone(s) de sortie.

Toute modification d'un de ces paramètres entraîne directement un changement du comportement du réseau. **[5]**

6 *Classification*

Une grande catégorie de problèmes industriels consiste à attribuer de façon automatique un objet à une classe, parmi plusieurs classes possibles. La résolution de ce type de problèmes demande de représenter les objets à l'aide de descripteurs, ou entrées (la représentation des objets par des graphes ou des chaines de caractères). La représentation choisie doit être discriminante du point de vue de la classification désirée. Les valeurs des entrées sont qualitatives (méthodes, opérateurs, services…) ou bien quantitatives (réelles ou binaires).

Exemples :

- Classification de chiffres décimaux manuscrits : Les entrées prennent la valeur 0 ou 1, selon celles des pixels de l'image binarisée du chiffre présent. Les classes d'appartenance sont {0,1,...9}.
- Classification de pages web : Les entrées sont les nombres d'occurrences de certains mots-clés ('distique', 'quintil', 'Aliénor', 'tercet', 'sizain', 'moyen-âge'). Les classes d'appartenance sont des thèmes d'intérêt (poésie, histoire).

Un classifieur est ainsi conçu pour attribuer un objet représenté par un vecteur d'entrée à une classe d'appartenance. Il réalise ainsi une fonction des entrées représentant l'objet. La valeur de cette fonction qui est la sortie du classifieur, fournit une classe d'appartenance.

Un classifieur à deux classes définit une surface de séparation dans l'espace des entrées : la sortie prend deux valeurs différentes (ici 0 et 1) de part et d'autre de la surface de séparation.

Un objet inconnu est affecté à l'une des deux classes à partir de sa position par rapport à la surface. La surface de séparation la plus simple est un hyperplan. La figure au-dessous illustre la séparation par une droite d'objets représentés par deux descripteurs x_1, x_2.

Exemples de la classe c_1 : o.

Exemples de la classe c_2 : x.

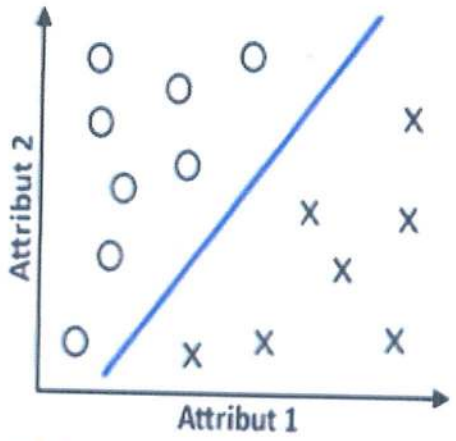


Figure I-10 : Problème de classification à deux classes.

La surface de séparation la plus simple est un hyperplan, par exemple une droite, comme ici, où les objets sont representes par deux descrepteurs $x_1$et x_2, lasortie du classifieur separateur et noteé y.

Plus généralement, un classifieur à c classes attribue à un objet inconnu une classe d'appartenance parmi c classes possibles. Pour ce faire, la sortie doit pouvoir prendre c valeurs différentes. On pourrait choisir une sortie vectorielle avec un codage 'binaire pur' mais on lui préfère toujours le codage '***un parmi c***'. Ce dernier est appelé codage 'grand-mère' dans la communauté neuronale : la sortie est un vecteur de dimension c, la classe c_i, étant codée par le vecteur dont toutes les composantes valent 0 sauf la $i^{éme}$, qui vaut 1.

Si l'on envisage la classification d'un point de vue probabiliste, l'entrée est considérée come un vecteur aléatoire. Dire que la représentation choisie est discriminante signifie que les densités de probabilité de ce vecteur pour chacune des classes se recouvrent peu. Les décisions de classification peuvent alors être prises à partir des probabilités d'appartenance de l'objet inconnu à chacune des ***c*** classes possibles. **[6]**

7 *Les types de réseaux de neurones artificiels*

Il existe essentiellement deux types de réseaux de neurones : les réseaux non bouclés (en anglais static, ou feedforward neural networks) et les réseaux bouclés (dynamic, ou feedback, ou recurrent neural networks).

7.1 Réseaux de neurones non bouclés

Les réseaux de neurones artificiels non bouclés sont les réseaux les plus utilisés, aussi bien en modélisation qu'en classification.

Un réseau de neurones non bouclé réalise une (ou plusieurs) fonctions algébriques de ses entrées, par composition des fonctions réalisées par chacun des neurones.

Un réseau de neurones non bouclé est donc représenté graphiquement par un ensemble de neurones « connectés » entre eux, l'information circulant des entrées vers les sorties sans « retour en arrière » : si l'on représente le réseau comme un graphe dont les nœuds sont les neurones et les arêtes les «connexions» entre ceux-ci, le graphe d'un réseau non bouclé est acyclique : si l'on se déplace dans le réseau, à partir d'un neurone quelconque, en suivant les connexions, on ne peut pas revenir au neurone de départ. Les neurones qui effectuent le dernier calcul de la composition de fonctions sont les neurones de sortie ; ceux qui effectuent des calculs intermédiaires sont les neurones cachés.

La seule contrainte sur le graphe des connexions d'un réseau de neurones non bouclé est qu'il ne contienne pas de cycle. On peut donc imaginer une grande variété de topologies pour ces réseaux.**[7]**

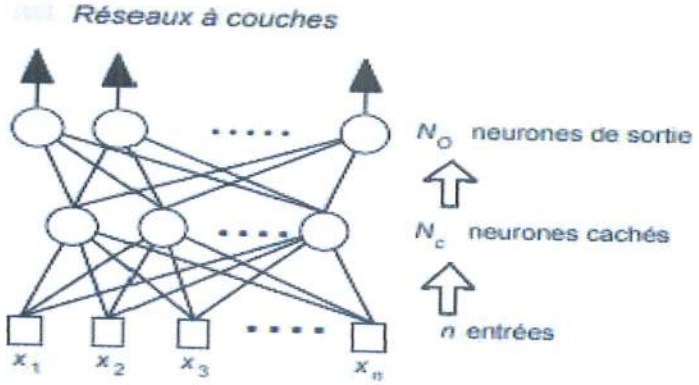


Figure I-11 : un réseau de neurone à n entrées, une couche de Nc neurones cachés, et No neurones de sortie.

7.2 Réseaux de neurones bouclés

Un réseau de neurones bouclés à temps discret réalise une (ou plusieurs) équations aux différences non linéaires, par composition des fonctions réalisées par chacun des neurones et des retards associés à chacune des connexions.

Les « réseaux bouclés », dont le graphe des connexions est cyclique : lorsqu'on se déplace dans le réseau en suivant le sens des connexions, il est possible de trouver au moins un chemin qui revient à son point de départ (un tel chemin est désigné sous le terme de « cycle »). La sortie d'un neurone du réseau peut donc être fonction d'elle même ; cela n'est évidemment concevable que si la notion de temps est explicitement prise en considération. Ainsi, à chaque connexion d'un réseau de neurones bouclé (ou à chaque arête de son graphe) est attaché, outre un poids comme pour les réseaux non bouclés, un retard, multiple entier (éventuellement nul) de l'unité de temps choisie. Une grandeur, à un instant donné, ne pouvant pas être fonction de sa propre valeur au même instant, tout cycle du graphe du réseau doit avoir un retard non nul.

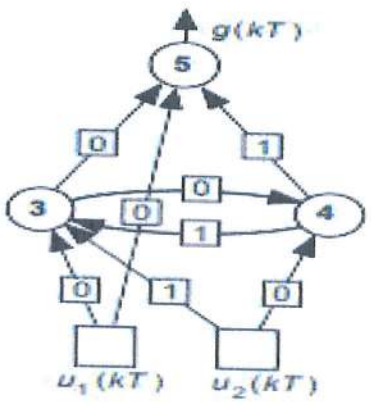


Figure I-12 : un réseau de neurones bouclé à deux entrées.

Les chiffres dans les carrés indiquent le retard attaché à chaque connexion, multiple de l'unité de temps (ou période d'échantillonnage) T. Le réseau contient un cycle, qui part du neurone 3, va au neurone 4, et revient au neurone 3. **[7]**

8 *Domaines d'application*

Dans ces dernières années les réseaux de neurones sont utilisés dans plusieurs applications de divers domaines tels que :

- **Traitement d'images** : Reconnaissance de caractères et de signatures, reconnaissance de formes …
- **Traitement du signal :** Traitement de la parole …
- **Classification d'espèces animales** étant donnée une analyse ADN.
- **Contrôle** : Diagnostic de pannes, robotique, …

- **Optimisation** : Allocation de ressources, planification, régulation de trafic, gestion, finance,...
- **La médecine** : Diagnostic médical. **[9]**

9 Historique

Les réseaux de neurones artificiels depuis leur création jusqu'à nos jours sont passés par différentes étapes de développements on les montre dans le table au-dessus :

1890	**W. James**	**Introduit le concept de mémoire associative.**
1942	**Norbert Wiener**	**Boucle de rétroaction cybernétique.**
1943	**McCulloch et Pitts**	**Modèle formel de neurone biologique.**
1949	**Hebb**	**Explique le conditionnement chez l'animal par les propriétés des neurones eux-mêmes. Loi pouvant expliquer le phénomène d'apprentissage.**
1957	**Frank Rosenblatt**	**Perceptron, premier neuro-ordinateur, première tentative de reproduire le cerveau de l'homme.**
1960	**B. Widrow**	**Développe le modèle Adaline à l'origine de l'algorithme de rétropropagation de gradient très utilisé aujourd'hui avec les Perceptrons multicouches.**
1969	**Minsky et Papert**	**Démontrent qu'une classe de problème ne peut pas être résolue par le perceptron. Ils donnent les limitations théoriques du perceptron et de la neuronique. Les financements se dirigent les systèmes experts.**

1982	**John Hopfield**	**On lui doit le renouveau d'intérêt pour les réseaux de neurones artificiels: perceptron multicouches.**
1983	**Boltzmann**	**La Machine de Boltzmann contourne les limitations du perceptron.**
1986	**Werbos puis Rumelhart, Hinton et Williams**	**Algorithme d'apprentissage applicable au perceptron de type récursif (back propagation). La classe de problème de Minsky et Papert peut être résolue.**

Tableau I-1 : Les différentes étapes de dévloppement des réseaux de neurones.

Aujourd'hui, les réseaux de neurones artificiels sont largement reconnus et utilisés dans la recherche et dans l'industrie. **[10]**

10 Conclusion

Nous avons présenté d'une manière globale les réseaux de neurones et leurs types, qui sont inspirés du comportement du cerveau humain, qui permettent à une machine de devenir intelligente. Ils ont aujourd'hui un impact considérable, et que leur importance ira en grandissant dans le futur.

CHAPITRE II : LES RESEAUX DE NEURONES CONVOLUTIONNELS

1 Introduction

Après le grand succès des réseaux de neurones artificiels, leur application dans plusieurs domaines, et leur grande capacité à résoudre des problèmes non linéaires et complexes plusieurs type sont apparus et sont entrés en compétition pour prouver lequel est le mieux adapté pour résoudre tel ou tel type de problèmes.

Parmi les types des réseaux de neurones le perceptron multi-couches ou le MLP[1], qui est composé de plusieurs couches cachées et une couche de sortie, qui est un réseau de rétro-propagation (*feedforward*) et qui permet de résoudre des problèmes complexes.

Les réseaux de neurone convolutionnels ou CNN[2] sont une extension du perceptron multi couches.

Ils sont spécialisés dans le traitement d'images et l'extraction des caractéristiques. Ils comprennent cinq types de couches : la couche de convolution, la couche qui utilise une fonction d'activation, la couche de regroupement (pooling), la couche de perte (LOSS) et en dernier les couches entièrement connectés (fully connected).

2 Le Deep Learning

L'apprentissage profond (en anglais deep learning) est un ensemble de méthodes d'apprentissage automatique tentant de modéliser avec un haut niveau d'abstraction des données grâce à des architectures articulées sur différentes transformations non linéaires. Ces techniques ont permis des progrès importants et rapides dans les domaines de l'analyse du signal sonore ou visuel et notamment de la reconnaissance faciale, de la reconnaissance vocale, de la vision par ordinateur.**[11]**

2.1 Quelques algorithmes du Deep Learning

Il existe différents algorithmes de Deep Learning. Nous pouvons ainsi citer :

- **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 »).

  On dit qu'il y a un **sur-apprentissage** si à chaque itération, on minimise l'erreur sur la base d'apprentissage mais on augmente l'erreur sur la base d'essai. Le modèle perd sa capacité de généralisation : **c'est l'apprentissage par cœur. [8]**

---

[1] MLP : MultiLayer Perceptron
[2] CNN : Convolutional Neural Network

- **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 créé 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, suivie de l'entraînement classique supervisé. Cette étape d'apprentissage non-supervisée, permet, en outre, de faciliter l'apprentissage supervisé. **[5]**

3 *Les réseaux de neurones convolutionnels*

Les réseaux de neurones convolutionnels sont à ce jour les modèles les plus performants pour classer des images. Désignés par l'acronyme CNN, de l'anglais Convolutional Neural Network, ils comportent deux parties bien distinctes. En entrée, une image est fournie sous la forme d'une matrice de pixels. Elle a 2 dimensions pour une image en niveaux de gris. La couleur est représentée par une troisième dimension, de profondeur 3 pour représenter les couleurs fondamentales [Rouge, Vert, Bleu].

La première partie d'un réseau de neurones convolutionnels est la partie convolutive à proprement parler. Elle fonctionne comme un extracteur de caractéristiques des images. Une image est passée à travers une succession de filtres, ou noyaux de convolution, créant de nouvelles images appelées cartes de convolutions. Certains filtres intermédiaires réduisent la résolution de l'image par une opération de maximum local. Au final, les cartes de convolutions sont mises à plat et concaténées en un vecteur de caractéristiques, appelé code CNN.

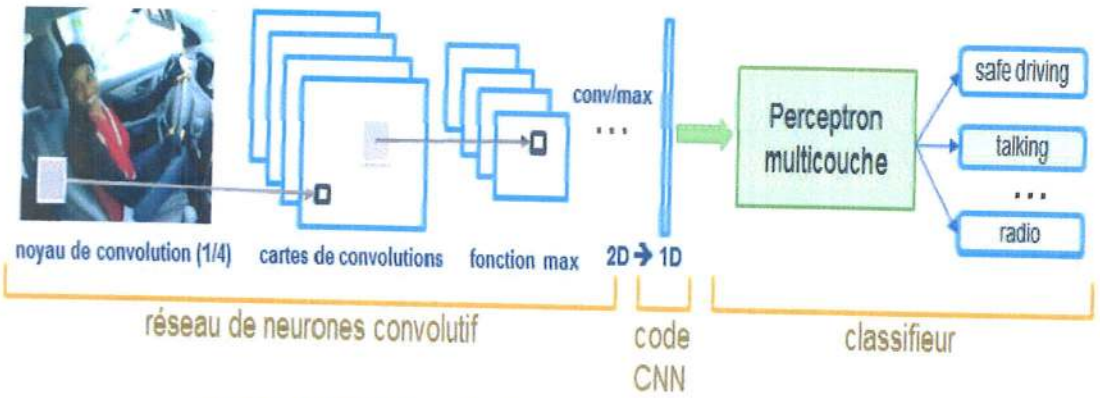


Figure II-1 : Architecture d'un réseau de neurone convolutionnel.

Ce code CNN en sortie de la partie convolutive est ensuite branché en entrée d'une deuxième partie, constituée de couches entièrement connectées (perceptron multicouche). Le rôle de cette partie est de combiner les caractéristiques du code CNN pour classer l'image.

La sortie est une dernière couche comportant un neurone par catégorie. Les valeurs numériques obtenues sont généralement normalisées entre 0 et 1, de somme 1, pour produire une distribution de probabilité sur les catégories. **[12]**

3.1 Architecture du réseau de neurones convolutionnels

Les réseaux de neurones convolutionnels sont basés sur le perceptron multicouche (MLP), et inspirés du comportement du cortex visuel des vertébrés, sont très efficaces pour le traitement d'images, les MLP ont des difficultés à gérer des images de grande taille, dû à la croissance exponentielle du nombre de connexions avec la taille de l'image.

Par exemple, si on prend une image de taille 32x32x3 (32 pixels de large, 32 pixels de haut, 3 canaux de couleur), un seul neurone entièrement connecté dans la première couche cachée du MLP aurait 3072 entrées (32x32x3). Une image 200x200 conduirait ainsi à traiter 120 000 entrées par neurone ce qui, multiplié par le nombre de neurones, devient énorme.

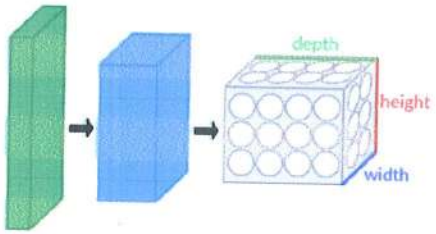


Figure II-2 : Une couche du CNN en 3 dimensions (vert = volume d'entrée, bleu = volume du champ récepteur, gris = couche de CNN, cercles = neurones artificiels indépendants).

Les réseaux de neurones convolutifs visent à limiter le nombre d'entrées tout en conservant la forte corrélation « spatialement locale » des images naturelles. Par opposition aux MLP, les CNN ont les propriétés suivantes :

1. ***Connectivité locale*** : grâce au champ récepteur qui limite le nombre d'entrées du neurone, tout en conservant l'architecture MLP (cf. point précédent), les réseaux de neurones convolutifs assurent ainsi que les « filtres » produisent la réponse la plus forte à un motif d'entrée spatialement localisé, ce qui conduit à une représentation parcimonieuse de l'entrée. Une telle représentation occupe moins d'espace en mémoire. De plus, le nombre de paramètres à estimer étant réduit, leur estimation (statistique) est plus robuste pour un volume de données fixé (comparé à un MLP).
2. ***Poids partagés*** : dans les réseaux de neurones convolutifs, les paramètres de filtrage d'un neurone (pour un champ récepteur donné) sont identiques pour tous les autres neurones d'un même noyau (traitant tous les autres champs récepteurs de l'image). Ce paramétrage (vecteur de poids et biais) est défini dans une « carte de fonction ».
3. ***Invariance à la translation*** : comme tous les neurones d'un même noyau (filtre) sont identiques, le motif détecté par ce noyau est indépendant de localisation spatiale dans l'image

Ensemble, ces propriétés permettent aux réseaux de neurones à convolution d'obtenir une meilleure robustesse dans l'estimation des paramètres sur des problèmes d'apprentissage puisque, pour une taille de corpus d'apprentissage fixée, la quantité de données par paramètres est plus grande. Le partage de poids permet aussi de réduire considérablement le nombre de paramètres

libres à apprendre, et ainsi les besoins en mémoire pour le fonctionnement du réseau. La diminution de l'empreinte mémoire permet l'apprentissage de réseaux plus grands donc souvent plus puissants.

