الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne démocratique et populaire

وزارة التعليم العالي و البحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كلية التكنولوجيا
Faculté de Technologie

قسم الإلكترونيك
Département d'Électronique

Mémoire de Master

Filière Électronique
Spécialité Électronique des Systèmes embarqués

présenté par

OUACHEMI Ahcene

&

OUCHFOUN Aness

Classification des Panneaux de Signalisation Routière par les Réseaux Neuronaux Convolutifs

Proposé par : Naceur Djamila

Année Universitaire 2019-2020

Remerciements

"Nous remercions Allah, le tout puissant de nous avoir aidé à accomplir ce travail, et qui a été avec nous à tous les moments de notre chemin d'étude. A nos chers parents pour leur Patience, leur Amour, leur Soutien et leur Encouragement. Nos sincères remerciements à tous les membres du jury qui nous ont fait l'honneur de réviser ce travail. Un remerciement spécial à Madame Djamila NACEUR pour sa patience, son aide et ses conseils pour élaborer ce travail. "

Résumé

De nos jours, les tâches de reconnaissance d'objets sont de plus en plus résolues avec les réseaux neuronaux convolutionnels (CNN). En raison de son haut taux de reconnaissance et une exécution rapide, les réseaux de neurones convolutifs ont amélioré la plupart des tâches de vision par ordinateur, à la fois existants et nouveaux.

L'objectif premier est l'implémentation d'un algorithme de classification de panneaux de signalisation utilisant une convolution réseau neuronal. L'entraînement du réseau de neurones est implémenté à l'aide de la bibliothèque TensorFlow et d'autres bibliothèques comme keras et d'une architecture massivement parallèle pour la programmation.

L'ensemble de la procédure de classification des panneaux de signalisation est exécuté en temps réel sur un GPU et CPU. Les résultats expérimentaux ont confirmé la haute efficacité du système de vision par ordinateur développé.

Mots clé : CNN, TensorFlow, Keras, CPU, GPU.

Abstract

Nowadays, more and more object recognition tasks are solved with convolutional neural networks (CNN). Due to its high recognition rate and rapid execution, convolutional neural networks have improved most computer vision tasks, both existing and new.

In this thesis, we propose an implementation of an algorithm for classification of traffic signs using a neural network convolution. This thesis also shows several CNN architectures, which are compared with each other. Neural network training is implemented using the TensorFlow library and other libraries like keras and a massively parallel architecture for programming.

Automatic recognition of traffic signs is required in advanced driver assistance systems and constitutes a challenging real-world computer vision and pattern recognition problem. A comprehensive, lifelike dataset of more than 34,000 traffic sign images has been collected. It reflects the strong variations in visual appearance of signs due to illumination, weather conditions, partial occlusions, and rotations. The images are complemented by several precomputed feature sets (rgb to grayscale, normalization) to allow for applying machine learning algorithms without background knowledge in image processing. The dataset comprises
43 classes with unbalanced class frequencies

A traffic sign image is passed through a set of layers such as convolution layers and max pooling layers, some filters reduce the image resolution by the local maximum operation, and after each convolution layer we get a new image with less characteristics compared to the previous image
Called a convolution map

In order to improve neural network performance and facilitate recognition work, we will use some simple and effective techniques such as data augmentation and dropout. We will also

cite information such as how long does it take to train and how much data of each class is needed to have a good classification model.

The entire procedure for recognizing traffic signs is executed in real time on a GPU and CPU. The experimental results confirmed the high efficiency of the developed computer vision system.

The results obtained showed that the choice of the epoch number, the size of the image database and the depth of the network have a great influence to have better results

Keyword : CNN, TensorFlow, Keras, CPU, GPU.

ملخص

الشبكات العصبونية التفافية هي شبكات عصبية متعددة طبقات تتمثل مهامها في التعرف على الأنماط في عملنا قمنا باستخدام هذه شبكات في تصنيف الصور للإشارات المرور حيث اقترحنا نموذج المتكون من طبقات التفافية طبقات انتقائية طبقات اتصال الكامل وعدد تكرارات

النتائج المتحصل عليها وحجم قاعدة الصور وعمق الشبكة لديها تأثير كبير للحصول على نتائج أفضل

.

Table des matières

Liste des figures 1

Liste des tableaux 3

Acronymes……4

Introduction Générale..... 5

Chapitre 1 : Classification des images

1. Introduction. 9

2. Motivation…….……..9

3. Différents type d'image …… ..... …. 10

3.1. Image binaire 10

3.2. Image en niveau de gris 11

3.3. Image RBV 12

3.4. Image indexé..... 12

4. Caractéristiques d'image ……..…...13

4.1. Pixels 13

4.2. La résolution 13

4.3. Voisinage 14

4.4. Contraste..... 14

4.5. Niveau de gris 15

4.6. Luminance 15

4.7. Bruit 15

4.8. Contour 15

5. Méthodes de classifications 16

5.1. Méthode supervisé..... 16

5.2. Méthode non supervisé ……16

6. Modèles de classification……..17

6.1. Matrice de confusion ……...17

6.2. Classification ROC et AUC ……....19

7. Classification des images en fonction machine Learning…….20

8. Classification des images par les réseaux de neurones…….21

9. Conclusion……22

Chapitre 2 : Les réseaux de neurones convolutionnels

1. Introduction 24

2. Le réseaux perceptron multicouche PMC…….…….26

2.1. Modèle du perceptron…….……27

2.2. Fonctions d'activation…….……28

2.3. La notion de rétropropagation…….……....30

3. Différents types de réseaux de neurones artificiels 31

3.1. Adaline (adaptive linéaire neurones)..... 31

3.2. PMC à fonctions radiales (RBF) 31

3.3. Kohonen …….32

4. L'apprentissage profond Deep learning …….....32

4.1. Qu'est-ce que l'apprentissage profond (DL)..... 32

4.2. Algorithmes de deep learning 32

5. Les réseaux de neurones convolutionnels 33

5.1. Réseaux de neurones convolutionnels et PMC 33

5.2. Architecture du réseau de neurones convolutionnels ……34

5.2.1. Couche de convolution …….35

5.2.2. Couche de Pooling (Pool) …….37

5.2.3. Couche de correction ……38

5.2.4. Couche entièrement connectée (FC) …….39

5.2.5. Couche de perte (LOSS) ……...39

6. Exemple de modèles de CNN ……..40

7. Choix des hyperparamètres ……..40

7.1. Nombre de filtres……40

7.2. Forme du filtre ……...41

7.3. Forme de max pooling ……...41

8. Méthodes de régularisation …… ……..41

8.1. Empirique……41

8.1.1. Dropout……...41

8.1.2. DropConnect……...42

8.1.3. Pooling sochastique……42

8.2. Explicite…….……..42

8.2.1. Taille de réseau ……..42

8.2.2. Dégradation des poids……42
9. Paramètres du le réseau CNN …….. ……....43
10. Conclusion ……44

Chapitre 3 : **L'implémentation**

1. Introduction 46
2. Logiciels et librairies Utilisés dans l'implémentation……...46
2.1. Python……..……46
2.1.1. Python et version utilisée……..…….……....…46
2.2. TensorFlow …… 47
2.2.1. Installation TensorFlow…….47
2.3. Keras……48
2.3.1. Installation keras……48
2.4. Scikit-learn……...48
2.5. Matplotlib……48
2.6. Configuration utilisé dans l'implémentation……49
3. La bases d'image GTSR..... …..49
3.1. Exploration et visualisation l'ensemble de données 51
3.1.1. Prétraitement des images……..……..……..51
3.1.2. L'augmentation des données……52
4. Architecture de notre réseau ……....52
5. Calcule des paramètres de notre modèle 54
6. Résultats obtenus et discussion……55
6.1. Discutions……....55
6.**2.** Implémentation sur un GPU ……...57
6.2.1. Configuration utilisé dans l'implémentation GPU……60
6.2.2. Résultat obtenu sur GPU……...60
6.3. Tableau de comparaison des résultats……..61
6.4. Exécution en temps réel par une webcam……62
6.**5.** Résultat……62
6.6. Observation……..65
7. Code Source du modèle……66

8. Conclusion ……....67
Conclusion générale ……68
Références bibliographiques……69
Annexe ……...……71

Liste des figures

Chapitre 1 : Classification des images

Figure 1.1: Image binaire et la valeur des pixels dans un voisinage 6 * 6 10
Figure 1.2: Image en niveaux de gris 11
Figure 1.3: Canaux RVB 12
Figure 1.4: Image de 1200 pixels 13
Figure 1.5: Voisinage à 4 14
Figure 1.6: Voisinage à 8 14
Figure 1.7: Image faible contraste et forte contraste 14
Figure 1.8: Taux de VP et de FP pour différents seuils de classification 19
Figure 1.9: AUC (aire sous la courbe ROC) 20

Chapitre 2 : Réseaux de neurones convolutionnels

Figure 2.1: Shallow Learning 24
Figure 2.2: Deep architecture "Deep learning" 24
Figure 2.3: Inconvénients des réseaux neuronaux précédents PMC 25
Figure 2.4: Schéma du réseau PMC 26
Figure 2.5: Modèle du perceptron 27
Figure 2.6: Fonction d'activation linéaire 28
Figure 2.7: Fonction d'activation sigmoïde 29
Figure 2.8: Fonction d'activation tangente hyperbolique 29
Figure 2.9: Fonction d'activation relu 30
Figure 2.10: Une couche du CNN en 3 dimensions. 34
Figure 2.11: Architecture d'un réseau de neurone convolutionnel 35
Figure 2.12: Calcul le nombre de neurones du volume de sortie 37
Figure 2.13: Pooling avec un filtre 2x2 et un pas de 2 38
Figure 2.14: Oopération relu 38
Figure 2.15: Après couche de pooling aplatie en tant que couche FC 39
Figure 2.16: Exemples de modèles de CNN 40
Figure 2.17: Carte de caractéristiques en entrés et en sortie 43