Une architecture de réseau de neurones convolutifs est formée par un empilement de couches de traitement :

- la couche de convolution (CONV) qui traite les données d'un champ récepteur ;
- la couche de regroupement ou 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). **[13]**

3.2 Couche de convolution (CONV)

La couche de convolution est le bloc de construction de base d'un réseau de neurones convolutionnels.

Trois hyperparamètres permettent de dimensionner le volume de la couche de convolution (aussi appelé volume de sortie) : la profondeur, le pas et la marge.

1. Profondeur de la couche : nombre de noyaux de convolution (ou nombre de neurones associés à un même champ récepteur).
2. Le pas : contrôle le chevauchement des champs récepteurs. Plus le pas est petit, plus les champs récepteurs se chevauchent et plus le volume de sortie sera grand.
3. La marge (à 0) ou 'zero padding' : parfois, il est commode de mettre des zéros à la frontière du volume d'entrée. La taille de ce 'zero-padding' est le troisième hyperparamètre. Cette marge permet de contrôler la dimension spatiale du volume de sortie. En particulier, il est parfois souhaitable de conserver la même surface que celle du volume d'entrée.

Si le pas et la marge appliquée à l'image d'entrée permettent de contrôler le nombre de champs récepteurs à gérer (surface de traitement), la profondeur permet d'avoir une notion de volume de sortie, et de la même manière qu'une image peut avoir un volume, si on prend une profondeur de 3 pour les trois canaux RGB d'une image couleur, la couche de convolution va également présenter en sortie une profondeur.

C'est pour cela que l'on parle plutôt de "volume de sortie" et de "volume d'entrée", car l'entrée d'une couche de convolution peut être soit une image soit la sortie d'une autre couche de convolution.

La taille spatiale du volume de sortie peut être calculée en fonction de la taille du volume d'entrée W_i la surface de traitement **K** (nombre de champs récepteurs), le pas **S** avec lequel ils sont appliqués, et la taille de la marge **P**. La formule pour calculer le nombre de neurones du volume de sortie est :

$W_0 = \frac{Wi-K+2P}{S} + 1$. Si W_0 n'est pas entier, les neurones périphériques n'auront pas autant d'entrée que les autres. Il faudra donc augmenter la taille de la marge (pour recréer des entrées virtuelles).

Souvent, on considère un pas S=1, on calcule donc la marge de la manière suivante : $P = \frac{K-1}{2}$ si on souhaite un volume de sortie de même taille que le volume d'entrée. Dans ce cas particulier la couche est dite "connectée localement". **[13]**

3.3 Couche de regroupement ou Pooling (POOL)

Un autre concept important des réseaux de neurones convolutionnels est le regroupement (pooling). C'est une forme de sous-échantillonnage de l'image. L'image d'entrée est découpée en une série de rectangles de n pixels de côté ne se chevauchant pas (pooling). Chaque rectangle peut être vu comme une tuile. Le signal en sortie de tuile est défini en fonction des valeurs prises par les différents pixels de la tuile.

Le regroupement (pooling) réduit la taille spatiale d'une image intermédiaire, réduisant ainsi la quantité de paramètres et de calcul dans le réseau. Il est donc fréquent d'insérer périodiquement une couche de regroupement (pooling) entre deux couches convolutives successives d'une architecture de réseau de neurones convolutifs pour réduire le sur-apprentissage. L'opération de regroupement (pooling) créait aussi une forme d'invariance par translation.

La couche de regroupement (pooling) fonctionne indépendamment sur chaque tranche de profondeur de l'entrée et la redimensionne uniquement au niveau de la surface. La forme la plus courante est une couche de mise en commun avec des tuiles de taille 2 × 2 (largeur/hauteur) et comme valeur de sortie la valeur maximale en entrée. On parle dans ce cas de « Max-Pool 2x2 »

Il est possible d'utiliser d'autres fonctions de regroupement (pooling) que le maximum. On peut utiliser un « average pooling » (la sortie est la moyenne des valeurs du patch d'entrée), du « L2-norm pooling ». Dans les faits, même si initialement l'average pooling était souvent utilisé il s'est avéré que le max-pooling était plus efficace car celui-ci augmente plus significativement l'importance des activations fortes. En d'autres circonstances, on pourra utiliser un pooling stochastique.

Le regroupement (pooling) permet de gros gains en puissance de calcul. Cependant, en raison de la réduction agressive de la taille de la représentation (et donc de la perte d'information associée), la tendance actuelle est d'utiliser de petits filtres (type 2×2). Il est aussi possible d'éviter la couche de pooling mais cela implique un risque de sur-apprentissage plus important. **[13]**

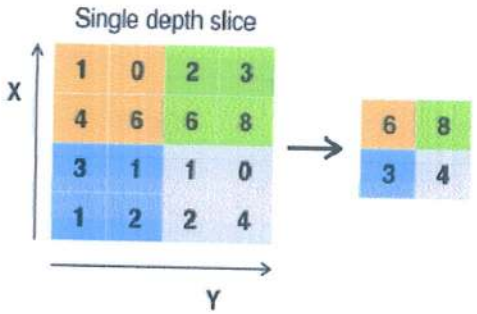


Figure II-3 : Max pooling avec un filtre 2x2 et un pas de 2.

3.4 Couche de correction (ReLU)

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.

La fonction ReLU[3] (abréviation de Unité Rectifié Linéaire) : F(x) = max(0, x). Cette fonction force les neurones à retourner des valeurs positives. **[13]**

3.5 Couche entièrement connectée (FC)

Après plusieurs couches de convolution et de max-pooling, le raisonnement de haut niveau 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.**[13]**

3.6 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 fonction «Softmax » permet de calculer la distribution de probabilités sur les classes de sortie. **[13]**

3.7 Exemple de modèle de réseau de neurones convolutionnels

La forme la plus commune d'une architecture de CNN 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 CNN 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 (au lieu d'un grand filtre) permet un traitement plus puissant, avec moins de paramètres. Cependant, avec l'inconvénient de demander plus de puissance de calcul (pour contenir tous les résultats intermédiaires de la couche CONV). **[13]**

[3] Rectified Linear Unit

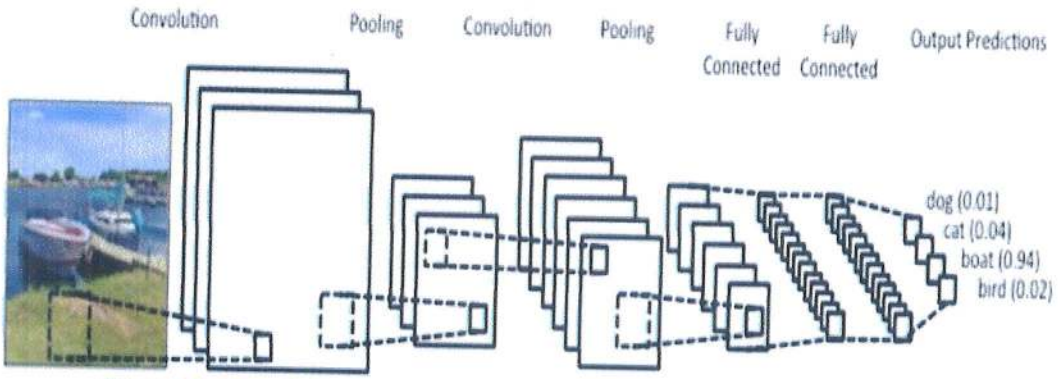


Figure II-4 : Exemple d'un modèle de réseau de neurones convolutionnels (CNN).

3.8 Choix des paramètres

Les réseaux de neurones convolutifs utilisent plus de paramètres qu'un MLP 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, leurs formes et la forme du max pooling.

3.8.1 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. **[13]**

3.8.2 Forme de 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[4] (28 x 28) sont habituellement dans la gamme de 5 × 5 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 12 × 12, voire 15 × 15.

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. **[13]**

[4] MNIST : Une base de données d'images représentant des chiffres manuscrits

3.8.3 Forme du Max Pooling

Les valeurs typiques sont 2 × 2. De très grands volumes d'entrée peuvent justifier un pooling 4 × 4 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. **[13]**

3.8.4 Méthodes de régularisation

Pour ne pas tomber dans le problème de sur apprentissage, il y a des méthodes de régularisation à utiliser. Elles sont de deux types : empirique ou explicite.

3.8.5 Empirique

3.8.5.1 Dropout

Les couches entièrement connectées (FC – Fully Connected) occupent la majeure partie de la mémoire du réseau de neurones convolutionnels. D'ailleurs le concept de FC crée un problème exponentiel de mémoire appelé "overfitting" ("sur-connexion" conduisant au sur-apprentissage) ralentissant le traitement de l'information. Pour prévenir cela, la méthode du dropout est utilisée pour "éteindre" les neurones aléatoirement (avec une probabilité prédéfinie, souvent un neurone sur deux) ainsi que les neurones périphériques. Ainsi, avec moins de neurones, le réseau est plus réactif et peut donc apprendre plus rapidement. À la fin de la séance d'apprentissage, les neurones "éteints" sont "rallumés" (avec leurs poids originaux). Plus la couche FC est proche de l'image source, moins on éteindra de neurones.

L'objectif est d'éteindre et rallumer les neurones aléatoirement, dans le cadre d'entraînements successifs. Une fois les séries d'entraînements terminées, on rallume tous les neurones et on utilise le réseau comme d'habitude. Cette technique a montré non seulement un gain dans la vitesse d'apprentissage, mais en déconnectant les neurones, on a aussi limité des effets marginaux, rendant le réseau plus robuste et capable de mieux généraliser les concepts appris. **[13]**

3.8.5.2 DropConnect

Le DropConnect est une évolution du dropout, où on ne va non plus éteindre un neurone, mais une connexion (l'équivalent de la synapse), et ce de manière toujours aléatoire. Les résultats sont similaires (rapidité, capacité de généralisation de l'apprentissage), mais présentent une différence au niveau de l'évolution des poids des connexions. Une couche FC avec un DropConnect peut s'apparenter à une couche à connexion "diffuse". **[13]**

3.8.5.3 Pooling stochastique

Le pooling stochastique reprend le même principe que le Max-pooling, mais la sortie choisie sera prise au hasard, selon une distribution multinomiale définie en fonction de l'activité de la zone adressée par le pool. Dans les faits, ce système s'apparente à faire du Max-pooling avec un grand nombre d'images similaires, qui ne varient que par des déformations localisées. On peut aussi considérer cette méthode comme une adaptation à des déformations élastiques de l'image. C'est pourquoi cette méthode est très efficace sur les images MNIST (base de données d'images

représentant des chiffres manuscrits). La force du pooling stochastique est de voir ses performances croître de manière exponentielle avec le nombre de couches du réseau. **[13]**

3.8.6 Explicite

3.8.6.1 Taille du réseau

La manière la plus simple de limiter le sur-apprentissage est de limiter le nombre de couches du réseau et de libérer les paramètres libres (connexions) du réseau. Ceci réduit directement la puissance et le potentiel prédictif du réseau. C'est équivalent à avoir une "norme zéro". **[13]**

3.8.6.2 Dégradation des poids

Le concept est de considérer le vecteur des poids d'un neurone (liste des poids associés aux signaux entrants), et de lui rajouter un vecteur d'erreur proportionnel à la somme des poids (norme 1) ou du carré des poids (norme 2 ou euclidienne). Ce vecteur d'erreur peut ensuite être multiplié par un coefficient de proportionnalité que l'on va augmenter pour pénaliser davantage les vecteurs de poids forts.

- La régularisation par norme 1 : La spécificité de cette régulation est de diminuer le poids des entrées aléatoires et faibles et d'augmenter le poids des entrées "importantes". Le système devient moins sensible au bruit.
- La régularisation par norme 2 : (norme euclidienne) La spécificité de cette régulation est de diminuer le poids des entrées fortes, et de forcer le neurone à plus prendre en compte les entrées de poids faible.
- Les régularisations par norme 1 et norme 2 peuvent être combinées : c'est la "régularisation de réseau élastique" (Elastic net regulation). **[13]**

4 Conclusion

Dans ce chapitre, nous avons présenté les réseaux de neurones convolutionnels. Ces réseaux sont simplement un empilement de plusieurs couches de convolution, de la fonction d'activation (ReLU), de regroupement (pooling), et d'entièrement connectée (fully connected). Ils sont capables d'extraire des caractéristiques d'images données, et de les classifier.

Les réseaux de neurones convolutionnels commencent par une couche de convolution et se termine par une couche entièrement connectée, et des couches intermédiaires peuvent s'empiler de différente manières. A condition que la sortie d'une couche est l'entrée de la suivante.

Les couches de convolution et de regroupement (pooling) possèdent des hyper-paramètres qui sont difficiles a évaluer a priori : la taille de filtre (receptive fields), le pas (stride) , la marge (padding), le nombre de canaux, pour la convolution et un stride et une taille de filtre pour le regroupement (pooling).

CHAPITRE III : LES MODELES DE RESEAUX DE NEURONES CONVOLUTIONNELS

1. Introduction

Après le lancement du modèle LeNet de Yan Lecun, plusieurs modèles sont apparus et sont utilisés dans le domaine de l'apprentissage profond.

Dans ce chapitre nous allons présenter les modèles de réseaux de neurones les plus connus en montrant les différences entres eux.

2. Les modèles de réseau de neurones les plus connus

Il existe plusieurs architectures dans le domaine des réseaux de neurones convolutifs, les plus connus sont : LeNet, AlexNet, ZFNet, GoogLeNet, VGGNet et ResNet.

2.1 LeNet

C'est le premier réseau de neurones à convolution dans le domaine de l'apprentissage profond réussi en 1994 par Yan LECUN après plusieurs itérations depuis 1988, et utilisé pour lire les chiffres manuscrits sur des chèques postaux.

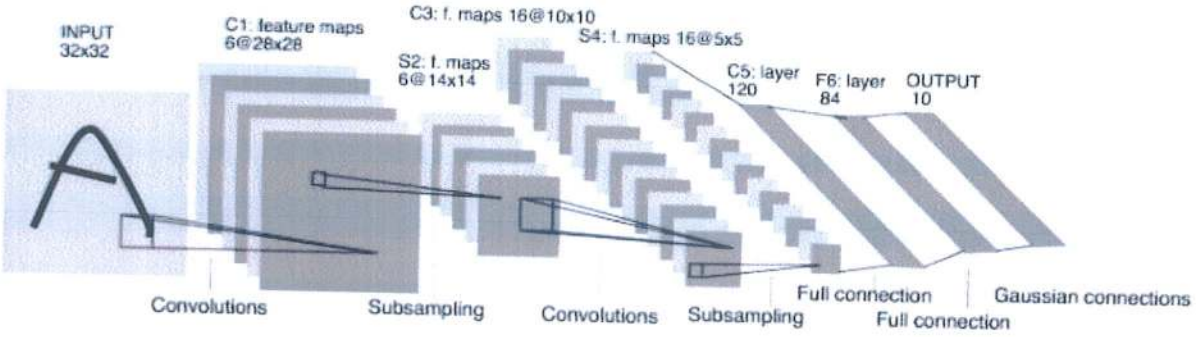


Figure III-1 : architecture du modèle LeNet5

Les fonctionnalités de LeNet5 peuvent être résumées comme suit :

- réseau neuronal convolutionnel séquence de 3 couches : convolution, regroupement, non-linéarité -> Cela peut être la caractéristique clé de Deep Learning pour les images.
- utiliser la convolution pour extraire des entités spatiales.
- sous-échantillon en utilisant la moyenne spatiale des cartes.
- non-linéarité sous forme de ***tanh*** ou de ***sigmoids.***
- réseau neuronal multicouche (MLP) comme classificateur final.
- matrice de connexion éparse entre les couches pour éviter un coût de calcul important.

Ce réseau été une véritable source d'inspiration pour de nombreux chercheurs dans ce domaine. **[14]**

2.2 AlexNet

En 2012, Le premier travail qui a popularisé les réseaux convolutionnels dans la vision par ordinateur a été AlexNet, développé par Alex Krizhevsky, Ilya Sutskever et Geoff Hinton. AlexNet a été soumis au concours ILSVRC[5] en 2012 et a nettement surperformé le deuxième finaliste. Le réseau avait une architecture très similaire à celle de LeNet, mais il était plus profond, plus grand et doté de couches convolutives empilées les unes sur les autres. **[15]**

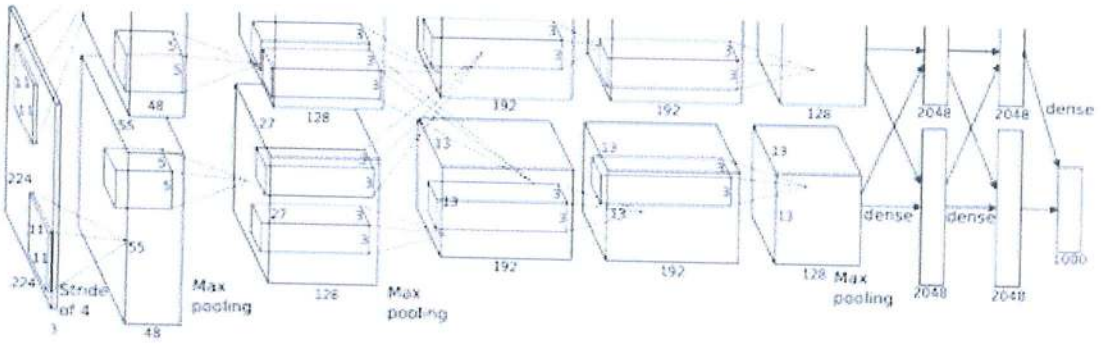


Figure III-2 : architecture du modèle AlexNet

- utilisation d'unités linéaires rectifiées (ReLU) en tant que non-linéarités.
- utilisation de la technique de l'abandon (DropOut) pour ignorer de manière sélective les neurones individuellement pendant l'entrainement ; une manière d'éviter un ajustement excessif des modèles.
- une superposition du *'Max Pooling'*, évitant les effets de moyenne du *'average pooling'*.
- utilisation des GPU NVIDIA GTX 580 pour réduire le temps d'entrainement.

Le succès d'Alexnet a déclenché une révolution pour les réseaux de neurones à convolution qui sont devenus le nouveau nom des les réseaux neuronaux pour résoudre plusieurs taches. **[14]**

2.3 ZFNet

Le gagnant de l'ILSVRC 2013 était un réseau convolutionnel de Matthew Zeiler et Rob Fergus. Il est devenu connu sous le nom de ZFNet (abréviation de Zeiler & Fergus Net). C'était une amélioration de AlexNet en modifiant les hyperparamètres de l'architecture, en particulier en augmentant la taille des couches de convolution au milieu et en réduisant la taille du pas (*stride*) et des filtres sur la première couche. **[15]**

[5] ImageNet Large Scale Visual Recognition Challenge

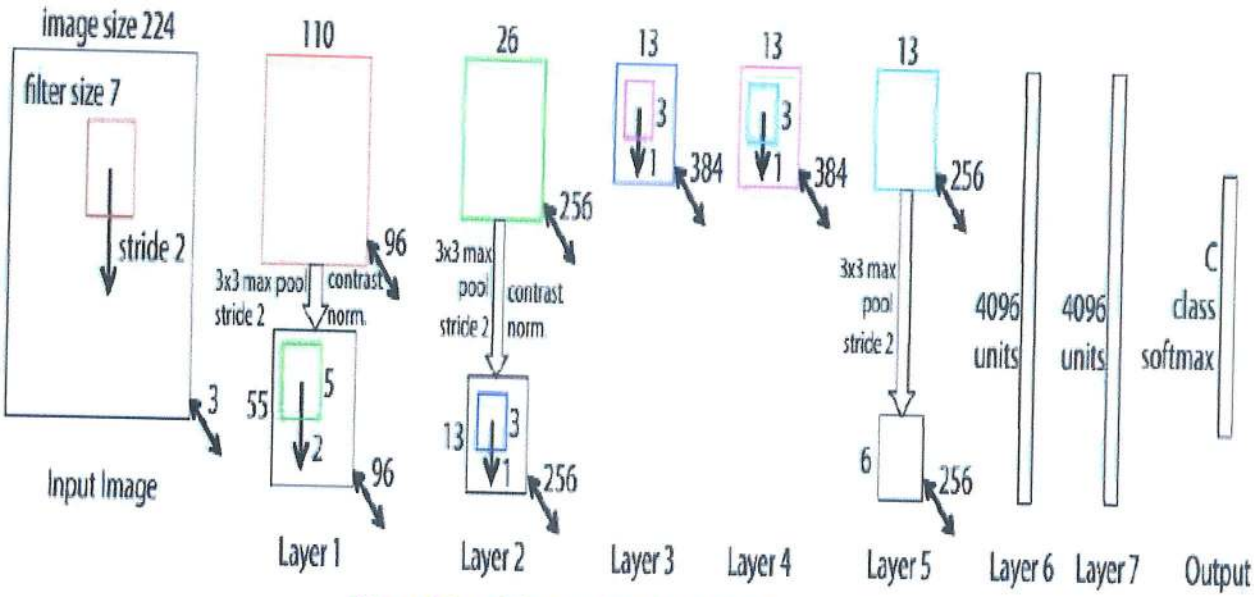


Figure III-3 : architecture du modèle ZFNet

2.4 GoogLeNet

Le gagnant de l'ILSVRC 2014 était un réseau convolutionnel de Szegedy et al.de Google. Sa principale contribution a été le développement d'un module de lancement (*Inception Module)* qui a considérablement réduit le nombre de paramètres dans le réseau (4M, comparé à AlexNet avec 60M). En outre, ce modèle utilise le '*Average Pooling*' au lieu des couches entièrement connectées en haut du ConvNet, éliminant une grande quantité de paramètres qui semblent peu importants. Il existe également plusieurs versions basées sur GoogLeNet, la plus récente est Inception-v4. **[15]**

Szegedy et son équipe ont proposé le module Inception :

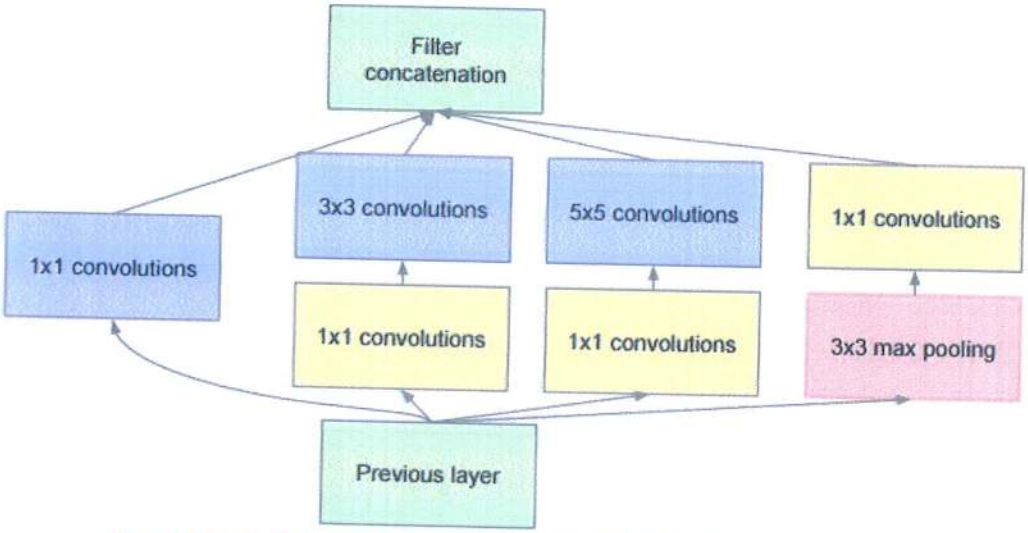


Figure III-4 : Architecture du module inception utilisé dans GoogLeNet

2.5 VGGNet

La deuxième place dans ILSVRC 2014 était le réseau de Karen Simonyan et Andrew Zisserman, connu sous le nom de VGGNet. Sa principale contribution a été de montrer que la profondeur du

réseau est un composant essentiel pour de bonnes performances. Leur meilleur réseau final contient 16 couches CONV / FC.

Les réseaux VGG d'Oxford ont été les premiers à utiliser des filtres 3x3 beaucoup plus petits dans chaque couche convolutionnelle et à les combiner en une suite de convolutions.

Cela semble être contraire aux principes de LeNet, où de grandes convolutions ont été utilisées pour capturer des caractéristiques similaires dans une image. Au lieu des filtres 9x9 ou 11x11 d'AlexNet, les filtres ont commencé à devenir plus petits, trop près des fameuses convolutions 1x1 que LeNet voulait éviter, du moins sur les premières couches du réseau. **[14]**

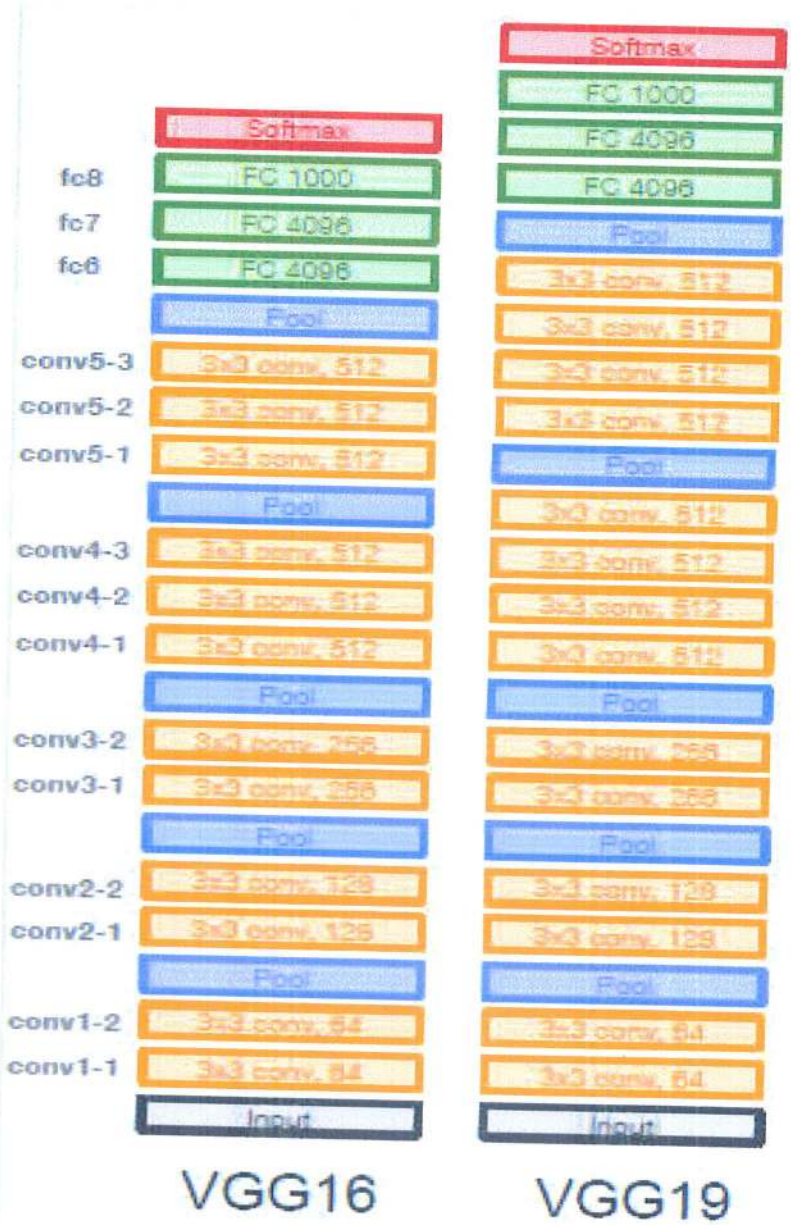


Figure III-5 : Architecture du modèle VGGNet les deux versions de 16 couches et de 19 couches

2.6 ResNet

Resnet (***Residual Network***) développé par Kaiming He et al. a été le gagnant d'ILSVRC 2015. Il propose des connexions spéciales et un usage intensif de la normalisation des lots. L'architecture

manque également de couches entièrement connectées à la fin du réseau. ResNets sont actuellement des modèles de réseaux neuronaux convolutifs de pointe et constituent le choix par défaut pour l'utilisation de ConvNets dans la pratique. **[15]**

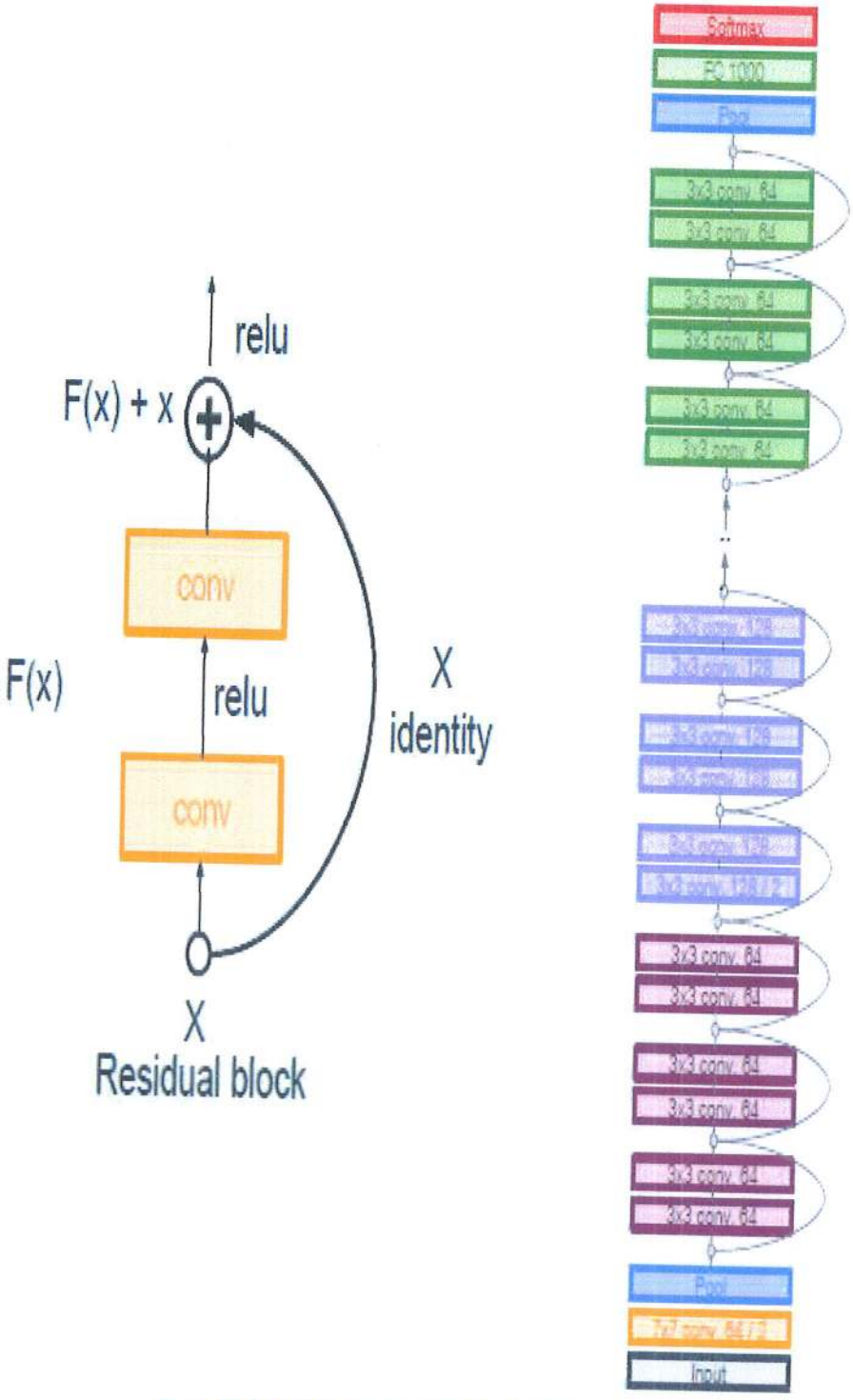


Figure III-6 : Architecture du modèle ResNet

- ResNet peut être vu comme des modules parallèles et en série, en pensant à l'inout comme allant à de nombreux modules en parallèle, tandis que la sortie de chaque module se connecte en série.
- ResNet peut également être considéré comme plusieurs ensembles de modules parallèles ou en série.

- Il a été constaté que ResNet opère généralement sur des blocs de profondeur relativement faible ~20-30 couches, qui agissent en parallèle plutôt que de parcourir en série toute la longueur du réseau.
- ResNet, lorsque la sortie est renvoyée à l'entrée, comme dans RNN, le réseau peut être considéré comme un meilleur modèle bio-plausible du cortex. **[14]**

3. Comparaison entre les modèles

Modèle	Taille de filtre	Stride	Padding	La taille de filtre de MaxPoolin	Nombres de couches
LeNet	5x5	1	1	Filtre 2x2 Stride 2	Conv : 2 couches Pool : 2 couches FC : 2 couches
AlexNet	11x11 5x5 3x3	4 1 1	0 2 1	Filtre 3x3 Stride 2	Conv : 5 couches Pool : 3 couches Norm : 2 couches FC : 3 couches
ZFNet	7x7 5x5 3x3	2 1 1	0 2 1	Filtre 3x3 Stride 2	Conv : 5 couches Pool : 3 couches Norm : 2 couches FC : 3 couches
GoogleNet	1x1 3x3 5x5	—	—	Filtre 3x3	22 couches Non FC
VGGNet	3x3	1	1	Filtre 2x2 Stride 2	Conv : 12 couches Pool : 5 couches FC : 3 couches avec softmax
ResNet	3x3	2	—	Filtre 2x2	152 couches Non FC

Tableau III-1 : Tableau comparatif des modèles de RN

Pour une comparaison plus approfondie on présente la figure 7 faite par Alfredo Canziani et Adam Paszke et Eugenio Culurciello, qui classifient les architectures de réseaux neuronaux populaires selon la précision par rapport à la quantité d'opérations requises pour un seul passage direct. **[16]**

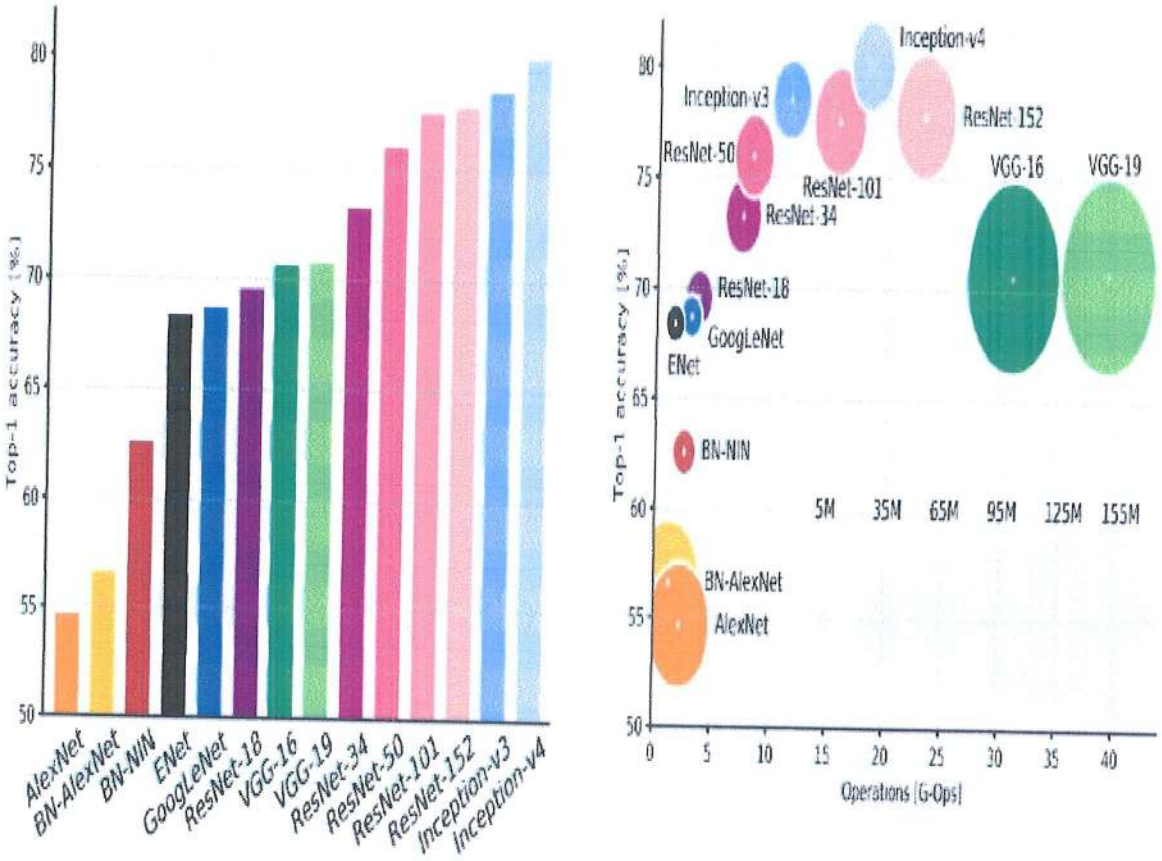


Figure III-7 : Comparaison des réseaux de neurones les plus connus.