Chapitre 3 : Implémentation

Figure 3.1: Les classes des panneaux de signalisation 49

Figure 3.2: MyData 49
Figure 3.3 : les étiquettes 50
Figure 3.4: Transformer des images RGB en niveau de gris 51
Figure 3.5: Normalisation des images 51
Figure 3.6: Distribution d'échantillons pour chaque classe. 52
Figure 3.7: Architecture de notre réseau 53
Figure 3.8: Configuration du notre modèle 54
Figure 3.9: Précision et Erreur pour le Modèle implémenté (CPU) 55
Figure 3.10: Détails sur l'apprentissage de notre modèle sur CPU 56
Figure 3.11: Précision et Erreur pour le Modèle implémenté avec 30 époques 57
Figure 3.12 : Les pilotes et les bibliothèques utilisées avec ses versions…..... 57
Figure 3.13 : Installation de CUDA..... 58
Figure 3.14 : Copier les bibliothèques de cuDNN……..……..……..…….58
Figure 3.15: Remplacement des bibliothèques avec les bibliothèques de cuDNN……...58
Figure 3.16 : Ajoutez CUDA, cuDNN à la variable d'environnement……..59
Figure 3.17 : Installation de Microsoft Visual c++……..……...59
Figure 3.18 : Limitation de mémoire……..……..……...60
Figure 3.19: Détails sur l'apprentissage de notre modèle sur GPU 60
Figure 3.20 : Précision et Erreur pour le Modèle implémenté (GPU) 61
Figure 3.21: Reconnaissance d'une image "Entrée interdite" 63
Figure 3.22: Reconnaissance d'une image "limitation de vitesse 70km/h" 63
Figure 3.23: Reconnaissance d'une image "limitation de vitesse 30 Km/h" 64
Figure 3.24: Reconnaissance d'une image "Interdiction de depasser" 64
Figure 3.25: Reconnaissance d'une image "Interdiction aux poids lourds" 65
Figure 3.26: Reconnaissance d'une image "Limitation de vitesse 120 km/h" 65
Figure 3.27: Reconnaissance d'une image "STOP" 66

Liste des tableaux

Chapitre 1 : Classification des images

Tableau 1.1: Classification vs clustering 17

Tableau 1.2: Matrice de confusion 18

Tableau 3.1 : Résultats sur CPU & GPU 61

Acronymes

TSR : Traffic Signs Recognition

ADAS : Advanced driver-assistance systems

CNN : Convolutional neural network

CPU : Central processing unit

GPU : Graphical Processing Unit

GTSR : Germany traffic sign recognition

ROC : Receiver operating characteristic

AUC : Aire sous la courbe ROC

ML : Machine learning

DL : Deep learning

PMC : Perceptron MultiCouche

Tanh : Tangente hyperbolique

Relu : Unité linéaire rectifiée

Adaline : Adaptive linéaire neurones

RNN : Réseau de neurones récurrent

ConvNets : Convolution network

CONV : Couche de convolution

Pool : Couche de pooling

FC : Fully connected (entièrement connectée)

Introduction générale

Un conducteur peut être distrait de sa tâche principale, la conduite, ce qui occasionne un manque de vigilance vis à vis de la signalisation courante. Cette situation augmente le risque d'accident. En effet, manquer un panneau de limitation de vitesse ou d'interdiction de doubler génère une situation à risque en plus du fait d'exposer le conducteur à des sanctions.

Afin de réduire ce risque, les industriels automobiles intègrent de plus en plus de systèmes d'aide à la conduite lors de la conception de nouveaux véhicules, vu que les experts en accidentologies, ont pu déterminer que le facteur humain représente la première cause dans 90% des accidents.

Le Développement du niveau technique des processeurs mobiles modernes a permis à de nombreux fabricants de véhicules d'installer des systèmes de vision par ordinateur dans les voitures des clients, Ces systèmes agiront à améliorer considérablement la sécurité et à mettre en œuvre une étape importante sur la voie de la conduite autonome.

Parmi les tâches résolues avec la vision par ordinateur dans cette situation, le problème de la reconnaissance des panneaux de signalisation, ce problème est l'un des plus connus et largement discuté par de nombreux chercheurs.

Cependant, les principaux problèmes de tels systèmes sont la faible précision de détection et la forte demande de matériel informatique performant, ainsi que l'incapacité de certains systèmes à classer les panneaux de signalisation de différents pays.

Pour contrôler les environnements, il existe des méthodes de classification, on a choisi une application de classification des panneaux de signalisation routière en temps réel utilisant une webcam, cela se fera en utilisant open cv à l'aide de réseaux de neurones.

Ce système crée une aide à la conduite automobile conçu dans le but d'éviter les situations dangereuses qui conduisent à des accidents, d'éloigner le conducteur de tous les effets de distraction, de le doter de la capacité de percevoir l'environnement extérieur et de lui permettre de percevoir et de gérer les risques à l'avance.

Un module de reconnaissance des panneaux de signalisation routière (Traffic Signs Recognition TSR) est un composant par laquelle un véhicule est capable de reconnaître les panneaux de signalisation par un flux vidéo provenant d'une caméra installée dans un véhicule, Cela fait partie des fonctionnalités appelées collectivement ADAS (Advanced driver-assistance systems)

Dans la fin des années 80 Yan le Cun a développé un type de réseau particulier qui s'appelle le réseau de neurone convolutionnel CNN (convolutional neural network), ces réseaux sont une forme particulière de réseau neuronal multicouche dont l'architecture des connexions est inspirée de celle du cortex visuel des mammifères. En 1995, Yan le Cun et deux autres ingénieurs ont développé un système automatique de lecture de chèques qui a été déployé largement dans le monde. À la fin des années 90, ce système lisait entre 10 et 20 % de tous les chèques émis aux États-Unis. Mais ces méthodes étaient plutôt difficiles à mettre en œuvre avec les ordinateurs de l'époque.

En 2012 un événement soudainement change la situation, les GPU (Graphical Processing Unit) capables de plus de mille milliards d'opérations par seconde sont devenus disponibles pour un prix moins cher. Ces puissants processeurs spécialisés, initialement conçus pour le rendu graphique des jeux vidéo, se sont avérés être très performants pour les calculs des réseaux neuronaux.

Dans notre projet on va utiliser les réseaux de neurones convolutionnels pour classifier les panneaux de signalisation routière, on va créer un modèle de réseau de neurones convolutifs proche de l'architecture LeNet-5 et par la suite on va appliquer ce modèle sur la base d'images GTSR.

Ce mémoire est organisé comme suit :

Le premier chapitre, s'ouvre sur une présentation des différents types d'images et de leurs caractéristiques, sur les méthodes de la classification des images de panneaux routière, ainsi que sur l'utilisation des réseaux de neurones dans la classification de ces images.

Le deuxième chapitre est consacré à la description des réseaux de neurones convolutionnels et de leurs intérêts dans le domaine de la classification des images.

Le troisième chapitre présente la partie expérimentale de notre travail donnant suite à la discussion des différents résultats obtenus

Et enfin on termine par une conclusion générale.

Chapitre 1 : Classification des images

1 Introduction

La classification d'images est un problème fondamental en vision par ordinateur, qui a de nombreuses applications concrètes telle la classification d'objets, la reconnaissance faciale, ou la classification de la maladie (COVID-19). Le but est de construire un système capable d'assigner correctement une classe à n'importe quelle image en entrée. Un tel système exploite des algorithmes de Machine Learning issus de l'apprentissage, il existe deux types d'apprentissage : supervisé et non supervisé, nous nous intéresserons à l'apprentissage supervisé. Ce chapitre va présenter des notions générales sur la classification.

2 Motivation

De nos jours, le nombre d'images augmente en fonction du domaine d'applications de façon exponentielle, par conséquent, il est important de les classer de manière fiable. Le concept de classification d'images classique consiste généralement à extraire des entités d'image locales, à les coder en tant que vecteur d'entités et à les classer à l'aide d'un modèle créé précédemment.

Toutes les applications de classification ont un point commun, la chaine de traitement qui consiste à extraire des caractéristiques pendant la phase d'apprentissage.

Avant de faire l'extraction des caractéristiques, il y a une phase très importante qui précède l'extraction dite le prétraitement, cette phase permet d'occulter ou d'atténuer toute information susceptible de nuire la description du contenu pertinent lors de la phase d'extraction de caractéristiques.

Parmi les techniques de prétraitement et d'amélioration de l'image, on en cite :

- Transformation de l'image de RGB en niveau de gris.
- Atténuation de bruits.
- Rehaussement de contraste.
- Extraction de contours par technique de filtrage.