4. Conclusion

Dans cette partie, nous avons présenté les modèles de réseaux de neurones les plus populaires.

D'après cette comparaison on a choisi le modèle VGGNet connu par sa profondeur et la taille fixe de ses filtres. En effet VGGNet est composé de couches de convolution qui ont une taille de filtre de 3x3 pour toutes les couches et utilise un stride de 1 et un padding de 1.

L'utilisation d'un filtre de petite taille permet l'extraction de plus de caractéristiques qu'un filtre de grande taille et cela permet un taux de classification correcte plus élevé.

CHAPITRE IV : IMPLEMENTATION ET RESULTATS

1 Introduction

Dans ce chapitre, nous allons présenter le déroulement du travail réalisé et nous discutons les résultats obtenus. Nous avons proposé deux modèles pour faire l'entrainement et la classification d'images : le premier est basé sur le modèle VGGNet qui est connu par sa profondeur (plusieurs couches), le deuxième modèle que nous avons créé est un réseau de neurones convolutionnel avec moins de couches que le premier et cela afin de pouvoir comparer leurs performances.

Les deux modèles sont appliqués sur la même base d'images que nous avons créée à partir de la grande base d'images ImageNet.

L'implémentation est faite avec Torch7 à l'aide de bibliothèques telles que « nn » pour la création d'un réseau de neurones, « optim » pour l'entrainement, et d'autres bibliothèques. Nous avons aussi proposé d'utiliser le « Dropout » pour améliorer la performance du modèle.

2 Les frameworks similaires à Torch7

2.1 TensorFlow

TensorFlow est une bibliothèque logicielle open source pour le calcul numérique haute performance. Son architecture flexible permet un déploiement facile du calcul sur diverses plates-formes (processeurs, processeurs graphiques, TPU[6]), et des ordinateurs de bureau aux clusters de serveurs, aux périphériques mobiles.

Initialement développé par des chercheurs et des ingénieurs de l'équipe de Google Brain au sein de l'organisation de l'IA de Google, il s'appuie sur l'apprentissage automatique et l'apprentissage en profondeur et donc les réseaux de neurones. **[17]**

2.2 Keras

Keras est une API de réseaux de neurones de haut niveau, écrite en Python et capable de fonctionner sur TensorFlow CNTK ou Theano. Il a été développé dans le but de permettre une expérimentation rapide. Être en mesure de passer de l'idée au résultat le plus rapidement possible est la clé pour faire de la recherche.

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 de Google.

En 2017, l'équipe TensorFlow de Google a décidé de soutenir Keras dans la bibliothèque principale de TensorFlow. **[18]**

---

[6] Tensor Processing Unit

2.3 Theano

Theano est un projet open source lancé sous la licence BSD et développé par le groupe LISA (aujourd'hui MILA) à l'Université de Montréal, Québec, Canada.

Theano est un compilateur pour les expressions mathématiques en Python. Il sait comment prendre vos structures et les transformer en code très efficace qui utilise NumPy.

Il a été spécifiquement conçu pour gérer les types de calcul requis pour les grands algorithmes de réseaux de neurones utilisés dans Deep Learning. Il s'agit de l'une des premières bibliothèques du genre (le développement a débuté en 2007) et est considéré comme un standard industriel pour la recherche et le développement Deep Learning. **[19]**

3 Pourquoi Torch

Le but de Torch est d'avoir un maximum de flexibilité et de rapidité dans la construction d'algorithmes scientifiques tout en rendant le processus extrêmement simple. Torch est livré avec plusieurs modules axés sur l'apprentissage automatique, la vision par ordinateur, le traitement du signal, le traitement parallèle, l'image, la vidéo, l'audio entre autres, et s'appuie sur la communauté Lua.

Au cœur de Torch, se trouve le populaire réseau de neurones et les bibliothèques d'optimisation qui sont simples à utiliser, tout en ayant une flexibilité maximale dans l'implémentation de topologies de réseaux neuronaux complexes. **[20]**

4 Configuration matérielle utilisée dans l'implémentation

Nous avons travaillé avec le logiciel « Oracle VM VirtualBox » installé sur un ordinateur portable TOSHIBA PORTéGé X30 avec la configuration matérielle suivante :

- Un processeur Intel i5-7200U CPU 2.70 GHZ
- Carte graphique Intel® HD Graphics 620
- RAM de taille 8 GO
- Disque dur de 240 GO

Sur cet ordinateur nous avons installé une machine virtuelle Linux Ubuntu version 16.04 avec une RAM de 3Go et un disque dur de 10 GO.

5 La base d'images

Nous avons créé notre base d'images à partir de la base de données ImageNet que nous avons téléchargée. Parmi les classes qu'ImageNet propose, nous avons choisi 10 classes qui sont les suivantes : « animal, arbre, avion, bateau, bus, camion, personne, train, vélo, voiture ».

Nous avons redimensionné les images pour qu'elles aient la même taille de 48x48 pixels et aussi pour ne pas prendre beaucoup d'espace. Nous avons utilisé 20000 images pour l'entrainement (dans

chaque classe il y'a 2000 images) et 10000 images pour la validation (dans chaque classe il y'a 1000 images).

6 L'architecture du modèle VGG

Il est créé par Karen Simonyan et Andrew Zisserman, en 2014. Ce modèle qui est le deuxième finaliste de ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

Son principal but est de montrer que la profondeur du réseau est essentielle pour une bonne performance. **[15]**

La figure ci-dessous présente l'architecture du modèle VGG qu'on a utilisé comme il s'affiche sous torch :

```
nn.Sequential {
  [input -> (1) -> (2) -> output]
  (1): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9) -> (10) -> (11) -> (12) ->
    (13) -> (14) -> (15) -> (16) -> (17) -> (18) -> (19) -> (20) -> (21) -> output]
    (1): nn.SpatialConvolution(3 -> 32, 3x3, 1,1, 1,1)
    (2): nn.SpatialBatchNormalization (4D) (32)
    (3): nn.ReLU
    (4): nn.SpatialConvolution(32 -> 32, 3x3, 1,1, 1,1)
    (5): nn.SpatialBatchNormalization (4D) (32)
    (6): nn.ReLU
    (7): nn.SpatialMaxPooling(2x2, 2,2)
    (8): nn.SpatialConvolution(32 -> 64, 3x3, 1,1, 1,1)
    (9): nn.SpatialBatchNormalization (4D) (64)
    (10): nn.ReLU
    (11): nn.SpatialConvolution(64 -> 64, 3x3, 1,1, 1,1)
    (12): nn.SpatialBatchNormalization (4D) (64)
    (13): nn.ReLU
    (14): nn.SpatialMaxPooling(2x2, 2,2)
    (15): nn.SpatialConvolution(64 -> 128, 3x3, 1,1, 1,1)
    (16): nn.SpatialBatchNormalization (4D) (128)
    (17): nn.ReLU
    (18): nn.SpatialConvolution(128 -> 128, 3x3, 1,1, 1,1)
    (19): nn.SpatialBatchNormalization (4D) (128)
    (20): nn.ReLU
    (21): nn.SpatialMaxPooling(2x2, 2,2)
  }
  (2): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> output]
    (1): nn.View(4608)
    (2): nn.Linear(4608 -> 512)
    (3): nn.ReLU
    (4): nn.BatchNormalization (2D) (512)
    (5): nn.Dropout(0.500000)
    (6): nn.Linear(512 -> 10)
    (7): nn.LogSoftMax
  }
}
```


Figure IV-1 : Affichage de l'architecture du premier modèle sous torch

La structure de l'architecture de ce modèle est composée de six couches de convolution et trois couches de Maxpooling et deux couches entièrement connectée.

L'image en entrée est de taille 48x48 en couleur(RGB). Elle passe par la première couche de convolution dotée de 32 filtres de taille 3x3 avec un stride et un padding de 1. Elle est suivie d'une normalisation de lot et d'une fonction d'activation ReLU qui force les neurones à retourner des valeurs positives, après cette étape les 32 features maps seront créées de taille 46x46 et paramètres égale à 864.

Les 32 features maps de la couche précédente deviennent les données d'entrées de la deuxième couche de convolution qui est composée de 32 filtres de taille 3x3 avec un stride et un padding de 1 et comme la couche précédente on applique le batch Normalisation (normalisation de lot) et ReLU. A la sortie nous aurons 32 feature maps de taille 46 avec 9216 paramètres.

En suite, on applique le Max Pooling avec un filtre de 2x2 et un stride de 2 pour réduire la taille de l'image et le nombre de paramètres. A la sortie de cette couche nous aurons une taille de 23x23 et 0 paramètres.

On répète la même chose pour la troisième couche de convolution, la sortie de la deuxième couche de convolution devient les données d'entrée de cette couche qui est composée de 64 filtres de taille 3x3 avec stride et padding de 1. A la sortie nous aurons 64 feature maps de taille 23x23 et 18432 paramètres.

Dans la quatrième couche de convolution, on a 64 filtres de taille 3x3 avec stride et padding est 1.a la sortie on a 64 featur maps de taille 23x23 et 36864 paramètres.

En suite, on applique le Max pooling avec un filtre de taille 2x2 et stride de 2 ce qui donne en sortie la taille de 12x12 avec 0 paramètres.

On applique les mêmes opérations sur le reste des couches de convolution. Ces couches sont composées respectivement de 64 filtres, et on aura a la sortie 128 feature maps de taille 12x12 et 73728 paramètres, et 128 filtres et on aura à la sortie 128 feature maps de taille 12x12 et 147456 paramètres.

En fin en applique le Max pooling à la sortie nous aurons une taille de 6x6 et 0 paramètres.

Après les six couches de convolution, on utilise un réseau de neurones composé de deux couches entièrement connectées. La première est composée de 512 neurones et utilise Dropout, normalisation de lot et la fonction ReLU, la deuxième utilise la fonction SoftMax qui permet de calculer la distribution de probabilité des 10 classes de notre base d'images.

7 *L'architecture du deuxième modèle*

La structure de l'architecture de ce modèle que nous avons créé est composée de cinq couches de convolution et trois couches de Maxpooling et trois couches entièrement connectée. Ce modèle est affiché sous torch comme suit :

```
nn.Sequential {
  [input -> (1) -> output]
  (1): nn.Sequential {
    [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9) -> (10) -> (11) -> (12) ->
    (13) -> (14) -> (15) -> (16) -> (17) -> (18) -> (19) -> (20) -> (21) -> (22) -> output]
    (1): nn.SpatialConvolution(3 -> 32, 3x3, 1,1, 1,1)
    (2): nn.ReLU
    (3): nn.SpatialConvolution(32 -> 64, 3x3, 1,1, 1,1)
    (4): nn.ReLU
    (5): nn.SpatialMaxPooling(2x2, 2,2)
    (6): nn.SpatialConvolution(64 -> 64, 3x3, 1,1, 1,1)
    (7): nn.ReLU
    (8): nn.SpatialMaxPooling(2x2, 2,2)
    (9): nn.SpatialConvolution(64 -> 128, 3x3, 1,1, 1,1)
    (10): nn.ReLU
    (11): nn.SpatialConvolution(128 -> 128, 3x3, 1,1, 1,1)
    (12): nn.ReLU
    (13): nn.SpatialMaxPooling(2x2, 2,2)
    (14): nn.View(4608)
    (15): nn.Linear(4608 -> 512)
    (16): nn.Dropout(0.500000)
    (17): nn.ReLU
    (18): nn.Linear(512 -> 512)
    (19): nn.Dropout(0.500000)
    (20): nn.ReLU
    (21): nn.Linear(512 -> 10)
    (22): nn.LogSoftMax
  }
}
```

Figure IV-2: Affichage de l'architecture du deuxième modèle sous torch

L'image en entrée est de taille 48x48 en couleur(RGB). Elle passe par la première couche de convolution. Cette dernière de 32 filtres de taille 3x3 avec un stride et un padding de 1. Elle est suivie d'une fonction d'activation ReLU qui force les neurones à retourner des valeurs positives. Après cette étape les 32 features maps seront créés de taille 46x46 et paramètres égale à 864.

Les 32 features maps de la couche précédente devient les données d'entrées de la deuxième couche de convolution qui est composé de 32 filtres de taille 3x3 avec un stride et un padding de 1 et comme la couche précédente on applique la fonction ReLU. A la sortie nous aurons 64 feature maps de taille 46x46 avec 18432 paramètres.

Après on applique le Max Pooling avec un filtre de 2x2 et un stride de 2 pour réduire la taille de l'image et le nombre de paramètres. A la sortie de cette couche nous aurons une taille de 23x23 et 0 paramètres.

On répète les mêmes opérations pour la troisième couche de convolution, la sortie de la deuxième couche de convolution est devenu les données d'entrée de cette couche qui est composée de 64 filtres de taille 3x3 avec stride et padding de 1. A la sortie nous aurons 64 feature maps de taille 23x23 et 36864paramètres.

Après on applique le Max Pooling avec un filtre de 2x2 et un stride de 2. A la sortie de cette couche nous aurons une taille de 12x12 et 0 paramètres.

Dans la quatrième couche de convolution, on a 64 filtres de taille 3x3 avec stride et padding est 1.a la sortie on a 128 featur maps de taille 12x12 et 73728 paramètres.

Dans la cinquième couche de convolution, on a 128 filtres de taille 3x3 avec stride et padding est 1.a la sortie on a 128 featur maps de taille 12x12 et 147456 paramètres.

Après on applique le Max Pooling avec un filtre de 2x2 et un stride de 2. A la sortie de cette couche nous aurons une taille de 6x6 et 0 paramètres.

Après les cinq couches de convolution, on utilise trois couches entièrement connectées. Les deux premières couches ont chacune 512 neurones où la fonction d'activation utilisée est le ReLU, et on utilise aussi le Dropout, la troisième utilise la fonction SoftMax qui permet de calculer la distribution de probabilité des 10 classes de notre base d'images.

8 *Résultats obtenus et discussions*

Au début nous avons testé les deux modèles avec une base d'images qui contient 1000 images d'apprentissage et 150 images de validation mais elle n'a pas donné des résultats correcte, où ca précision est de 27,40 et le taux d'erreur est de 2,00, car elle est petite ne contient pas beaucoup d'images comme une base pour l'entrainement. Alors on a augmenté le nombre d'images jusqu'à 20000 images d'apprentissage et 10000 images de validation. Les résultats sont donnés en fonction du nombre époques.

Pour les deux modèles, les résultats obtenus sont donnés en termes de précision, d'erreur et de matrice de confusion.

8.1 Résultats obtenus du premier modèle

Le tableau suivant montre l'évolution du pourcentage de la précision de chaque classe en fonction du nombre d'époques.

Classes	Epoque 25	Epoque 35	Epoque 52	Epoque 100
animal	63.25%	63.65%	66.15%	65.65%
arbre	83.05%	83.75%	85.10%	84.95%
avion	67.05%	71.15%	73.25%	71.85%
bateau	70.50%	70.55%	72.70%	73.60%
bus	64.80%	65.30%	66.90%	66.35%
camion	62.05%	60.50%	62.75%	64.60%
personne	78.80%	79.60%	79.20%	80.95%
train	64.40%	64.60%	67.80%	65.05%

vélo	65.35%	67.30%	69.45%	69.35%
voiture	64.95%	67.85%	67.85%	66.35%

Tableau IV-1 : Taux de précision de chaque classe selon le nombre d'époques du 1er modèle.

La figure IV-3 illustre les résultats du premier modèle en terme de taux de précisions en fonction du nombre d'époques.

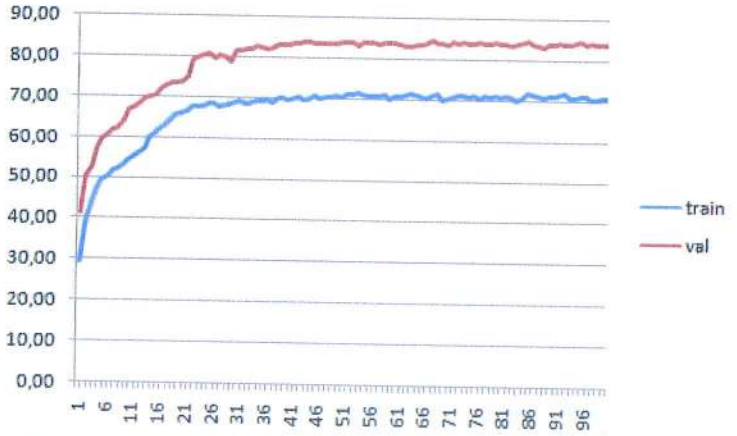


Figure IV-3 : Précision du premier modèle en fonction du nombre d'époques

La figure ci-dessous montre les taux d'erreurs du premier modèle en fonction du nombre d'époques.

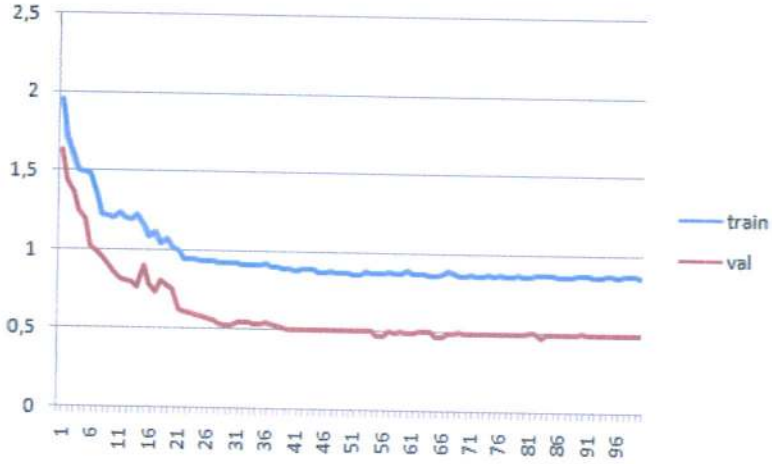


Figure IV-4 : L'erreur du premier modèle en fonction du nombre d'époques.

Discussions

D'après la figure IV-3, on remarque que la précision de l'apprentissage et de la validation augmente avec le nombre d'époques jusqu'à atteindre le taux de 70% pour l'entrainement et 80% pour la validation et cela à partir de 50 époques. Cela signifie que le modèle continue à apprendre plus d'informations à chaque époque jusqu'à 50 époques. Au-delà des 50 époques, le modèle se stabilise.

Parfois on remarque que la précision diminue, ce qui veut dire qu'on aura besoin de plus d'information ; alors automatiquement on doit augmenter le nombre d'époques.

D'après la figure IV-4, on remarque la même chose, pour l'erreur de l'apprentissage et de la validation qui diminue avec le nombre d'époques.

La figure IV-5 montre la matrice de confusion du premier modèle à l'époque 100. La matrice de confusion permet d'évaluer la performance du modèle en donnant la relation entre les différentes classes.

A la classe i sont associées la ligne i et la colonne i.

La ligne i contient la base d'entrainement de la classe i. Par exemple, la ligne 3 correspond à la base d'entrainement de la 3^{e} classe qui est « avion ». En faisant la somme, on retrouvera les 2000 images de la classe « avion ». Pour chaque colonne j, combien de fois la classe j était le résultat de classification du réseau de neurones en donnant une image de la classe i.

La colonne j contient les résultats de classification du réseau de neurones en partant de toutes les classes i.

Par exemple la colonne 3 correspond au nombre de fois où la classe « avion » était le résultat de classification du réseau de neurones pour toutes les classes.

Un pourcentage de classification correcte est fourni pour chaque classe (juste à gauche du nom de la classe).

```
ConfusionMatrix:
[[ 1313  130   83   78    7   29  206   29   88   37]  65.650%  [class: animal]
 [   93 1699   29   33    9   14   21   20   75    7]  84.950%  [class: arbre]
 [   88   20 1437  137   35   30   34   60   72   87]  71.850%  [class: avion]
 [   88   17  150 1472   46   36   23   72   38   58]  73.600%  [class: bateau]
 [    9   18   42   48 1327  204   20  173   39  120]  66.350%  [class: bus]
 [   49   36   68   55  191 1292   18  121   58  112]  64.600%  [class: camion]
 [  171   25   30   20    6   22 1619   30   60   17]  80.950%  [class: personne]
 [   55   22   58   83  177  122   49 1301   70   63]  65.050%  [class: train]
 [   95  116   71   57   46   51   70   59 1387   48]  69.350%  [class: velo]
 [   52   21  106   55  141  120   27   70   81 1327]]  66.350%  [class: voiture]
+ average row correct: 70.870000123978%
+ average rowUcol correct (VOC measure): 55.101175904274%
+ global correct: 70.87%
```

Figure IV-5 : La matrice de confusion du premier modèle à l'époque 100.

La figure IV-6 montre pour chaque classe le nombre d'images bien et mal-classées dans le premier modèle.

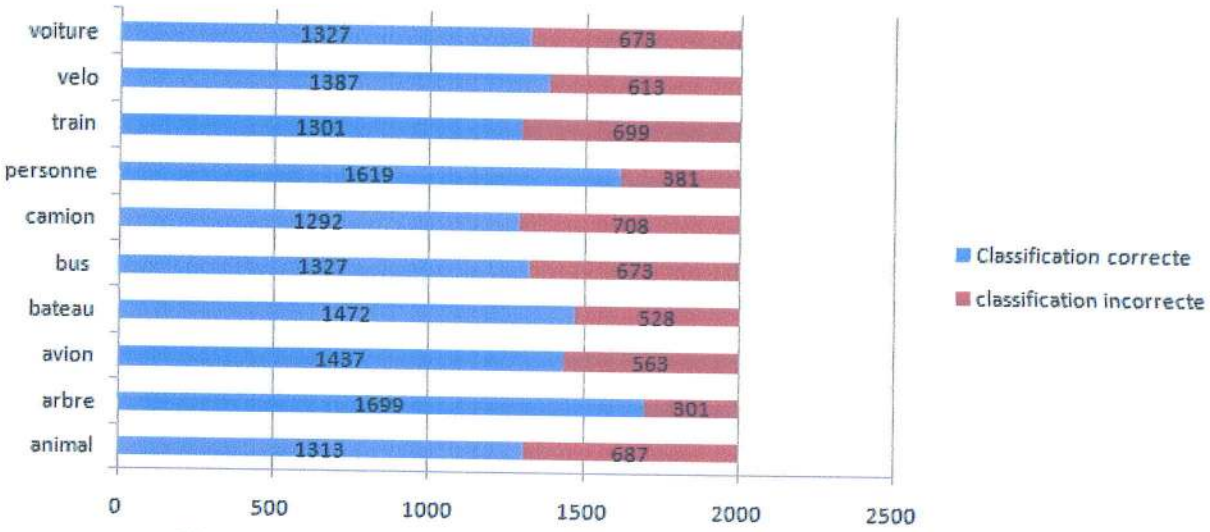


Figure IV-6 : Nombre d'images mal et bien classées dans le premier modèle.

Figure IV-7 : Secteur représentant la totalité des images classées dans le premier modèle.

8.2 Résultats obtenus du deuxième modèle

Le tableau suivant montre l'évolution du pourcentage de la précision de chaque classe en fonction du nombre d'époques.

Classes	Epoque 25	Epoque 35	Epoque 52	Epoque 100
animal	61.10%	63.30%	67.85%	65.65%
arbre	84.85%	85.70%	86.40%	86.05%
avion	66.50%	69.45%	69.90%	70.10%
bateau	64.25%	66.95%	69.95%	70.75%
Bus	63.95%	64.95%	69.45%	66.55%
camion	57.45%	59.85%	63.40%	63.10%

personne	77.60%	80.10%	79.65%	81.05%
train	63.20%	64.85%	66.75%	66.50%
vélo	64.25%	66.00%	69.30%	69.80%
voiture	64.35%	66.55%	69.15%	67.85%

Tableau IV-2 : Taux de précision de chaque classe selon le nombre d'époques du 2e modèle.

La figure IV-8 illustre les résultats du deuxième modèle en terme de taux de précisions en fonction du nombre d'époques.

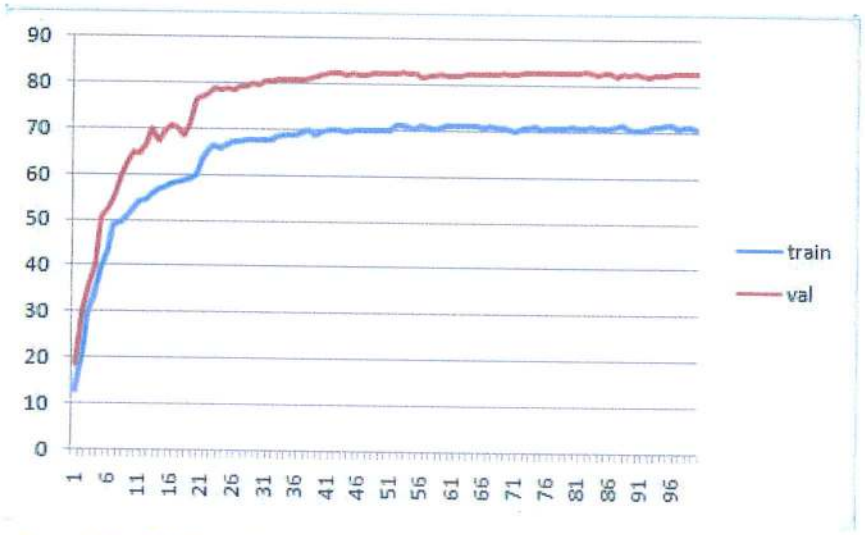


Figure IV-8 : Précision du deuxième modèle par rapport au nombre d'époques

La figure ci-dessous montre les taux d'erreurs du deuxième modèle en fonction du nombre d'époques.

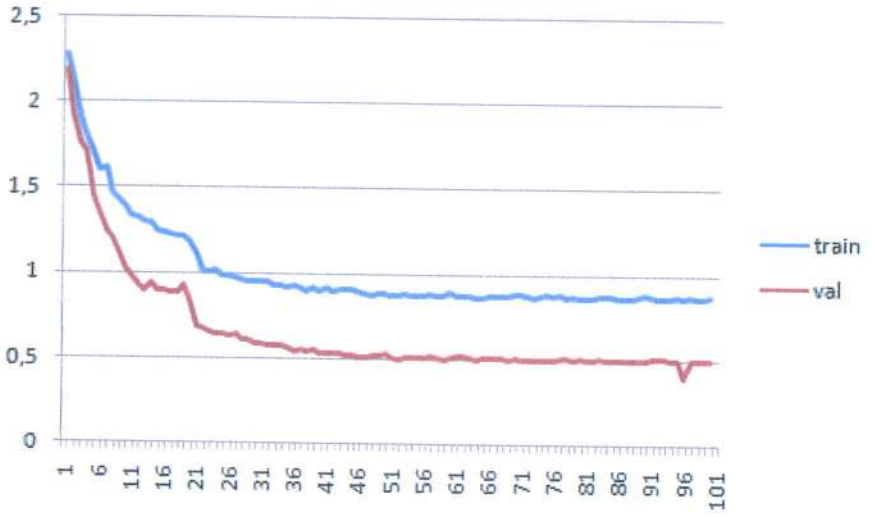


Figure IV-9 : L'erreur du deuxième modèle par rapport au nombre d'époques.

Discussions

D'après la figure IV-8 on remarque que la précision de l'apprentissage et de la validation augmente avec le nombre d'époques qui signifier que le modèle est entraient d'apprendre plus d'informations chaque époque. Parfois en remarque que la précision est diminuée qui veut dire que on aura besoin de plus d'information alors automatiquement en doit augmenter le nombre d'époque.

D'après la figure IV-9 on remarque la même chose, pour l'erreur de l'apprentissage et de la validation diminue avec le nombre d'époques

La figure IV-10 montre la matrice de confusion du deuxième modèle à l'époque 100.

```
ConfusionMatrix:
[[  1313   130   105    56     5    20   198    36   105    32]  65.650%  [class: animal]
 [    91  1721    19    24     5    19    14    16    80    11]  86.050%  [class: arbre]
 [    90    23  1402   179    19    49    37    50    69    82]  70.100%  [class: avion]
 [    73    33   180  1415    39    42    30    68    40    80]  70.750%  [class: bateau]
 [    15    23    42    59  1331   205    17   167    31   110]  66.550%  [class: bus]
 [    51    29    71    52   215  1262    24   110    62   124]  63.100%  [class: camion]
 [   184    15    34    17     5    15  1621    24    71    14]  81.050%  [class: personne]
 [    43    33    60    73   177   135    29  1330    66    54]  66.500%  [class: train]
 [   125   129    60    43    31    42    73    53  1396    48]  69.800%  [class: velo]
 [    58    24   101    72   119   122    19    37    91  1357]]  67.850%  [class: voiture]
 + average row correct: 70.73999941349%
 + average rowUcol correct (VOC measure): 54.945366084576%
 + global correct: 70.74%
```

Figure IV-10 : La matrice de confusion du deuxième modèle à l'époque 100.

La figure IV-11 montre pour chaque classe le nombre d'images bien et mal-classées dans le deuxième modèle.

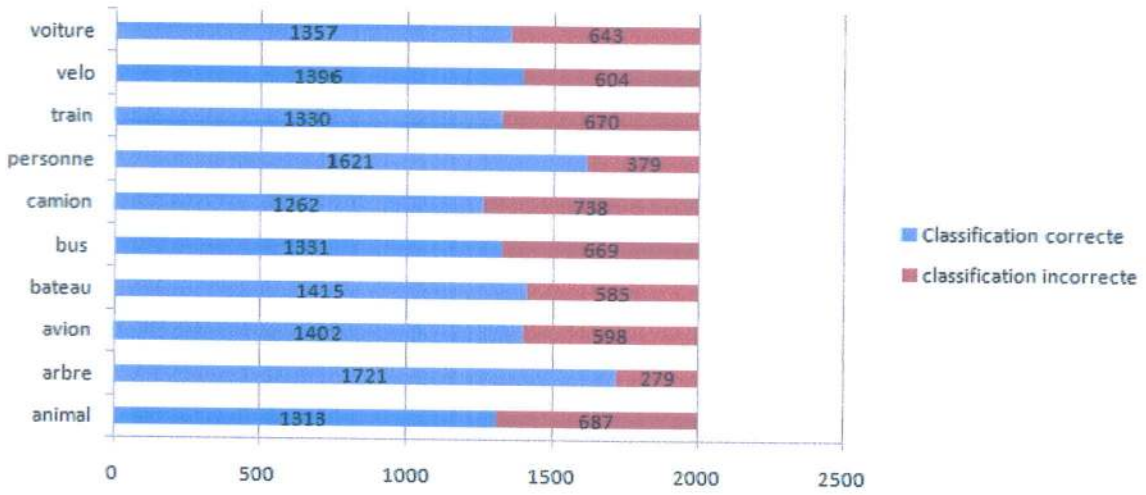


Figure IV-11 : Nombre d'images mal et bien classées dans le deuxième modèle.

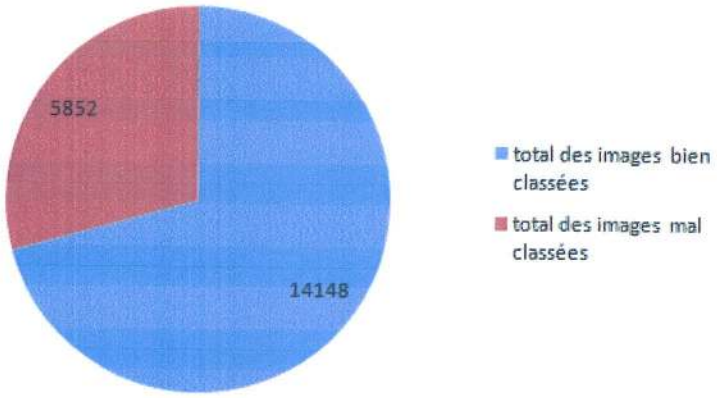


Figure IV-12 : Secteur représentant la totalité des images classées dans le deuxième modèle.

8.3 Tableau de comparaison des résultats

Le tableau ci dessous montre les différents résultats obtenus pour les deux modèles.

Modèle	Précision obtenue sur		Erreur obtenue sur		Temps d'exécution
	Apprentissage	Validation	Apprentissage	Validation	
1	70.87	84.02	0.86	0.49	32 heures et 30 min
2	70.74	82.92	0.87	0.50	30 heures

Tableau IV-3 : comparaison des résultats des deux modèles à l'époque 100.

Le tableau montre les résultats obtenus pour les deux modèles à l'époque 100. Le tableau donne les taux de précision et d'erreur des deux bases d'entrainement et de validation ainsi que le temps de calcul.

On remarque que le premier modèle donne un résultat légèrement meilleur par rapport au deuxième modèle mais avec un temps d'exécution plus grand ; cela signifie que le nombre d'époques et la profondeur des couches jouent un rôle important dans le temps de calcul et dans la performance des résultats du modèles où la suppression d'une couche influence sur les résultats du modèle et que plus le modèle est profond il donne les résultats sont mieux.

8.4 Affichage des filtres intermédiaires des couches du modèle N°2

Pour afficher les filtres intermédiaires, nous devons d'abord définir notre réseau de neurones convolutionnels qui est le notre deuxième modèle qu'on a crée et entraîner sur la base d'images de dix classes qu'on a créée.