L'objectif de la classification d'images est d'effectuer une tâche qui peut s'avérer coûteuse à acquérir par un être humain en raison notamment de contraintes physiques comme la concentration, la fatigue ou le temps nécessité par un volume important de data set. **[1]**

3 Différents types d'image

Un pixel est le plus petit élément d'une image numérique auquel on peut accéder, et son adresse définit les coordonnées physiques dans lesquelles il se trouve. En fonction des pixels et de la façon dont ils sont stockés, il existe les différents types d'images suivants **[2]** :

- Image binaire
- Image en niveaux de gris
- Image RVB
- Image indexée

3.1 Image binaire :

C'est le type d'image numérique dans lequel il n'y a qu'un seul bit par pixel 1 ou 0, généralement 0 signifie « noir » et 1 signifie « blanc », bien qu'il soit possible d'utiliser deux couleurs quelconques, Les images binaires sont codées sous forme de tableau 2D (matrice). **[2]**

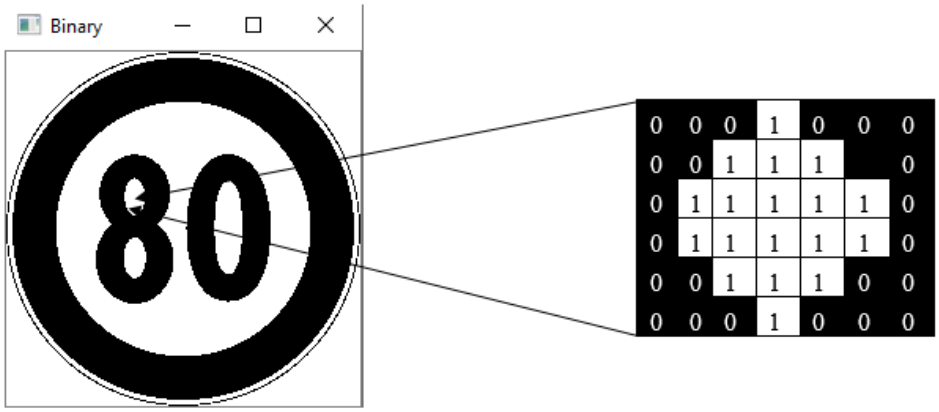


Figure 1.1: Image binaire et la valeur des pixels dans un voisinage 6 * 6

Le code Python complet et final, pour convertir une image RVB colorée en binaire est donné ci-dessous :

```
import cv2
import numpy

img = cv2.imread("index.png",0)


ret, bw = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
cv2.imshow("Binary",bw)
cv2.waitkey(0)


cv2.destroyAllwindows()
```

3.2 Image en niveaux de gris :

L'image en niveaux de gris ou monochrome est codée sous la forme d'un tableau 2D de pixels. Si le pixel est codé sur 8bits, chaque valeur de pixel va aller de 0 à 255, 0 correspondant au « noir » et 255 au « blanc ». Les valeurs intermédiaires indiquent différentes nuances de gris**. [2]**

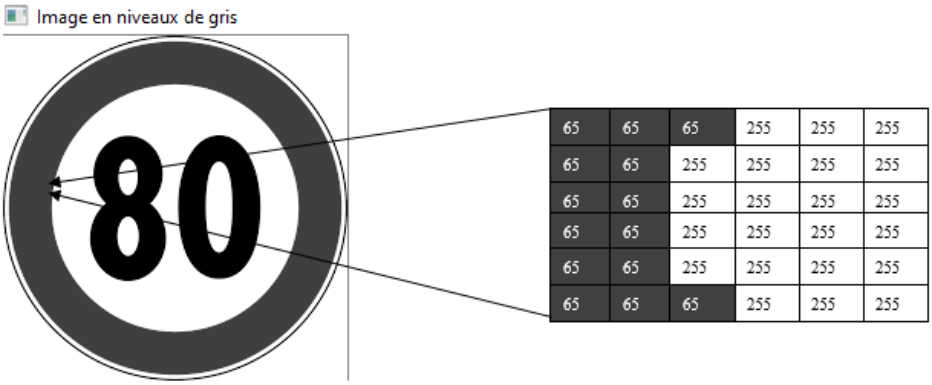


Figure 1.2: Image en niveaux de gris

Le code Python complet et final, pour convertir une image RVB colorée en niveau de gris est donné ci-dessous :

```
import cv2
import numpy



img= cv2.imread("index.png",0)

cv2.imshow("Image en niveaux de gris ",img)
cv2.waitkey(0)
```

3.3 Image RBV :

Une image peut étre représentée a l'aide de trois tableaux 2D (méme taille) chaque tableau représente un canal de couleur : Rouge (R) , vert(V) et bleu (B).

Chaque tableau contient une valeur de 8 bits sur une échelle [0,255] .la combinaison des trois valeur de 8bits forme un nombre de 24 bits , ce qui signfie 244 combinaisons de couleur**.[2]**

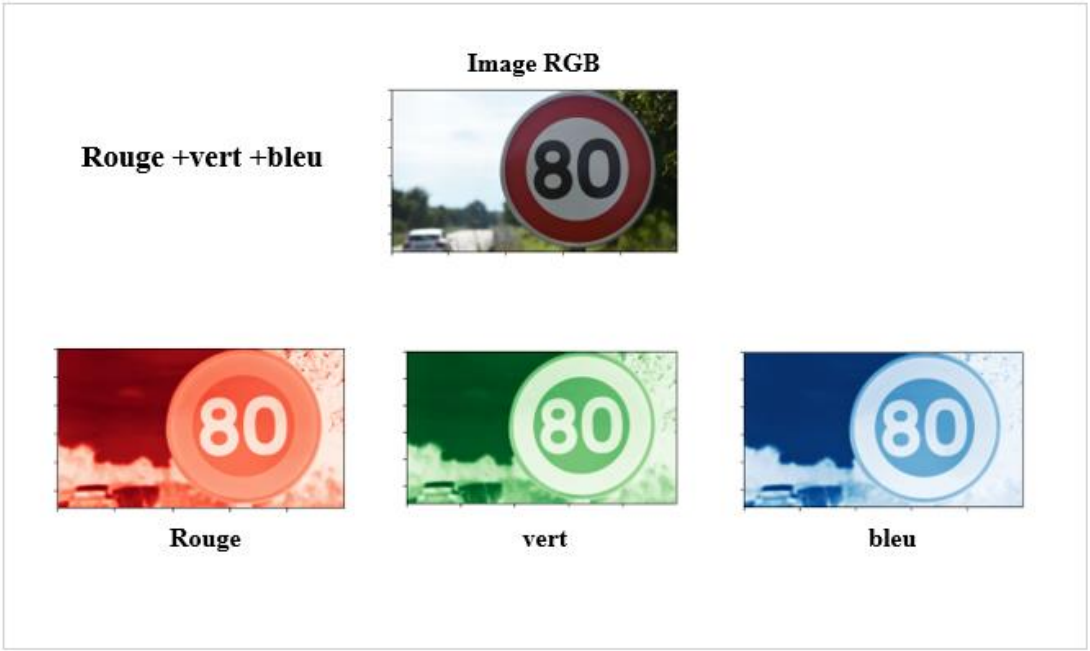


Figure 1.3: Canaux RVB

3.4 Image indexé :

Une telle image comporte deux parties :

- Matrice d'index
- Carte couleur

Ces images sont une représentation indexée, dans laquelle un tableau 2D, contient des index / pointeurs vers une palette de couleurs de taille maximale fixe (généralement 256 couleurs). La carte des couleurs n'est qu'une liste de couleurs utilisées dans l'image. [**2]**

4 Caractéristiques de l’image

L’image est une représentation d'un ensemble d'information formalisé par des paramètres suivants :

4.1 Pixels :

Généralement on exprime la définition d’une image en indiquant le nombre de pixels répartis sur la largeur et sur la hauteur, par exemple on dira de l'image suivante que sa définition est de 40 par 30, ce que signifie qu'elle possède 40 pixels sur la largeur et 30 sur la hauteur soit 1200 pixels en tout, ce qui représente la taille de l’image.

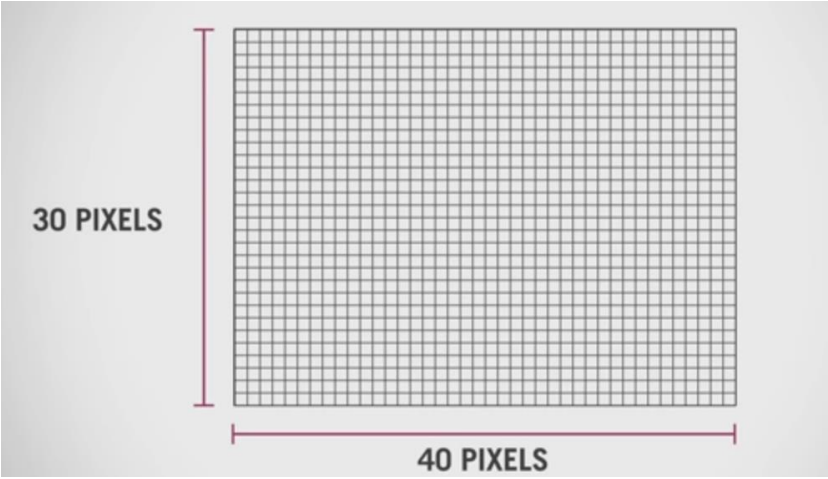


Figure 1.4: Image de 1200 pixels

L’information par pixel représentée l’intensité de l’image ou bien le niveau de gris ou la couleur, on a deux types d’image : image monochrome et l’image couleur.
La différence entre elles réside dans la quantité d’information contenu dans chaque pixel. L’image couleur représente l’intensité sur 3 chaines (3 OCTETS RBV : ROUGE BLEU VERT) par contre l’image monochrome est sur un octet.

4.2 La résolution :

La résolution d'une image est le nombre de pixels par pouce qu'elle contient (1 pouce = 2.54 centimètres). Elle est exprimée en "PPP" (points par pouce). Plus il y a de pixels (ou points) par pouce et plus il y aura d'information dans l’image. **[3]**

4.3 Voisinage :

La surface d'une image est représentée par l'ensemble des surfaces rectangulaires
On distingue deux type de voisinage **[4] :**

- Voisinage à 4 : pixels qui ont un coté commun avec le pixel centré.
- Voisinage à 8 : tout pixel situé autour du pixel centré.

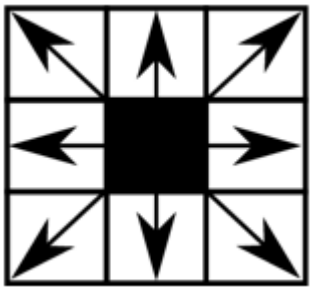


Figure 1.5: Voisinage à 8

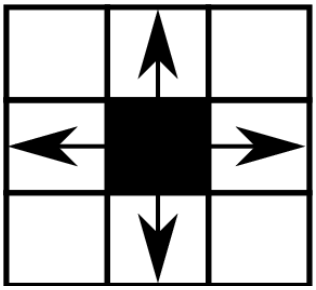


Figure 1.6: Voisinage à 4

4.4 Contraste :

Le contraste est une propriété intrinsèque d'une image qui quantifie la différence de luminosité entre les parties claires et sombres d'une image.

- **Une image contrastée** : présente une bonne dynamique de la distribution des valeurs de gris sur tout l'intervalle des valeurs possibles, avec des blancs bien clairs et des noirs profonds.
- **Une image faible contrastée :** a une faible dynamique, la plupart des pixels ayant des valeurs de gris très proches.