Les étapes à suivre pour les afficher sont :

Comme une premier étape on prend une image et la redimensionner et la sauvegarder, on import le modèle qui est déjà entraîné et enregistré sous le nom « nv_best.h5 », et on lui demande classifier l'image. Le résultat est montré dans les deux figures suivantes :

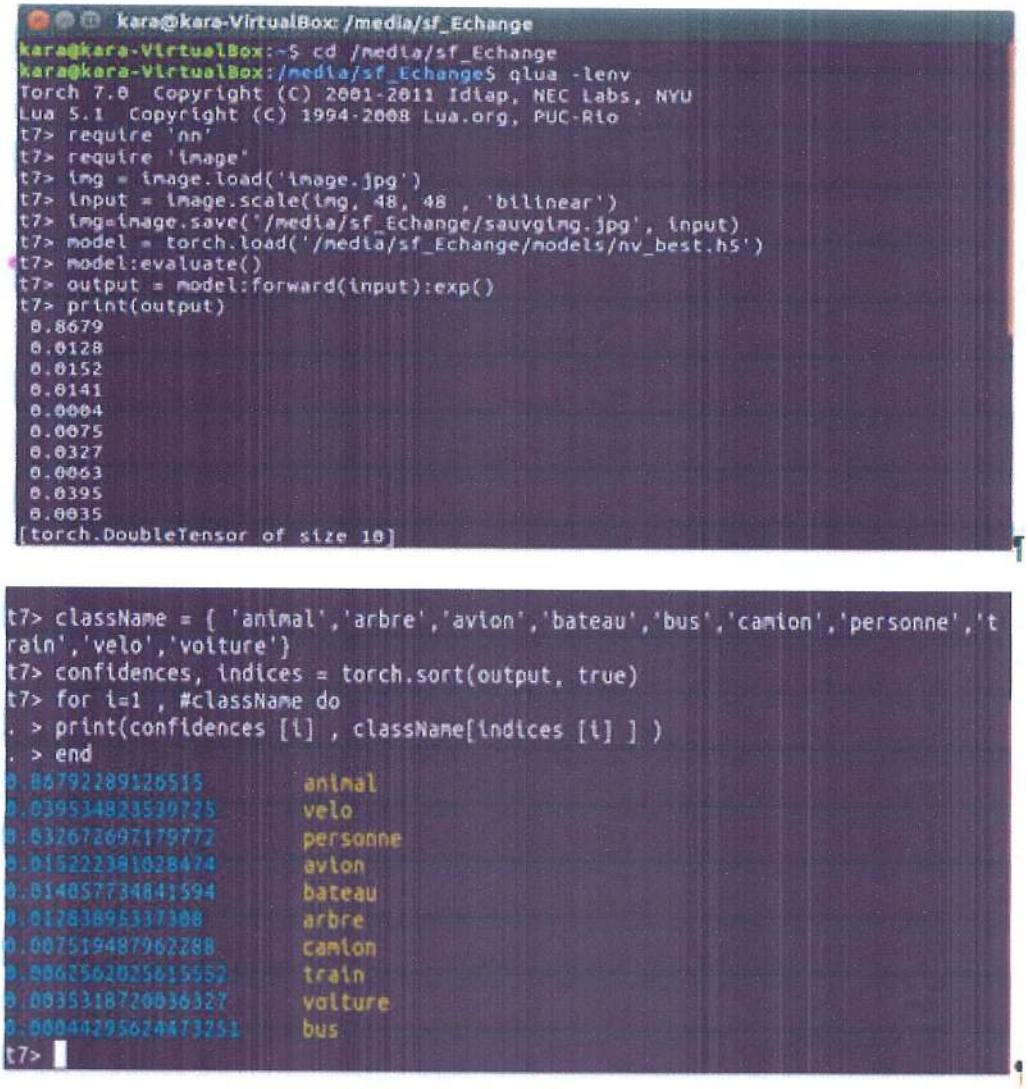


Figure IV-13 : Classification de l'image en entrée.

L'image est bien classée et appartient à la classe 'animal' avec une précision de 86.79%.

Pour afficher le modèle on utilise « print(model) », et pour afficher la première séquence du modèle on utilise « print(model:get(1)) ».

« print(n:get(1)) » pour afficher la première couche du modèle, et c'est une couche de convolution avec 32 filtres de taille 3x3 comme le montre la figure IV-14 :

```
kara@kara-VirtualBox: /media/sf_Echange
}
t7> n = model:get(1)
t7> print(n:get(1))
nn.SpatialConvolution(3 -> 32, 3x3, 1,1, 1,1)
{
  padW : 1
  nInputPlane : 3
  output : DoubleTensor - size: 32x48x48
  gradInput : DoubleTensor - size: 32x3x48x48
  _type : torch.DoubleTensor
  gradBias : DoubleTensor - size: 32
  dW : 1
  nOutputPlane : 32
  padH : 1
  kH : 3
  finput : DoubleTensor - size: 27x2304
  weight : DoubleTensor - size: 32x3x3x3
  train : false
  gradWeight : DoubleTensor - size: 32x3x3x3
  kW : 3
  dH : 1
  bias : DoubleTensor - size: 32
  fgradInput : DoubleTensor - size: 32x27x2304
}
```

Figure IV-14 : L'affichage de la première couche du modèle.

Pour faciliter l'affichage des filtres des autres couches on crée une fonction « show » et qui nous donne les résultats suivants :

Figure IV-15 : Les filtres de la première couche (à gauche) et l'image en entrée (à droite).

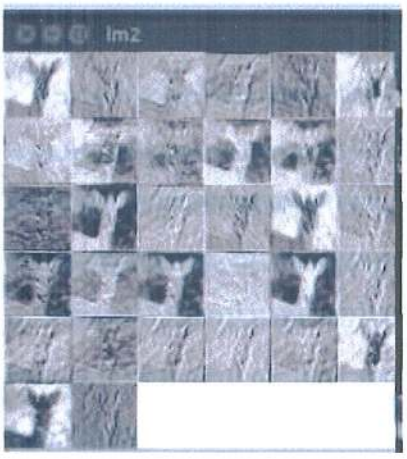


Figure IV-16 : Les 32 filtres de la première couche de convolution.

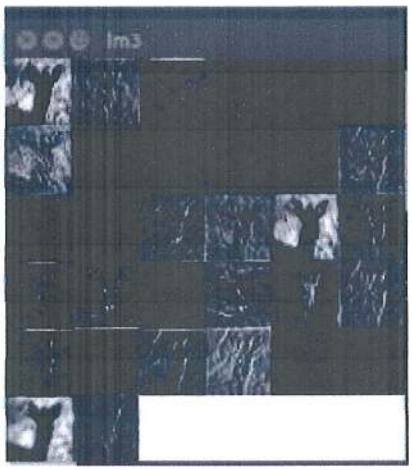


Figure IV-17 : Application de la fonction ReLU.

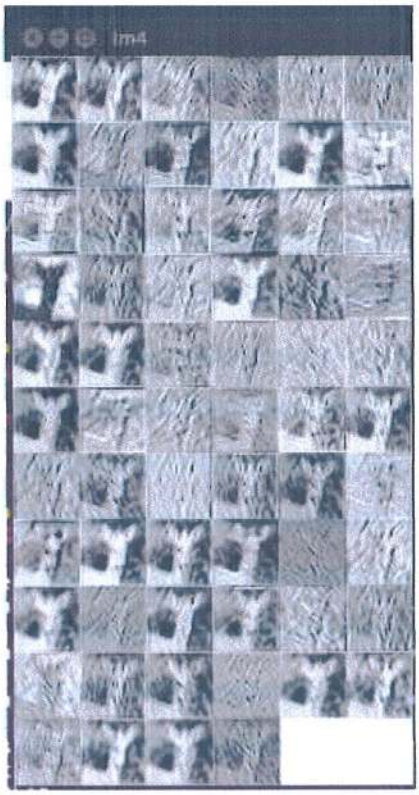


Figure IV-18 : Les 64 filtres de la deuxième couche de convolution.

Figure IV-19 : Application de la fonction ReLU.

Figure IV-20 : Application du MaxPooling.

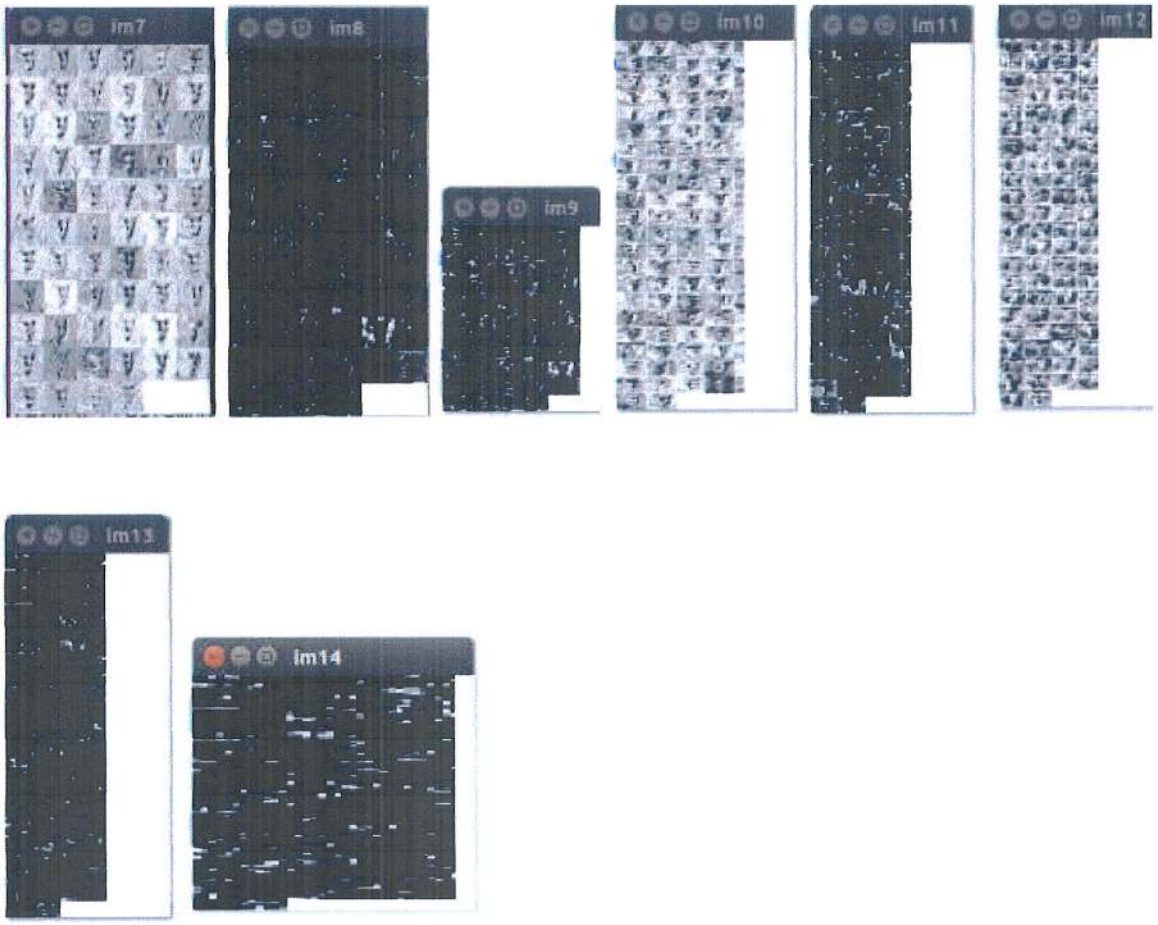


Figure IV-21 : Affichage des résultats des couches intermédiaires.

Dans la figure IV-21 : im7 représente les 64 filtres de la 3eme couche de convolution, im8 représente l'application de la fonction d'activation ReLU, im9 représente l'application du MaxPooling, im10 représente les 128 filtres de la 4eme couche de convolution, im11 représente l'application de la fonction de ReLU, im12 représente les 128 filtres de la 5eme couche de convolution, im13 représente l'application de la fonction de ReLu, im14 représente l'application du MaxPooling.

9 *Conclusion*

Dans ce chapitre nous avons présenté une classification d'images basée sur les réseaux de neurones convolutionnels, où nous avons testé deux modèles : l'un basé sur VGGNet et l'autre propre que nous avons créé.

Les résultats obtenus montrent que la base d'images, le nombre d'époques et la profondeur du réseau ont un impact sur la précision des résultats. Les résultats sont déterminés en fonction du taux de précision et d'erreur.

CONCLUSION GENERALE

Dans ce mémoire nous avons décrit des notions fondamentales des réseaux de neurones en général et les réseaux de neurones convolutionnels en particulier. Nous avons introduit les CNN qui sont structurés dans une architecture composée de deux parties : la première partie composée d'une succession de couches de convolutions, de fonction d'activation où nous avons choisi d'utiliser la fonction ReLU, et de couches de pooling, permettant l'extraction des informations de l'image en entrée afin de la classifier en fusionnant ces informations, la deuxième partie c'est la couche 'entièrement connectée' (fully connected). On a utilisé comme méthode de régularisation la méthode de Dropout pour éviter le problème du sur-apprentissage.

Nous avons utilisé deux modèles (réseaux de neurones) : le premier est basé sur le modèle VGGNet tandis que le deuxième est un modèle propre que nous avons créé. Cela est pour but de comparer les résultats des deux modèles et de tester l'influence du nombre de couches sur les performances. Ces performances sont définies en termes de précision et d'erreur de classification.

A partir de la grande base d'images ImageNet, nous avons créé une petite base de 1000 images pour l'entrainement et 150 images pour la validation. Les résultats obtenus étaient insuffisants.

Pour améliorer les résultats, nous avons créé une deuxième base plus importante avec 20000 images pour l'apprentissage et 10000 images pour la validation.

Les taux de précision (classification correcte) des deux modèles avoisinent les 70% pour l'entrainement et 80% pour la validation.

Les résultats obtenus ont montré que la suppression d'un nombre de couches permet de gagner en temps de calcul mais la précision du modèle se trouve diminuée et par conséquent le taux d'erreur augmente.

Ce travail permet de classer une image en entrée parmi un certains nombre de classes. Il peut être augmenté par une détection de plusieurs objets (classes d'objets) dans une même image en entrée.

ANNEXE A : INSTALLATION DE TORCH7 SOUS UBUNTU 16.02 (64 BITS)

1- Télécharger et installer ***Oracle VM Virtual Box*** à partir de :
https://download.virtualbox.org/virtualbox/5.2.16/VirtualBox-5.2.16-123759-Win.exe

2- Télécharger et installer ***VirtualBox Extension Pack*** à partir de :
https://download.virtualbox.org/virtualbox/5.2.16/Oracle_VM_VirtualBox_Extension_Pack-5.2.16.vbox-extpack

3- Télécharger la version souhaitée d'Ubuntu à partir de : http://releases.ubuntu.com/

Version 16.04.4 à partir de :

http://releases.ubuntu.com/xenial/ubuntu-16.04.4-desktop-amd64.iso (64 bits)

http://releases.ubuntu.com/xenial/ubuntu-16.04.4-desktop-i386.iso (32 bits)

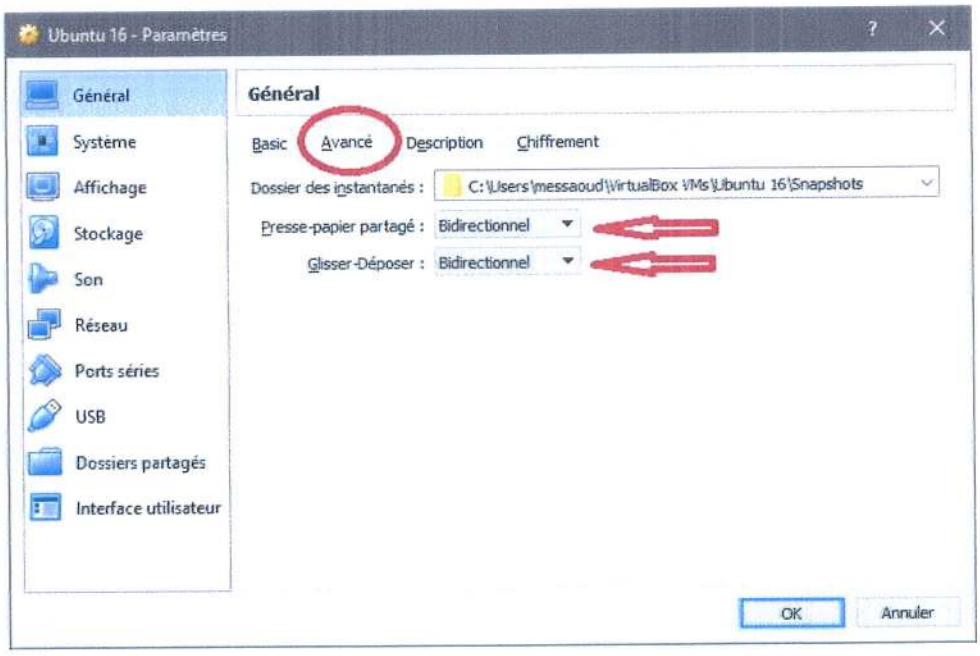
4- Installer Ubuntu dans Virtual Box

5- Activation du copier/coller entre la machine hôte (Windows) et la machine virtuelle (Ubuntu) :

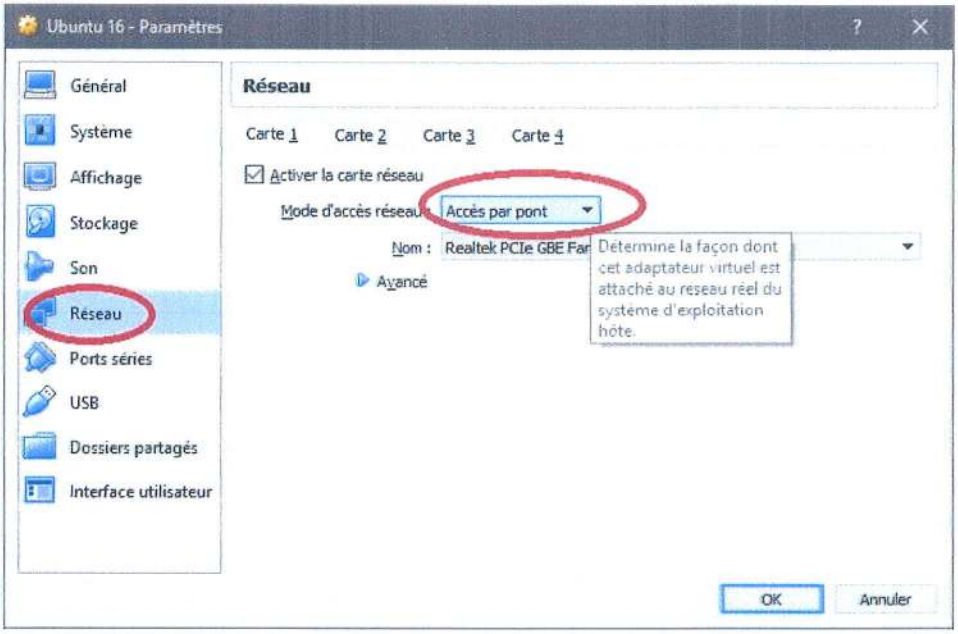
- Lancer Oracle VM Virtual Box

- Sélection la machine virtuelle Ubuntu

- Cliquer sur l'icone Configuration. La fenêtre des paramètres ci-dessous est lancée.

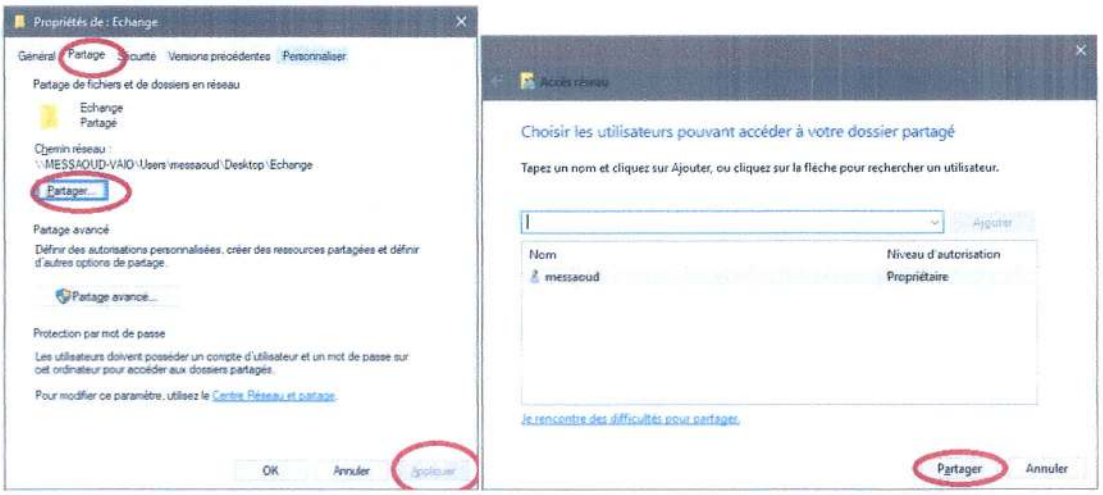


6- Activation de la carte réseau et choisir le mode qui permet d'avoir la connexion à Internet.

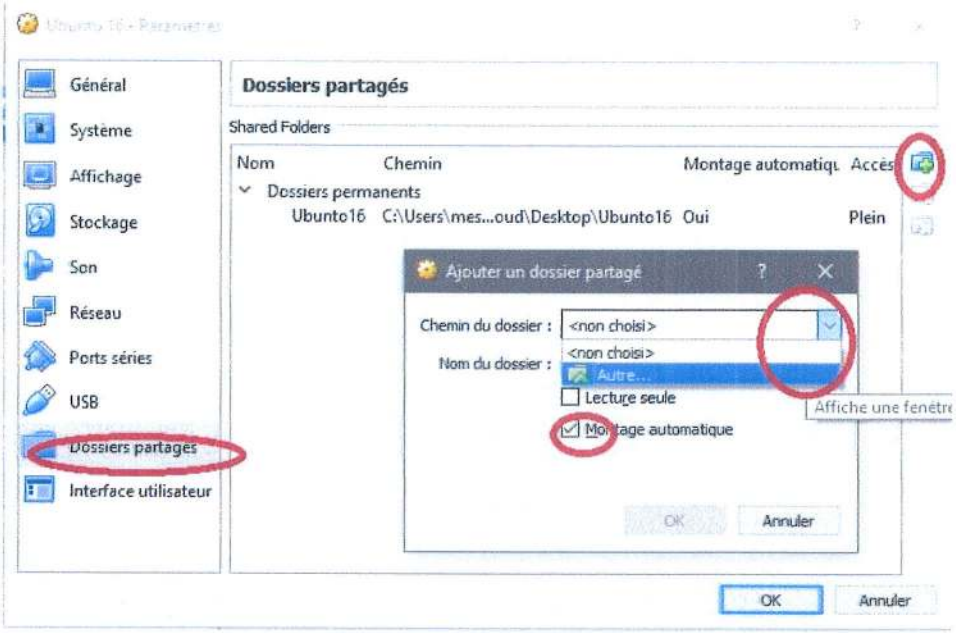


7- Créer un dossier partagé entre la machine hôte (Windows) et la machine virtuelle (Ubuntu) :

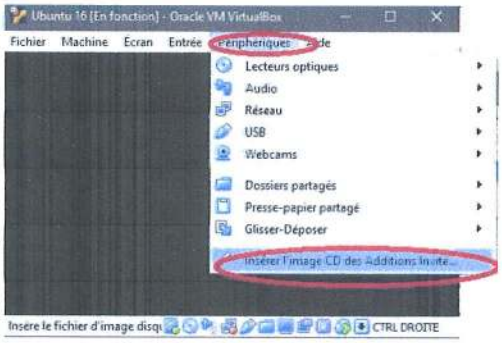
- Créer un dossier sous Windows, nommer le « Echange », par exemple. Clic droit dessus, propriétés, clic sur onglet partage, clic sur bouton partager. Dans la fenêtre qui s'ouvre cliquer sur le bouton partager. Fermer la fenêtre en cliquant sur le bouton appliquer.



- Dans les paramètres de la machine virtuelle ajouter le dossier partagé



- Lancer la machine virtuelle Ubuntu, insérer l'image CD des Additions invité…



- Après que l'installation soit terminée, redémarrer la machine virtuelle Ubuntu.

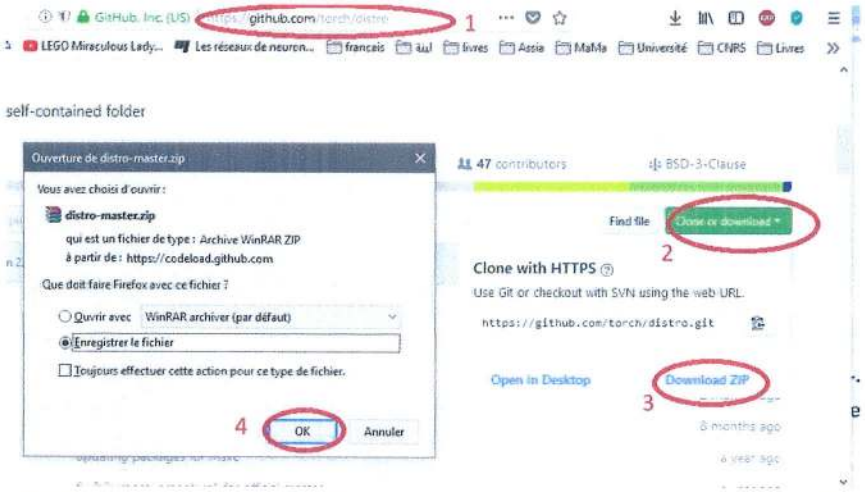
- Sous Ubuntu, dans un terminal, tapper la commande suivante :

`$sudo adduser username vboxsf` (username est votre nom d'utilisateur sous Ubuntu).

Si cette commande réussit, le dossier partagé (Echange) apparait dans le répertoire **/media/sf_Echange**.

Ce répertoire permettre de transférer directement des fichiers entre Windows et Ubuntu.

8- Télécharger le code source de Torch à partir du lien : **https://github.com/torch/distro**. Cliquer sur '**Clone or download**' puis sur **Zip**. Le fichier '**distro-master.zip**' sera téléchargé.



Si vous l'avez téléchargé à partir de Windows, copier le dans le répertoire partagé 'Echange'. Dans la MV Ubuntu, il est dans le répertoire **/media/sf_Echange/distro-master.zip**. Copier le dans votre répertoire de travail.

Si vous l'avez téléchargé à partir de Ubuntu, copier le à partir du répertoire '**Téléchargements**' et mettez le votre répertoire de travail.

Dans un Terminal, tapez les commandes suivantes :

$ unzip distro-master.zip (Un répertoire **distro-master** apparaît)

$ mv distro-master torchV1 (renommer le répertoire)

$ cd torchV1

$ bash install-deps (Mettre à jour le système. Le problème de **Liberror** sera résolu. Il faut que la MV Ubuntu a accès à Internet. Cette commande peut prendre beaucoup de temps et le système vous demande de temps à autre votre mot de passe ou le choix entre quelques options.

Si tous est exécuté avec succès, TORCH est prêt à être installé. Vous pouvez suivre la procédure décrite sur le site : **http://torch.ch/docs/getting-started.html#_**

Ces étapes sont : (dans un terminal, tapez les commandes suivantes)

1- **$ git clone https://github.com/torch/distro.git ~/torch --recursive**

2- **$ cd ~/torch; bash install-deps;**

3- **$ ./install.sh**

4- Lancer un autre terminal et tapez les commandes suivantes :

5- **$ luarocks install image**

6- **$ luarocks list**

7- **$ th** (Lancer Torch 7)

Pour Tester torch7, vous pouvez regarder les exemples fournis à l'adresse :
http://torch.ch/docs/five-simple-examples.html#_

ANNEXE B : UTILISATION DE L'OUTIL TORCH7

1 Torch 7

Torch est une bibliothèque d'apprentissage automatique open source, un framework de calcul scientifique et un langage de script basé sur le langage de programmation Lua, qui est maintenu par Ronan Collobert, Koray Kavukcuaglu, Clément Farabet. Il fournit une large gamme d'algorithmes pour l'apprentissage en profondeur, et utilise le langage de script LuaJIT, et une implémentation C/CUDA sous-jacente. **[20]**

1.1 Les fonctionnalités de base

- Il fournit un tableau ou un Tensor N-dimensionel flexible, qui supporte les routines de base pour l'indexation.
- Le moulage de type, le redimensionnement, le partage de stockage, et le clonage.
- Routines d'algèbre linéaire.
- Réseaux de neurones, basés sur les modules.
- Routines d'optimisation numérique.
- Prise en charge rapide et efficace du GPU.
- Intégrable, avec des ports pour les backends iOS et Androïd. **[20]**

1.2 Applications

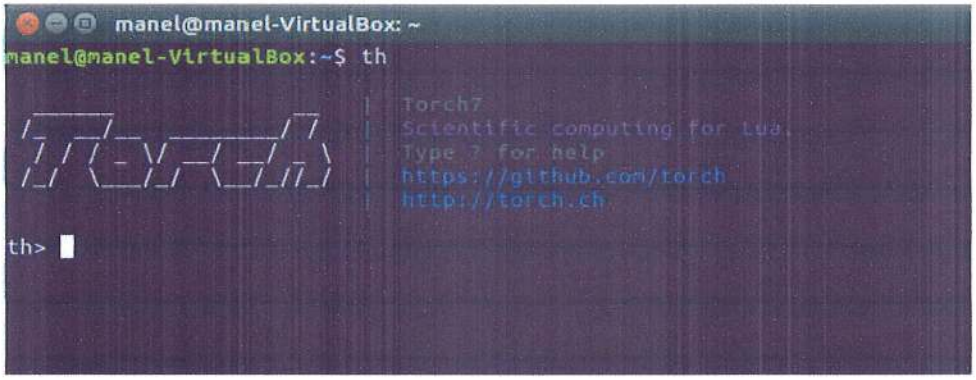
Facebook a publié un ensemble de modules d'extension en tant que logiciel open source.

Torch est utilisé par le groupe de recherche facebook AI (Artificial Intelligence), IBM, Yandex et l'institut de recherche Idiap. Torch a été étendu pour être utilisé sur Androïde et iOS. Il a été utilisé pour construire des implémentations matérielles pour des flux de données comme ceux trouvés dans les réseaux de neurones. **[20]**

Pour commencer avec Torch, il faut tout d'abord l'installer correctement (voir annexe I).

1.3 Une vue globale sur Torch7

Après l'installation, l'utilisation de Torch est assez simple et facile. Pour le lancement nous devons lancer un terminal tout d'abord puis entrer la commande « th » qui permet de lancer des commandes sous Torch (Voir la figure ci-dessous).



- Pour quitter Torch tapez : ctrl + C deux fois ou tapez la commande `os.exit()`.

- Pour Lancer l'exécution du code qui se trouve dans un fichier, il faut utiliser la commande dofile qui a la syntaxe suivante :

`dofile "nom de fichier.lua"`

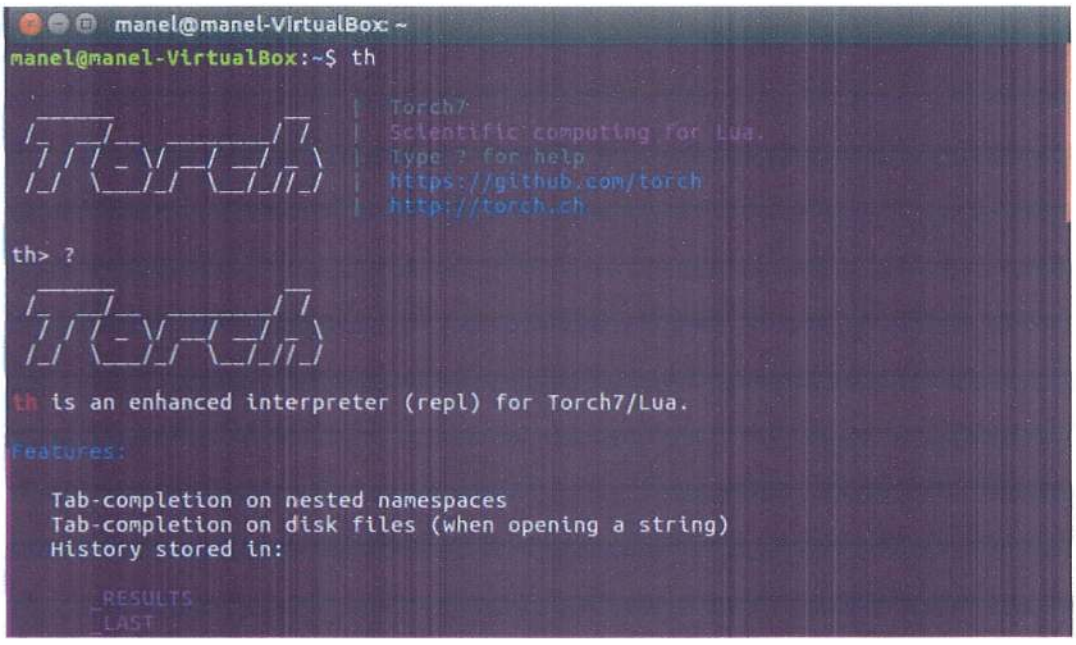
- Pour installer de nouveaux paquets, on utilise le gestionnaire de paquets luarocks comme suit :

`luarocks install nom_de_paquet.`

Exemple : **`$luarocks install image`**

- Les commentaires sont précédés par les deux caractères `"--"` pour une ligne. si il y a plusieurs lignes on met `"--[[ le commentaire ]]"`.

- Et pour plus d'information et d'aide sur Torch, on peut utiliser le point d'interrogation `" ? "`.



1.4 Exemples d'utilisation de Torch

Quelques exemples explicatifs simple appliqués sur les chaines, les tableaux, les fonctions, les boucles, etc.

Les chaines de caractères

L'affichage se fait avec ' `Print` '.

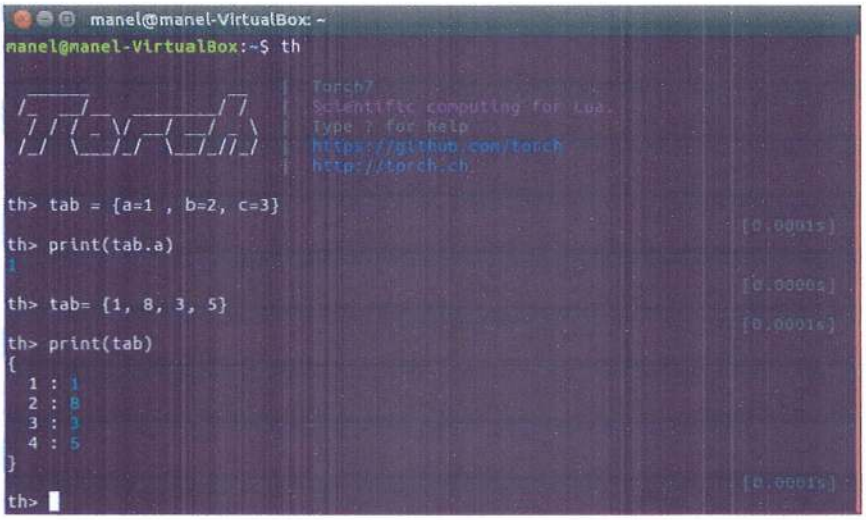
La concaténation de chaines se fait avec l'operateur '`..`' .