Figure 1.7: Image faible contraste et forte contraste

4.5 Niveau de gris (Grayscale) :

Une intensité lumineuse d'un pixel situé dans la surface de l'image, la plage de l'intensité entre 0 (noire) et 255 (blanc) elle représente aussi la quantité de la lumière réfléchie pour 8bits, on dispose 256 niveau de gris, plus le nombre de bit augmente plus les niveaux sont nombreux.

4.6 Luminance :

La luminance est une mesure photométrique de l'intensité lumineuse par unité de surface de lumière se déplaçant dans une direction donnée, pour une bonne luminance :

- Il faut que l'image soit lumineuse.
- Il faut un bon contraste sa qui veux dire que le contraste d'une image ne tend pas ni vers le noir ni vers le blanc, pour éviter des pertes de détails dans les zones sombres ou lumineuses.
- L'absence de **bruit.**

4.7 Bruit :

Dans une image, le bruit est toute fluctuation parasite ou dégradation que subit l'image de l'instant de son acquisition jusqu'à son enregistrement à cause de variation de l'intensité d'un pixel par rapport à ses voisins. Ce qui donne certains défauts (petits nuages, poussière et la diminution de l'intensité électrique sur les capteurs), Les sources de bruit sont multiples, certaines sont physiques liées à la qualité de l'éclairage de la scène, et électroniques liées à la stabilité du capteur de l'image durant l'acquisition **[6].**

4.8 Contour :

Le contour est la frontière qui sépare des objets dans une image qui ont des pixels dont les niveaux de gris différents, ou la limite des objets qui marquant des changement d'intensité.

5 Méthodes de classification

De nombreuses méthodes classiques ont été consacrés, elles peuvent être séparées en deux grandes catégories **:** les méthodes de classification supervisée et les méthodes de classification non supervisée.

5.1 Méthodes supervisés :

Les Méthodes de classification supervisée sont des méthodes dans lesquelles les classes sont connues avant l'identification des éléments de l'image. On utilise comme première phase, la phase l'apprentissage, cette dernière permet de définir des règles de classement à partir d'un ensemble d'objets de référence dont on connait l'identité à priori et qui sont représentatifs de chaque classe, Il est nécessaire de vérifier la crédibilité de ces règles pour évaluer notre apprentissage (sous apprentissage ou sur apprentissage en fonction de la complexité du modèle).

- Quelques méthodes qu'on peut citer en classification supervisé **[7]** :
  - ❖ Classification bayésienne.
  - ❖ Réseaux neurones.
  - ❖ Arbres de décision.

5.2 Méthodes non supervisés (clustering) :

Au Contraire de la classification supervisée, les classe sont pas identifiés et les données ne sont pas étiquetées et nécessitent aucun apprentissage, Elle consiste à représenter ensemble de groupes appelé cluster, chaque groupe collecte des objets, des objets qui sont semblables entre eux et des objets qui sont dissemblables par rapport d'autre objets de groupes, on peut le classer comme domaine de l'analyse des données **[7].**

- Quelques méthodes qu'on peut citer en classification non supervisé :
  - ❖ K- moyennes.
  - ❖ Nuées dynamiques.
  - ❖ Classification ascendante hiérarchique (analyse des données).

Classification	Clustering
• Nombre de classe connu • Sur entrainement • Utilisé pour classifier des données futures	• Nombre de classe inconnu • Pas de connaissance préalable • Utilisé pour comprendre et explorer les données

Tableau 1.1: Classification vs clustering

6 Modèle de classification

Nous allons maintenant nous concentrer sur les modèles de classification : on utilise des données étiquetées pour prédire à quelle classe un objet appartient. Nous allons surtout parler de classification binaire, où il s'agit de distinguer si un objet appartient ou non à une classe. Par exemple, dire si une image représente une girafe ou non. Si oui, on dit que cette image est positive ; sinon, qu'elle est négative.

Jusqu'à présent, en évaluant des modèles de classification, on déduit le nombre d'erreurs comme mesure de performance d'un modèle. Mais ce n'est pas le seul critère, dans cette partie on va montrer différentes manières d'évaluer un modèle de classification.

6.1 Matrice de confusion :

Pour mesurer les performances d'une classification, on prend un exemple de classification binaire qui prédit 2 classes, la classe 0 et la classe 1. Il s'agit d'évaluer par 4 types les éléments rassemblés sous forme d'un tableau dans une matrice de confusion **[8]** :

- Vrai positif : classe 1 correctement prédite.
- Vrai négatif : classe 0 correctement prédite.
- Faux positif : classe 1 mal prédite.
- Faux négatif : classe 0 mal prédite.

		Classe réelle	
		Classe 0	**Classe 1**
Classe prédite	**Classe 0**	**Vrai négatifs**	**Faux positifs**
	Classe 1	**Faux négatifs**	**Vrais positifs**

Tableau 1.2: Matrice de confusion

À partir de la matrice de confusion on peut dériver tout un tas de critères de performance. Voici quelques exemples de mesures de performance souvent utilisées :

- **Rappel** : le rappel ou sensibilité est le taux de vrais positifs, c'est à dire la Proportion d'éléments bien classés pour une classe donnée, ou bien la proportion de positifs que l'on a correctement identifiés :

$$\textbf{Rappel} = \frac{\textbf{Vrai positifs}}{\textbf{Vrai positifs} + \textbf{Faux négatifs}}$$

- **Précision :** la Proportion d'éléments bien classés par rapport au nombre d'éléments de la classe à prédire :

$$\textbf{précision} = \frac{\textbf{Vrai positifs}}{\textbf{Vrai posifits} + \textbf{Faux positifs}}$$

- **F-mesure :** Pour évaluer un compromis entre rappel et précision, on peut calculer la "F-mesure", qui est leur moyenne harmonique :

$$\textbf{F} - \textbf{mesure} = \textbf{2} \times \frac{\textbf{précision} \times \textbf{reppel}}{\textbf{précision} + \textbf{rappel}}$$

$$\text{F} - \text{mesure} = \frac{2 \text{ vrai positifs}}{2 \text{ vrai positifs} + \text{faux positifs} + \text{faux négatifs}}$$

Donc Mesure globale de performance d'un classifieur. :

K : nombre de classe

$$\textbf{rappel} = \frac{1}{K}\sum_{i=1}^{k}\frac{\textbf{vrai positifs}_i}{\textbf{vrai positifs}_i + \textbf{faux négatifs}_i}$$

$$\textbf{précision} = \frac{1}{k}\sum_{i=1}^{1}\frac{\textbf{vrai positifs}_i}{\textbf{vrai positifs}_i + \textbf{faux positifs}_i}$$

$$\textbf{F} - \textbf{mesure} = 2 \times \frac{\textbf{précision} \times \textbf{rappel}}{\textbf{précision} + \textbf{rappel}}$$

6.2 Classification : ROC et AUC :

Une courbe ROC (receiver operating characteristic) est un graphique représentant les performances d'un modèle de classification pour tous les seuils de classification. Cette courbe trace le taux de vrais positifs en fonction du taux de faux positifs **[8].**

Le **taux de vrais positifs (TVP)** est l'équivalent du : rappel

Le **taux de faux positifs (TFP)** est défini comme suit :

$$\textbf{taux faux positifis} = \frac{\textbf{faux positifis}}{\textbf{faux positifs} + \textbf{vrai négatifs}}$$

Une courbe ROC trace les valeurs TVP et TFP pour différents seuils de classification. Diminuer la valeur du seuil de classification (seuil de décision) permet de classer plus d'éléments comme positifs, ce qui augmente le nombre de faux positifs et de vrais positifs. La figure ci-dessous représente une courbe ROC classique.

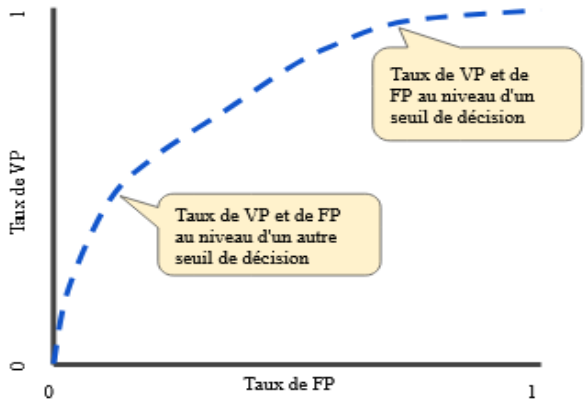


Figure 1.8: Taux de VP et de FP pour différents seuils de classification

AUC "aire sous la courbe ROC". Cette valeur mesure l'intégralité de l'aire à deux dimensions située sous l'ensemble de la courbe ROC (par calculs d'intégrales) de (0,0) à (1,1).

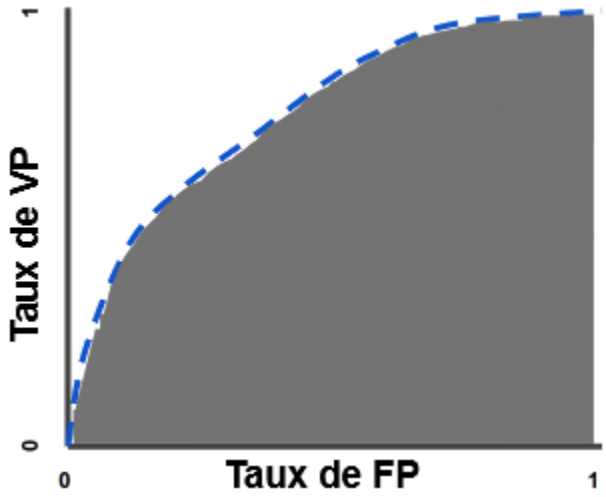


Figure 1.9: AUC (aire sous la courbe ROC)

7 Classification des images en fonction de machine Learning

Pour une machine, une image est un tableau de nombres indiquant la luminosité de chaque pixel, Comment par exemple une machine peut-elle identifier un chien ou une chaise dans le tableau de nombres d'une image quand l'apparence d'un chien ou d'une chaise ou d'objets qui les entourent peut varier infiniment ?