Exemples :

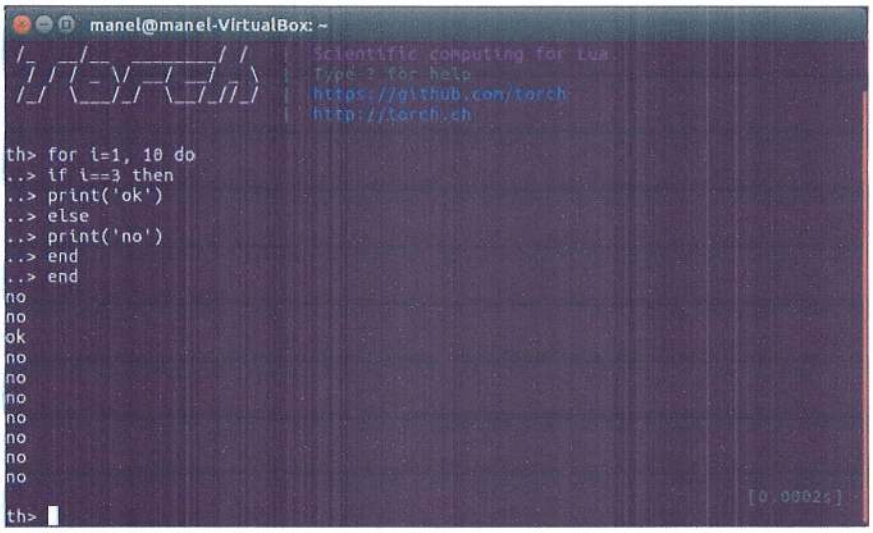
Les tableaux

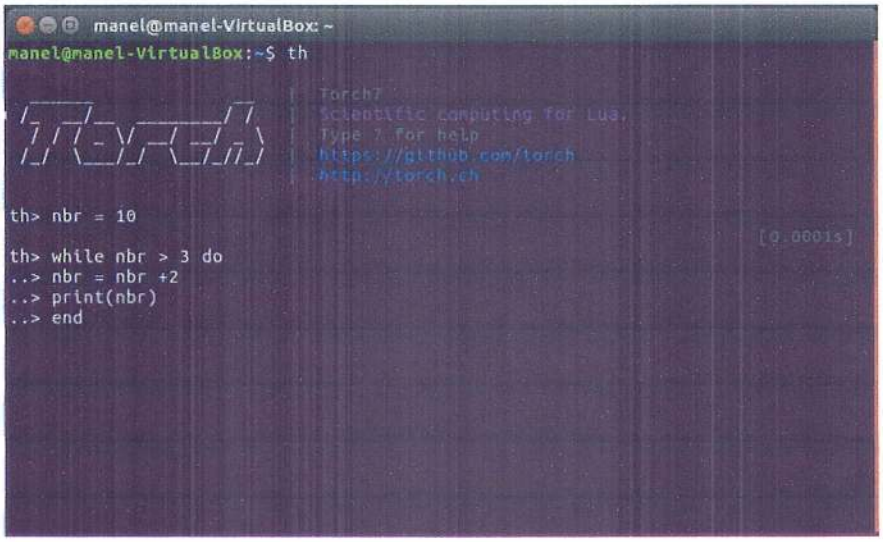
On déclare un tableau vide avec '`NomTableau = {}`'

Exemple :



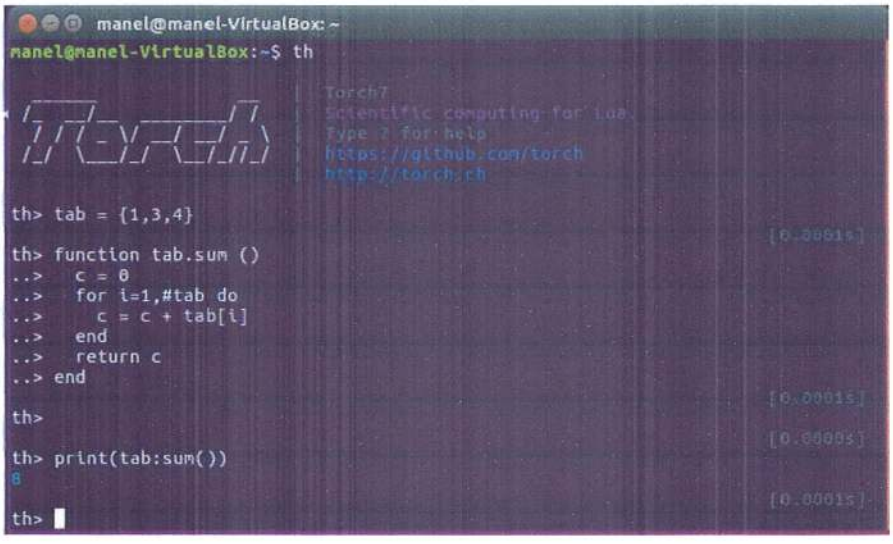
Les boucles





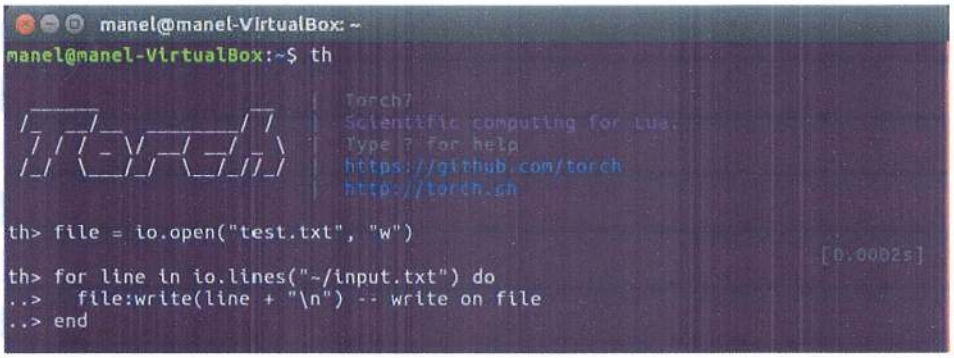
Les fonctions

Exemple :



La Lecture/Écriture (Input/Output)

Exemple :



Exemple d'utilisation du paquet 'nn' (Neural Network) :

On applique un simple exemple sur le paquet 'nn' :

Pour appeler le paquet on fait :

```
require 'nn'
```

Supposons, que nous avons besoin de deux couches feedforward cachées. La fonctionnalité d'une couche de feedforward simple est donnée par le module nn.Linear, qui applique simplement une transformation linéaire aux données d'entrée. Les réseaux de neurones sont adaptés pour l'apprentissage de la représentation non linéaire des données d'entrée. Par conséquent, nous voulons généralement appliquer une sorte de non-linéarité à chaque couche. Pour cela, on ajoute une fonction de transfert (par exemple nn.Tanh, nn.Sigmoid, nn.ReLU, etc ...) au modèle. Supposons que nous voulons utiliser tanh comme fonction de transfert, nous avons une entrée à 10 dimensions et nous voulons 10 unités dans chaque couche cachée. [21]

```
require 'nn'

mlp = nn.Sequential()

inputSize = 10

hiddenLayer1Size = opt.units

hiddenLayer2Size = opt.units


mlp:add(nn.Linear(inputSize, hiddenLayer1Size))

mlp:add(nn.Tanh())

mlp:add(nn.Linear(hiddenLayer1Size, hiddenLayer2Size))

mlp:add(nn.Tanh())
```

Enfin, nous avons besoin d'une couche de sortie. En supposant que nous voulons effectuer une sorte de tâche de classification (par exemple, en choisissant entre 2 classes), la fonction de transfert ***Softmax*** est le choix le plus commun.

```
nclasses = 2


mlp:add(nn.Linear(hiddenLayer2Size, nclasses))
mlp:add(nn.LogSoftMax())
```

Nous pouvons maintenant imprimer le modèle :

```
print mlp
```

Ce qui produira quelque chose comme suit :

```
nn.Sequential {
  [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> output]
  (1): nn.Linear(10 -> 10)
  (2): nn.Tanh
  (3): nn.Linear(10 -> 10)
  (4): nn.Tanh
  (5): nn.Linear(10 -> 2)
  (6): nn.LogSoftMax
}
```

Une chose utile à tester lors de la définition d'un NN est d'être sûr que la passe avant fonctionne et fait ce qu'elle est censée faire. C'est facile à réaliser avecle `Module:forward`.

```
out = mlp:forward(torch.randn(1,10))
print(out)
```

La sortie doit être un tenseur 1x2, où out[1][i] est la probabilité de log que l'entrée (générée aléatoirement dans cet exemple) appartient à la classe i.

Exemple sur l'entrainement :

Pour l'entrainement on peut utiliser l'implémentation prête à l'emploi de la descente stochastique `nn.StochasticGradient`. Il prend le modèle que nous venons de définir et une fonction de perte (*Critère*).

`nn.StochasticGradient` a plusieurs paramètres qui peuvent être modifiés, dont l'un est le taux d'apprentissage, que nous allons maintenant définir à la valeur spécifiée dans les options de ligne de commande **[21]**:

```
criterion = nn.ClassNLLCriterion()
```

```
trainer = nn.StochasticGradient(mlp, criterion)
trainer.learningRate = opt.learningRate
```

Ou on utilise le paquet `'optim'` et sa classe `'sgd'` pour l'apprentissage du réseau. Cette méthode offre plus de libertés dans les choix d'optimisation et permet aussi d'entrainer le réseau en mode mini-batch. Le mini-batch consiste à grouper des exemples d'apprentissage avant rétro-propagation du gradient.

Exemple sur les graphiques :

Nous pouvons maintenant tracer nos graphiques. Une première approche simple est d'utiliser `gnuplot.plot(x, y)`. Cette commande doit obligatoirement suivre la commande `gnuplot.figure()` pour s'assurer que les parcelles sont sur des figures différentes. [21]

```
require 'gnuplot'
gnuplot.figure(1)
gnuplot.title('CG loss minimisation over time')
gnuplot.plot(cgtime, cgevaluations)

gnuplot.figure(2)
gnuplot.title('SGD loss minimisation over time')
gnuplot.plot(sgdtime, sgdevaluations)
```

Une façon plus avancée, qui trace tout sur le même graphique serait la suivante. Ici, nous enregistrons tout dans un fichier PNG.

```
gnuplot.pngfigure('plot.png')
gnuplot.plot(
   {'CG', cgtime, cgevaluations, '-'},
   {'SGD', sgdtime, sgdevaluations, '-'})
gnuplot.xlabel('time (s)')
gnuplot.ylabel('J(x)')
gnuplot.plotflush()
```

Désinstallation de Torch7

Pour désinstaller Torch7, il suffit de supprimer son répertoire d'installation. Lancer un terminal et taper la commande : `rm -rf ~/torch`.

ANNEXE C : CODE SOURCE DU DEUXIEME MODELE

Dans torch7, le modèle est défini comme suit :

```
require 'nn'        # le package pour créer un réseau de neurones.
```

```
1 : className = { 'animal', 'arbre', 'avion', 'bateau', 'bus', 'camion', 'personne', 'train', 'velo', 'voiture'}
```

1 : déterminer les classes utilisées.

```
2 : net = nn.Sequential()
```

2 : création d'une séquence de couches.

```
3 : net:add(nn.SpatialConvolution(3, 32, 3, 3, 1,1,1,1))
```

3 : cette commande permet de créer une couche de convolution de 32 filtres avec un filtre de taille 3x3, un stride de 1 et un padding de 1.

```
4 : net:add(nn.ReLU())
```

4 : Une fonction d'activation de type ReLU.

```
5 : net:add(nn.SpatialConvolution(32, 64, 3, 3, 1,1,1,1))
```

5 : cette commande permet de créer une couche de convolution de 64 filtres avec un filtre de taille 3x3, un stride de 1 et un padding de 1.

```
6 : net:add(nn.ReLU())
```

6 : Une fonction d'activation de type ReLU.

```
7 : net:add(nn.SpatialMaxPooling(2,2,2,2))
```

7 : cette commande permet de réduire la taille de l'image avec un filtre de 2x2 et stride de 2.

```
8 : net:add(nn.SpatialConvolution(64, 64, 3, 3, 1,1,1,1))
```

8 : cette commande permet de créer une couche de convolution de 64 filtres avec un filtre de taille 3x3, un stride de 1 et un padding de 1

```
9 : net:add(nn.ReLU())
```

9 : Une fonction d'activation de type ReLU.

```
10 : net:add(nn.SpatialMaxPooling(2,2,2,2))
```

10 : cette commande permet de réduire la taille de l'image avec un filtre de 2x2 et stride de 2.

```
11 : net:add(nn.SpatialConvolution(64, 128, 3, 3, 1,1,1,1))
```

11 : cette commande permet de créer une couche de convolution de 128 filtres avec un filtre de taille 3x3, un stride de 1 et un padding de 1.

```
12 : net:add(nn.ReLU())
```

12 : Une fonction d'activation de type ReLU.

```
13 : net:add(nn.SpatialConvolution(128, 128, 3, 3, 1,1,1,1))
```

13 : cette commande permet de créer une couche de convolution de 128 filtres avec un filtre de taille 3x3, un stride de 1 et un padding de 1.

```
14 : net:add(nn.ReLU())
```

14 : Une fonction d'activation de type ReLU.

```
15 : net:add(nn.SpatialMaxPooling(2,2,2,2))
```

15 : cette commande permet de réduire la taille de l'image avec un filtre de 2x2 et stride de 2.

```
16 : net:add(nn.View(64*6*6))
```

16 : remodeler le tenseur avec la taille 128 * 6 * 6 en vecteur 1D.

```
17 : net:add(nn.Linear(64*6*6, 512))
```

17 : cette commande permet de créer une couche cachée de 512 neurones.

```
18 : net:add(nn.Dropout(0.5))
```

18 : cette commande permet de ne pas tomber dans le sur-apprentissage.

```
19 : net:add(nn.ReLU())
```

19 : Une fonction d'activation de type ReLU.

```
20 : net:add(nn.Linear(512, 512))
```

20 : cette commande permet de créer une couche cachée de 512 neurones.

```
21 : net:add(nn.Dropout(0.5))
```

21 : cette commande permet de ne pas tomber dans le sur-apprentissage.

```
22 : net:add(nn.ReLU())
```

22 : Une fonction d'activation de type ReLU.

```
23 : net:add(nn.Linear(512, #className))
```

23: cette commande permet de créer une couche de sortie de 10 classes.

```
24 : net:add(nn.LogSoftMax())
```

24 : la fonction softmax est utilisé pour calculer les probabilités de chaque classe.

La base d'image :

```
25: function DataGen:__init(path)
      torch.setnumthreads(1)
      self.rootPath = path
```

```
        self.trainImgPaths = self.findImages(paths.concat(self.rootPath,
'train_file/'))
        self.valImgPaths = self.findImages(paths.concat(self.rootPath,
'test_file/'))
        self.nbTrainExamples = #self.trainImgPaths
        self.nbValExamples = #self.valImgPaths
    end
```

25: Pour accéder à la base d'images que nous avons créée, composée de deux bases (répertoires) train_file pour l'entrainement, et test_file pour la validation.

Partie entrainement :

```
26 : self.optimState = { learningRate = opt.learningRate or 0.01,
learningRateDecay = 0.0, momentum = opt.momentum,
weightDecay = opt.weight_decay}
```

26: définir optimiseur à utiliser, on a choisi Stochastic Gradient Descent (sgd),

```
27 : self.model:training()
```

27 : Permet de retourner le modèle en mode entrainement.

```
28 : output = self.model:forward(input)
     local loss = self.criterion:forward(output, target)
     local critGrad = self.criterion:backward(output, target)
     self.model:backward(input, critGrad)
```

28 : La fonction forward qui calcule la sortie des entrées données passent par le modèle, elle est suivi de la fonction backward qui fait la retro-propagation des paramètres et on a utilisé une fonction de perte criterion

```
29 : self.model:evaluate()
```

29 : Permet de mettre le modèle en mode de évaluation

```
30 : output = self.model:forward(input)
     local loss = self.criterion:forward(output, target)
```

30 : utilisation de la fonction forward pour le calcule de la sortie pour les entrées données, et de la fonction de perte pour calculer l'erreur.

```
31 : output = self.model:forward(input)
     local loss = self.criterion:forward(output, target)
```

31 : pour la validation on utilise seulement une fonction de forward pas de propagation.

BIBLIOGRAPHIE

[1] BARRIERE Valentin. 04/09/2015. APPROCHES<<DEEP LEARNING>> APPLIQUEES AUX SIGNAUX AUDIO : PAROLE ET MUSIQUE. Rapport de fin d'études. IRIT - Institut de Recherche en Informatique de Toulouse, Toulouse, France

[2] Reconnaissance de formes, wikipedia, https://fr.wikipedia.org/wiki/Reconnaissance_de_formes

[3] COUR Timothée, GIRAUD Guillaume, KODSI Antoine *et al.* Juillet 2002. *RECONNAISSANCE DE FORMES PAR RESEAU DE NEURONES. Rapport de projet. ECOLE POLYTECHNIQUE, Palaiseau, France.*

[4] TAFFAR Mokhtar. Janvier 2016. *INITIATION A L'APPRENTISSAGE AUTOMATIQUE.* Support de Cours pour étudiants en Master en Intelligence Artificielle, Université de Jijel, http://elearning.univ-jijel.dz/elearning/pluginfile.php/4333/mod_resource/content/1/SupportCours_Mokhtar_Taffar_AppArAuto.pdf.

[5] MOKRI Mohammed Zakaria. 03/07/2017. *Classification des images avec les réseaux de neurones convolutionnels.* Mémoire de fin d'études de Master Informatique. Université Abou Bakr Belkaid. Tlemcen. http://dspace.univ-tlemcen.dz/bitstream/112/12235/1/Classification-des-images-avec-les-reseaux-de-neurones.pdf.

[6] Léon Personnaz, Isabelle Rivals, (2003). *Réseaux de neurones formels pour la modélisation, la commande et la classification.* Préface de Gérard Toulouse: CNRS EDITTIONS.

[7] Gérard Dreyfus, Manuel Samuelides, Jean-Marc Martinez, Mirta B. Gordon, Fouad Badran, Sylvie Thiria, Laurent Hérault. "Réseaux de neurones. Méthodologie et applications – Chapitre 1". 2004 2ième édition. Eyrolles. (https://www.eyrolles.com/Chapitres/9782212110197/chap01.pdf).

[8] Gérald PETITJEAN, INTRODUCTION AUX RESEAUX DE NEURONES PDF http://www.lrde.epita.fr/~sigoure/cours_ReseauxNeurones.pdf.

[9] Réseaux de neurones : Applications des réseaux de neurones https://fr.wikiversity.org/wiki/R%C3%A9seaux_de_neurones/Applications_des_r%C3%A9seaux_de_neurones.

[10] Intelligence artificielle, RÉSEAUX DE NEURONES ARTIFICIELS, 2015. http://villemin.gerard.free.fr/Wwwgvmm/Logique/IAneuron.htm.

[11] Apprentissage profond, wikipedia, https://fr.wikipedia.org/wiki/Apprentissage_profond.

[12] « Classification d'images : les réseaux de neurones convolutifs en toute simplicité », octo talks, Posté le 25/10/2016 par Julien Krywyk, Pierre-Alain Jachiet, https://blog.octo.com/author/julien-krywyk-kry/.

[13] Réseau *neuronal convolutif, wikipedia,* https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif.

[14] Eugenio Culurciello, 19/04/2017, "The History of Neural Networks"', https://dataconomy.com/2017/04/history-neural-networks/.

[15] CS231n Convolutional Neural Networks for Visual Recognition, http://cs231n.github.io/convolutional-networks/#conv.

[16] Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 14/04/2017, "An Analysis of Deep Neural Network Models for Practical Applications", https://arxiv.org/pdf/1605.07678.

[17] Tensorflow: An open source machine learning framework for everyone, siteweb, https://www.tensorflow.org/.

[18] Keras: The Python Deep Learning library, siteweb, https://keras.io/.

[19] Introduction to the Python Deep Learning Library Theano, siteweb, https://machinelearningmastery.com/introduction-python-deep-learning-library-theano/.

[20] Getting started with Torch, siteweb, http://torch.ch/docs/getting-started.html.

[21] Torch7. Hello World, Neural Networks!, siteweb, http://mdtux89.github.io/2015/12/11/torch-tutorial.html.