Il n'est pas facile d'écrire un programme qui solutionne cette problématique, C'est là qu'intervient le ML (machine Learning), très utilisé au sein d'importantes entreprises, qui l'utilisent depuis longtemps pour filtrer les contenus indésirables et sélectionner les informations intéressantes pour chaque utilisateur.

Le concept de l'apprentissage automatique peut être vu comme un système qui a une entrée par exemple une image, et une sortie qui peut représenter la catégorie de l'objet dans l'image, c'est ce qu'on appelle système de classification ou système de reconnaissance.

Dans la plupart du temps, l'apprentissage automatique est supervisé, la machine va s'entrainer plusieurs fois sur un ensemble d'images pour ajuster ses paramètres internes de manière à rapprocher sa sortie de la sortie désirée. La machine est alors capable par la suite de reconnaitre les images de l'entrainement et de les classifier correctement, elle

peut même classifier des images qu'elle n'a jamais vu durant la phase d'apprentissage c'est ce qu'on appelle la capacité de généralisation.

Deux étapes importantes pour la reconnaissance d'une image, sont l'extraction des caractéristiques (featur extraction) et le classifieur entrainable. L'extracteur de caractéristiques permet l'extraction de tableaux de nombres qui représentent l'image (l'extraction est programmée par data scientiste ou faite à la main Hand-designed feature Extraction) et le transforme en une série de nombres, un vecteur de caractéristiques dont chacun indique la présence ou l'absence d'un motif simple dans l'image.

Ce vecteur de données va entrer dans un classifieur, qui calcule une somme pondérée des caractéristiques, chaque nombre est multiplié par un poids, puis la somme est calculée, si la somme est supérieure à un seuil, la classe est reconnue, les poids sont modifiés lors de l'apprentissage pour bien optimiser la classification, Les premières méthodes de classification linéaire entraînable datent de la fin des années cinquante et sont toujours largement utilisées aujourd'hui **[9].**

8 Classification des images par les réseaux de neurones

Le problème de l'approche classique de la reconnaissance des images ce qu'on appelle "shallow Learning" est très difficile, et surtout peu utilisé dans ces derniers temps, C'est là qu'intervient l'apprentissage profond (deep Learning) qui est connu depuis la fin des années 1980, mais dont l'utilisation n'est que depuis 2012.

Le DL s'appuie sur un réseau de neurones artificiels s'inspirant du cerveau humain un peu comme l'avion est inspiré de l'oiseau.

L'architecture de réseau de neurones peut être vue comme un réseau multicouche, interconnectés par des poids entraînables. C'est ce qu'on appelle un réseau neuronal multicouche. Chacune recevant et interprétant les informations de la couche précédente

Le système est entraîné de bout en bout, à chaque exemple, tous les paramètres de tous les modules sont ajustés de manière à rapprocher la sortie produite par le système de la sortie désirée.

Pour entraîner le système, il faut connaitre combien ajuster chaque paramètre de chaque module. Pour cela il faut calculer un gradient, qui représente la quantité par laquelle

l’erreur en sortie augmentera ou diminuera lorsqu’on modifiera le paramètre, Le calcul de ce gradient se fait par la méthode de rétropropagation [10].

9 Conclusion

Dans ce chapitre nous avons présenté les concepts de la classification dans le domaine de l’imagerie, et nous avons également introduit l'apprentissage automatique et l'utilisation des réseaux de neurones dans le domaine de la classification.

Le chapitre suivant sera consacré précisément à l'utilisation des réseaux de neurones convolutifs dans la classification des images.

Chapitre 2 : Les Réseaux de Neurones Convolutionnels CNN

1 Introduction

Le Perceptron MultiCouche (PMC) est un des réseaux de neurones artificiels les plus utilisés actuellement, pour la classification supervisée sur toute la classification des images, avec extraction des caractéristiques c'est-à-dire extraction de tableau de nombres qui représente l'image. Deux approches principales sont alors nécessaires :

1) Extraire des caractéristiques par un algorithme écrit par un utilisateur (Hand-designed feature Extraction) et les présenter en entrée d'un réseau de neurones pour les classifier. Cette approche appelée « shallow learning » (Reconnaissance traditionnelle) est comme illustrée sur la Figure 2.1 :

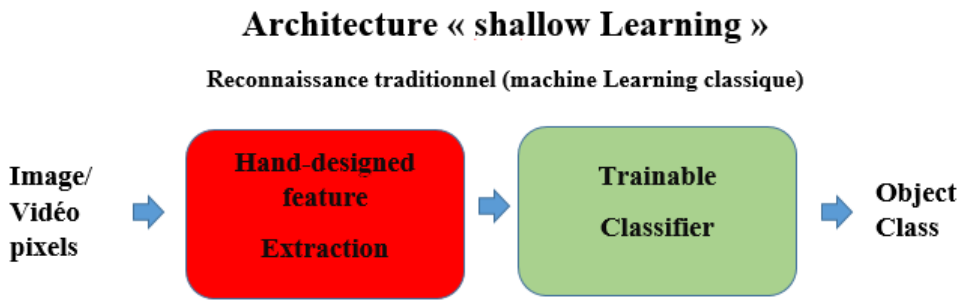


Figure 2.1: Shallow Learning

2) Présenter l'image à un ensemble de couches cachées, en entrée du réseau de neurones dite approche deep Learning et illustrée sur la Figure 2.2, et celui qui fait l'extraction des caractéristiques, même des caractéristiques qu'on peut pas les voir avec l'œil nu. L'image nécessite cependant d'être vectorisée, c'est à dire mise sous forme d'un vecteur dont la dimension est égale au nombre de pixels de l'image.

Figure 2.2: Deep architecture "Deep learning"

Dans le cadre de la classification des images, la classification par PMC peut avoir beaucoup de problèmes et de défauts, parmi ces défauts, le nombre énorme de paramètres (le nombre de connexion entre les différents neurones), si une image a une taille de 32 × 32, la dimension en entrée sera 1024 avec une couche cachée de 100 neurones par exemple, cela donne 102400 paramètres, et ce nombre va augmenter pour toutes les couches suivantes. Cette grande complexité et le nombre énorme de paramètres du réseau impose d'avoir un sur apprentissage qui proposera donc une mauvaise capacité de généralisation.

Un autre défaut du PMC pour une application des images : Peu ou pas d'invariance au décalage, à la mise à l'échelle et à d'autres formes de distorsion (pas d'invariance de translation) c'est-à-dire en cas de décalage d'une image les pixels changent ce qui donne aussi un changement de formes.

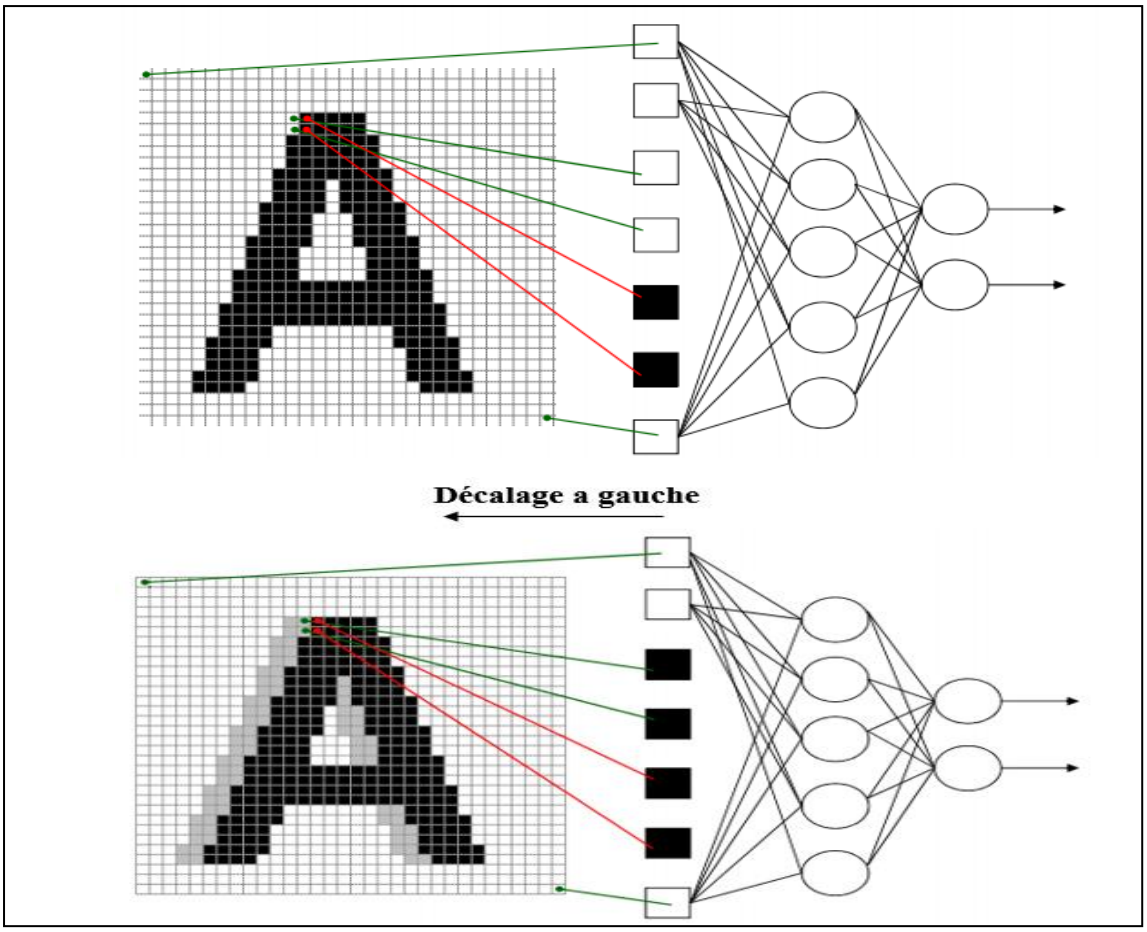


Figure 2.1 : Inconvénients des réseaux neuronaux PMC

Pour répondre aux défauts de PMC, il existe un type de réseaux de neurone artificiel CNN (Les réseaux de neurones convolutionnels) découvrir par Hubel et Wiesel **[11]** , et développer par Y. Lecun **[12],** ce réseau est inspiré de neurones en orientation dans le système visuel du chat**[11]** , il fait l'extraction des caractéristiques automatiquement avec l'application de la notion de partage des poids pour réduire le nombre de paramètres du réseau , cette application permet également de prendre en compte de manière forte les corrélations locales contenues dans une image (les pixels).

La première application réalisée avec les réseaux CNN dont l'apprentissage par propagation arrière (backpropagation), est la reconnaissance de caractères manuscrits qui a été effectuée par Y. Lecun **[13].** Dans ce chapitre on va présenter les réseaux de neurones convolutionnels.

2 Le réseau perceptron multicouche PMC

Le perceptron multicouche est un type de réseau neuronal artificiel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement, Figure 2.4. Il est capable de traiter des données qui ne sont pas linéairement séparable, et avec l'arrivée des algorithmes de rétro propagation, il devient le type de réseaux de neurones le plus utilisé.

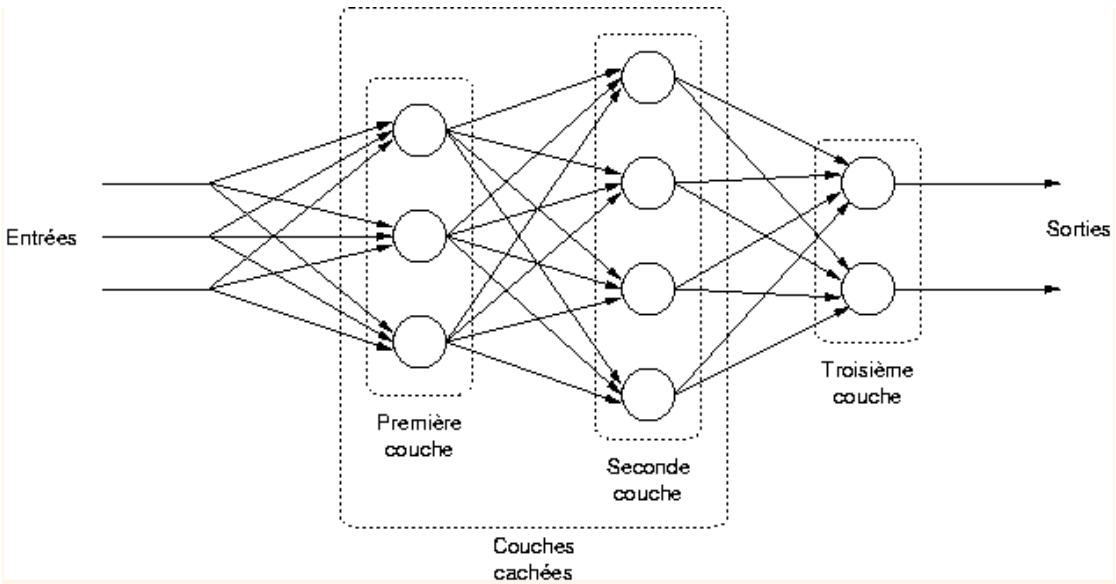


Figure 2.2: Schéma du réseau PMC

2.1 Modèle du perceptron

Un neurone de perceptron réalise un produit scalaire entre son vecteur d'entrée et un vecteur de paramètres appelé poids, y ajoute un biais, et utilise une fonction d'activation pour déterminer sa sortie, Figure 2.5 **[14]** :

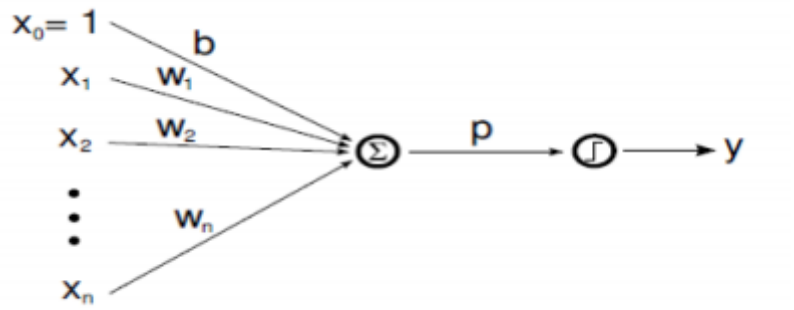


Figure 2.3: Modèle du perceptron

- P le potentiel :

$$\mathbf{p} = \sum \mathbf{W_n} \,.\mathbf{X_n} \qquad \textbf{(2.1)}$$

- Y la sortie :

$$\mathbf{y} = \mathbf{p}(\mathbf{X_n} \,.\mathbf{W_n} + \mathbf{b}) \qquad \textbf{(2.2)}$$

B : Biais Une valeur particulière peut être considérée comme une entrée supplémentaire dont la valeur est toujours à 1.

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

$$\mathbf{y} = \begin{cases} \mathbf{0} & \mathbf{si} \quad \mathbf{p} < 0 \\ \mathbf{1} & \mathbf{si} \quad \mathbf{p} \geq \mathbf{1} \end{cases} \qquad \textbf{(2.3)}$$

2.2 Fonctions d'activation

Tout apprentissage d'un modèle de réseau neurones se fait par la détermination des poids du réseau et de leur mise à jour pour que la sortie se rapproche de la sortie désirée, ceci se fait par un algorithme de rétro propagation du gradient, cet algorithme repose sur le calcul du gradient de sortie puis sur la rétro propagation de celui-ci à travers la fonction de seuil puis des poids. C'est pourquoi la fonction de seuil doit être dérivable. La fonction d'activation de Heaviside est donc remplacée par des fonctions d'activation qui sont dérivables, parmi les fonctions classiquement utilisées sont la fonction linéaire, la tangente hyperbolique (**Tanh)** , la fonction sigmoïde standard et la fonction relue :

- **Fonction d'activation linéaire** : C'est une fonction simple de la forme : f(p) = a. p ou bien f(p) =p. L'entrée passe à la sortie sans une très grande modification ou alors sans aucune modification. On reste ici dans une situation de proportionnalité **[15].**

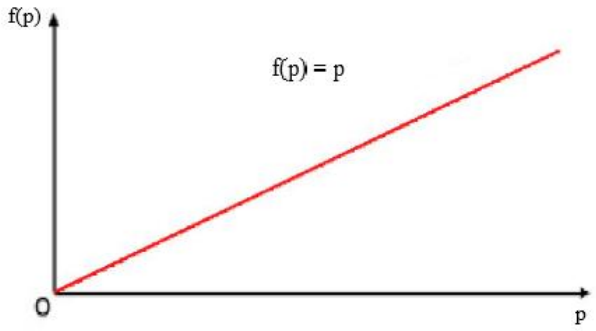


Figure 2.4 : Fonction d'activation linéaire

- **Sigmoïde :** Le premier but de la fonction est de réduire la valeur d'entrée pour entre 0 et 1. En plus d'exprimer la valeur sous forme de probabilité, si la valeur en entrée est un très grand nombre positif, la fonction convertira cette valeur en une probabilité de 1. A l'inverse, si la valeur en entrée est un très grand nombre négatif, la fonction convertira cette valeur en une probabilité de 0. D'autre part, l'équation de la courbe est telle que, seules les petites valeurs influent réellement sur la variation des valeurs en sortie. L'image ci-dessous représente la fonction Sigmoïde :

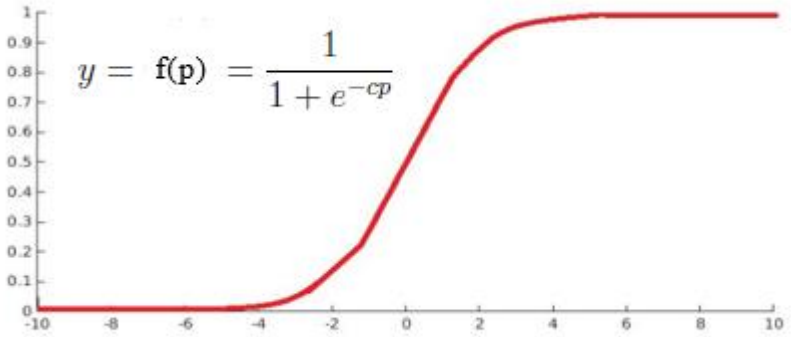


Figure 2.5 : Fonction d'activation sigmoïde

La fonction sigmoïde a plusieurs défauts, parmi ces défauts le fait que la fonction n'est pas centrée sur zéro, c'est à dire que des entrées négatives peuvent engendrer des sorties positives, et aussi elle influe assez faiblement sur les neurones par rapport à d'autres fonctions d'activation. Le résultat est souvent très proche de 0 ou de 1 causant la saturation de certains neurones**. [15]**

- **Tanh :** La fonction Tanh est également appelée "tangente hyperbolique". Cette fonction ressemble à la fonction Sigmoïde. La différence avec la fonction Sigmoïde est que la fonction Tanh produit un résultat compris entre -1 et 1. La fonction Tanh est en terme général préférable à la fonction Sigmoïde car elle est centrée sur zéro. Les grandes entrées négatives tendent vers -1 et les grandes entrées positives tendent vers 1. (Figure 8)

Mis à part cet avantage, la fonction Tanh possède les mêmes autres inconvénients que la fonction Sigmoïde**. [15]**

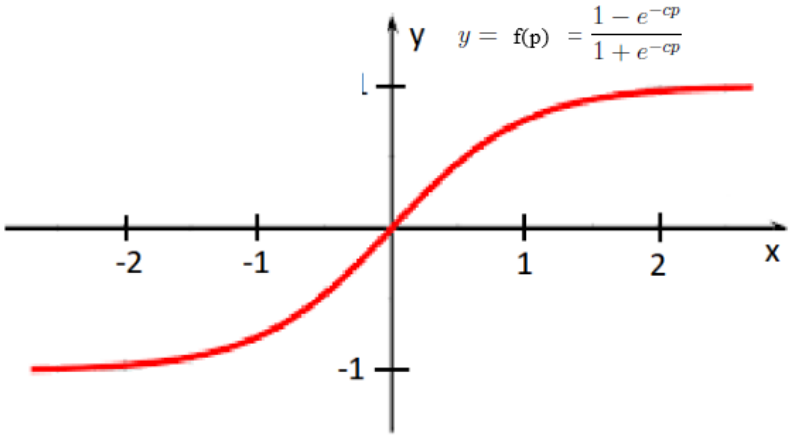


Figure 2.6 : Fonction d'activation tangente hyperbolique

- **Relu (Unité linéaire rectifiée) :** La fonction ReLU est interprétée par la formule : f(x) = max (0, x). Si l'entrée est négative la sortie est 0 et si elle est positive alors la sortie est x. Cette fonction d'activation augmente considérablement la convergence du réseau et ne sature pas.

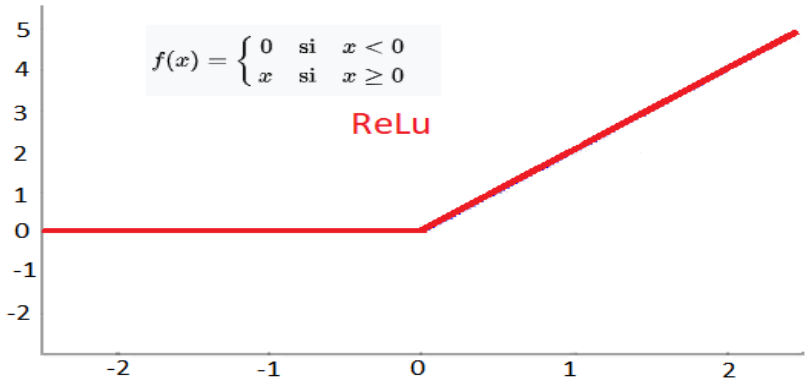


Figure 2.7 : Fonction d'activation relu

2.3 La notion de rétropropagation du gradient :

L'équation de propagation peut être définie comme :

$$p = W \,.\, x \qquad (2.4)$$

$$Y = \theta(p) \qquad (2.5)$$

La règle d'apprentissage utilisée pour mettre à jour les poids du neurone se fait de manière à ce que l'erreur E soit minimale :

$$E = \frac{1}{2}\,(y - z)^2 \qquad (2.6)$$

Où **y** représente la sortie d'un perceptron et **z** la sortie désirée. Donc le gradient de l'erreur en sortie :

$$\frac{dE}{dy} = y - z \qquad (2.7)$$

La rétropropagation du gradient, c'est l'estimation des gradients de E par rapport aux poids en progressant de la sortie vers l'entrée du réseau de neurone :

$$\frac{dE}{dp} = \frac{dE}{dy}\frac{dy}{dp}$$

$$= \frac{dE}{dy}\,\theta(p)'$$

$$\frac{dE}{dx} = \frac{dE}{dp}\frac{dp}{dx}$$

$$= \frac{dE}{dp}\, w$$

Dans un apprentissage supervisé, les poids vont être corrigés de façon itérative.

A l'itération k, étant données les poids, w_n^k, composantes du vecteur w^k , on sélectionne aléatoirement un exemple x^k dont on connait la sortie y^k et soi z^k la sortie donnée par le neurone , l'utilisation de l'algorithme de descente du gradient implique que :

$$w_n^{k+1} = w_n^k + \alpha^k \frac{dE}{dw_n}(x^k, w^k)$$

Taux d'apprentissage

le taux d'apprentissage α^k peut améliorer significativement la convergence de l'algorithme . En générale, il est choisi 0.001.

3 Différents types de réseaux de neurones artificiels

3.1 Adaline (adaptive linéaire neurones) :

Le réseau ADALINE est proche du modèle perceptron, seule sa fonction d'activation est différente puisqu'il utilise une fonction linéaire. Afin de réduire les parasites reçus en entrée, les réseaux ADALINE utilisent la méthode des moindres carrés.

Le réseau réalise une somme pondérée de ses valeurs d'entrées et y rajoute une valeur de seuil prédéfinie. La fonction de transfert linéaire est ensuite utilisée pour l'activation du neurone. Lors de l'apprentissage, les coefficients synaptiques des différentes entrées sont modifiés en utilisant la loi de Widrow-Hoff. Ces réseaux sont souvent employés en traitement de signaux, notamment pour la réduction de bruit.

3.2 PMC à fonctions radiales (RBF) :

La Couche cachée est à fonction d'activation radiale (e.g. gaussienne), l'idée est de paver l'espace des entrées avec ces "champs récepteurs"

La Couche de sortie est une combinaison linéaire sur la couche cachée **[16].**

Il y a quatre paramètres principaux à régler dans un réseau RBF :

- Nb de couches cachées.
- Position des centres des champs récepteurs.
- Diamètre des champs récepteurs.

- Poids vers la couche de sortie (moyenne pondérée).

3.3 Kohonen :

Ce réseau de neurones peut être considéré comme dynamique, des neurones peuvent être détruits et créés, le réseau n'a pas de taille fixe. Généralement ce réseau est appelé carte de kohonen, en effet ce réseau est représenté à plat comme une grille rectangulaire à 1, 2, 3 ou 4 dimensions.

Les applications sont multiples : sélection de données représentatives dans une grande base de cas, compression d'images, diagnostic de pannes, optimisation combinatoire (dont le fameux "voyageur de commerce", modélisation de la cartographie des aires visuelles....

4 L'apprentissage profond Deep learning

4.1 Qu'est-ce que l'apprentissage profond (DL) ?

A présent que nous avons précisé comment fonctionnent les réseaux de neurones de manière générale, nous allons aborder le domaine du Deep Learning. Cette famille d'algorithmes a permis de faire des progrès importants dans les domaines de la classification des images et du traitement du langage par exemple. Les modèles de Deep Learning sont bâtis sur le même modèle que les perceptrons multicouches précédemment décrits. Cependant, il convient de souligner que les différentes couches intermédiaires sont plus nombreuses. Chacune des couches intermédiaires va être subdivisée en sous partie, traitant un sous problème, plus simple et fournissant le résultat à la couche suivante, et ainsi de suite. [17]

4.2 Algorithmes de deep learning :

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

- **Les réseaux de neurones dits convolutifs (CNN)** : Ces réseaux reposent sur des filtres de convolution (matrices numériques). Les filtres sont appliqués aux entrées avant que celles-ci ne soient transmises aux neurones.

  Ces réseaux de neurones sont utiles pour le traitement la classification et la prévision d'images.
- **Réseau de neurones récurrent (RNN) :** Ces réseaux nous aident à prévoir les séries chronologiques dans les applications commerciales et à prévoir les mots dans les applications de type chatbot. Ils peuvent fonctionner avec différentes longueurs d'entrée et de sortie et nécessitent une grande quantité de données.
- **Les auto encodeurs :** Un auto-encodeur, est un réseau de neurones artificiels utilisé pour l'apprentissage non supervisé de caractéristiques discriminantes.

L'objectif d'un auto-encodeur est d'apprendre une représentation d'un ensemble de données, généralement dans le but de réduire la dimension de cet ensemble.
Les auto encodeurs sont appliqués principalement à la détection d'anomalie (par exemple pour détecter la fraude en banque ou bien pour trouver des anomalies dans une ligne de production industrielle.

5 Les réseaux de neurones convolutionnels

Le modèle le plus utilisé pour la classification des images est le réseau de neurones covolutionnels CNN, en raison de son haut taux de reconnaissance et de son exécution rapide, ainsi que de ses performances qui fonctionnent comme un extracteur de caractéristiques des images. L'image passe à travers un ensemble de couches tel que les couches de convolution et les couches du max pooling, certains filtres réduisent la résolution de l'image par l'opération du maximum local, et après chaque couche de convolution on obtient une nouvelle image avec moins de caractéristiques par rapport à l'image précédente appelée carte de convolution.

5.1 Réseaux de neurones convolutionnels et PMC :

Les perceptrons multicouches (PMC) ont des difficultés à gérer des images de grande taille, en raison de la croissance exponentielle du nombre de connexions avec la taille de l'image, du fait que chaque neurone est « totalement connecté » à chacun des neurones de la couche précédente et suivante.

Les réseaux de neurones convolutifs limite au contraire le nombre de connexions entre un neurone et les neurones des couches adjacentes ce qui diminue drastiquement le nombre de paramètres à apprendre **[18]**

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 PMC, les CNN ont les traits distinctifs suivants :

- **Connectivité locale** : grâce au champ récepteur qui limite le nombre d'entrées du neurone, tout en conservant l'architecture PMC, 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 PMC). **[18]**

- **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 ». **[18]**
- **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. **[18]**

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**. [18]**

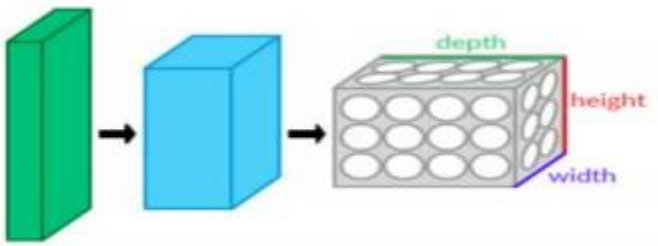


Figure 2.10: Une couche du CNN en 3 dimensions. (Vert = volume d'entrée, bleu = volume du champ récepteur, gris = couche de CNN, cercles = neurones artificiels indépendants)

5.2 Architecture du réseau de neurone convolutionnel :

Le réseau de neurone CNN est une solution pour remédier aux défauts du réseau PMC, tel que le nombre de connexion (nombre de paramètres) et l'invariance à la translation. Le réseau de neurones convolutifs, ConvNets ou CNNs, est utilisé principalement pour faire la classifications d'images, et la reconnaissance d'images, c'est les domaines où les CNN sont largement utilisés.

La classification des images CNN prend une image d'entrée, la traite et la classe dans certaines classes (par exemple chien, chat). Les ordinateurs voient une image d'entrée comme

un tableau de nombre ou un tableau de pixels et cela dépend de la résolution de l'image, H x L x D, H = hauteur, L = largeur, D = dimension. Par exemple, une image d'un tableau 32 x 32 x 3 de matrice RVB (3 se réfère aux valeurs RVB) et une image d'un tableau 32 x 32 x 1 de matrice d'image en niveaux de gris.

Les neurones d'un CNN sont divisés en une structure tridimensionnelle (longueur, largeur et profondeur), le nombre d'entrées de neurones limité par le champ de récepteurs, chaque ensemble de neurones analysant une petite région ou caractéristique de l'image.

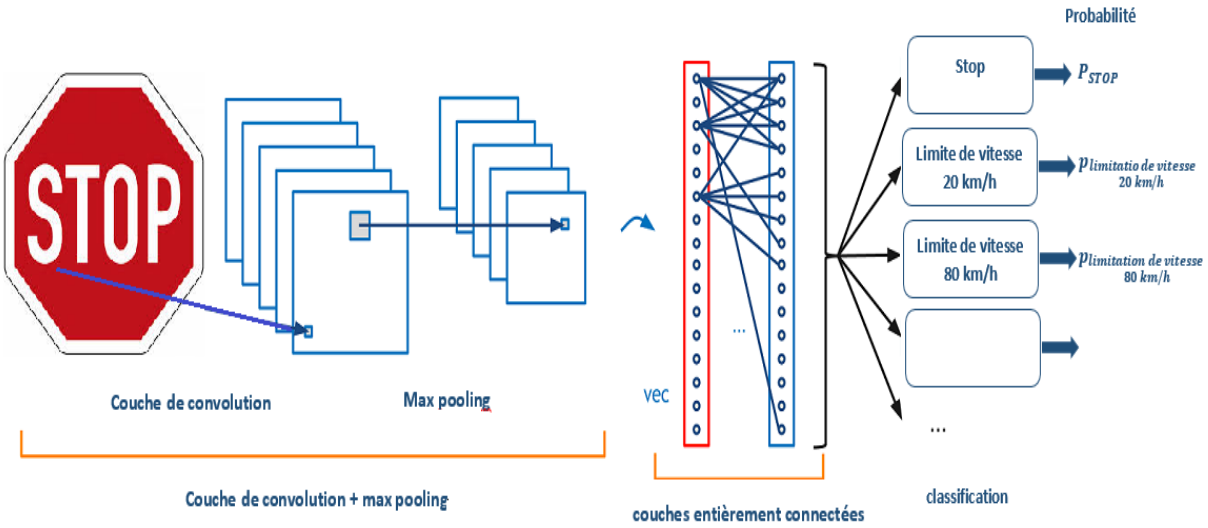


Figure 2.11 : Architecture d'un réseau de neurone convolutionnel

Techniquement, les modèles CNN d'apprentissage en profondeur à entrainer et à tester, chaque image d'entrée passent à travers une série de couches de convolution avec filtres (Kernals), couches du max Pooling, ainsi que par les couches entièrement connectées (Fully Connected) pour combiner toutes les caractéristiques de l'image d'entrée. La fonction Softmax est alors appliquée pour classer un objet avec des valeurs probabilistes entre 0 et 1. Sans oublier que le nombre du neurones en sortie est égale au nombre de classes dans la base de données.

Le bloc de construction du réseau de neurones convolutifs est formée par un empilement de couches de traitement : **[18]**

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

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

5.2.1 Couche de convolution (CONV)

La convolution est la première couche à extraire des entités d'une image d'entrée. La convolution préserve la relation entre les pixels (invariance de translation). Il s'agit d'une opération mathématique pour calculer la taille spatiale du volume de sortie (la sortie de couche de convolution) qui prend deux entrées, la taille du volume d'entrée Wi (matrice d'image si par exemple une image de dimension 32*32*3, donc Wi= 32) et un filtre k ou noyau (surface de traitement dimension de karnel). Un autre paramètres le pas S avec lequel ils sont appliqués (souvent, on considère un pas S=1), et la taille de la marge P (il est commode de mettre des zéros à la frontière du volume d'entrée) La formule pour calculer le nombre de neurones du volume de sortie est : **[18]**

$$W_0 = \frac{W_i - k + 2p}{s} + 1$$

NB : si on souhaite un volume de sortie de même taille que le volume d'entrée "connectée localement". On prend la marge de la manière suivante :

$$\mathrm{p} = \frac{\mathrm{K} - 1}{2}$$

Vérification :

$$W_o = \frac{Wi - k + 2 \times \frac{K-1}{2}}{S} + 1$$

Avec S = 1 :

Donc : $$W_0 = W_i$$

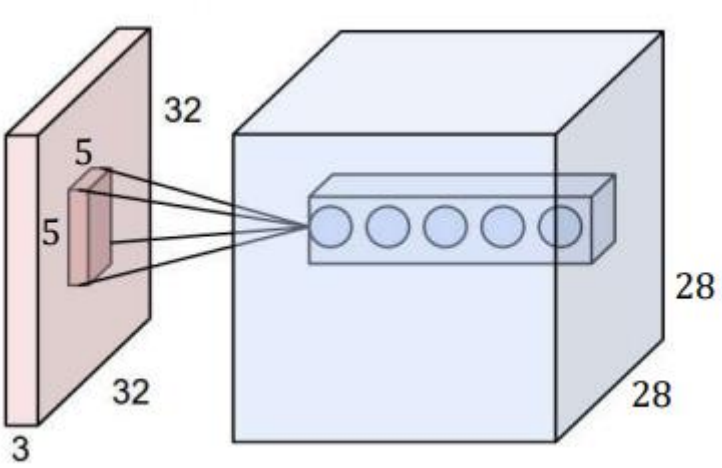


Figure 2.12 : Calcul le nombre de neurones du volume de sortie

Cercles : Ensemble de neurones. Bleu : créant la profondeur d'une couche de convolution
Rouge : Ils sont liés à un même champ récepteur.

$W_i = 32$, $= 5$, $S = 1$, P =0

$$W_0 = \frac{32 - 5}{1} + 1 = 28$$

Donc La dimension ou bien le nombre de neurones du volume de sortie et : 28*28*3 (l'image est RGB) ou 28*28*1 (au niveau de gris).

5.2.2 Couche de pooling (POOL) :

Le pooling est une forme de sous-échantillonnage de l'image, L'image d'entrée est découpée en une série de rectangles, chaque rectangle possède un ensemble de pixels, Le pooling réduit la taille spatiale d'une image intermédiaire, réduisant ainsi la quantité de paramètres et de calcul dans le réseau, en général la couche de pooling est insérée entre deux couches convolutives successives d'une architecture CNN pour contrôler l'overfitting (sur-apprentissage). Cette opération crée aussi une forme d'invariance par translation.

La couche pooling la plus utilise c'est la couche de max pooling, elle prend comme sortie le plus grand élément de la carte des caractéristiques (la sortie de la couche de convolution), il existe aussi d'autres types de pooling comme « average pooling » qui prend la moyenne entre les valeurs de patch, on pourra utiliser aussi un pooling stochastique. Le 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 2x2).** Il est aussi possible d'éviter la couche de pooling mais cela implique un risque de sur-apprentissage plus important**. [18]**

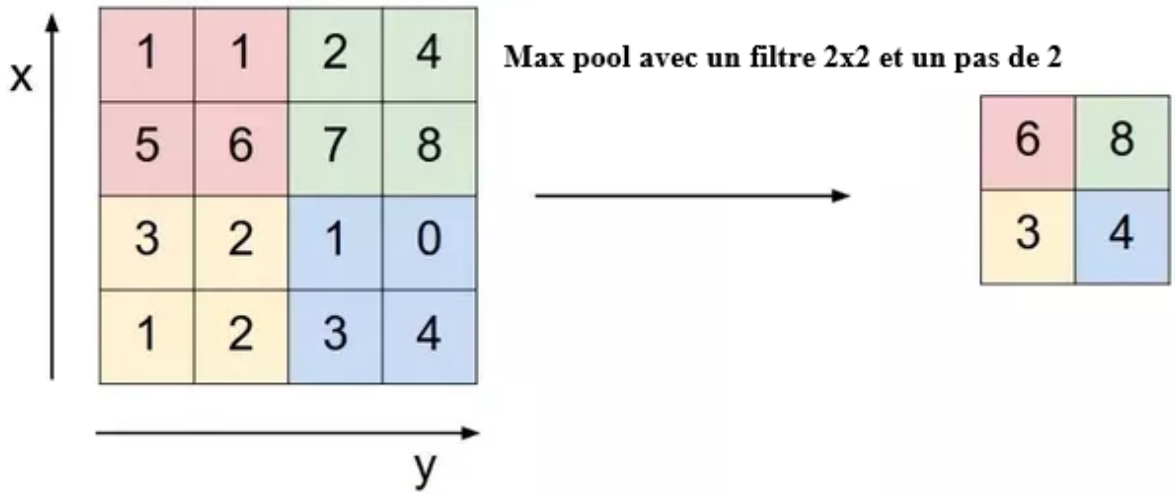


Figure 2.13 : Pooling avec un filtre 2x2 et un pas de 2

5.2.3 Couches 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 (abréviation de Unités Rectifié linéaires) : F(x)=max(0,x) Cette fonction force les neurones à retourner des valeurs positives.**[18]**

Il existe d'autres fonctions non linéaires telles que tanh ou sigmoïde qui peuvent également être utilisées à la place de ReLU. La plupart des data scientists utilisent ReLU car ReLU en termes de performances est meilleur que les deux autres.

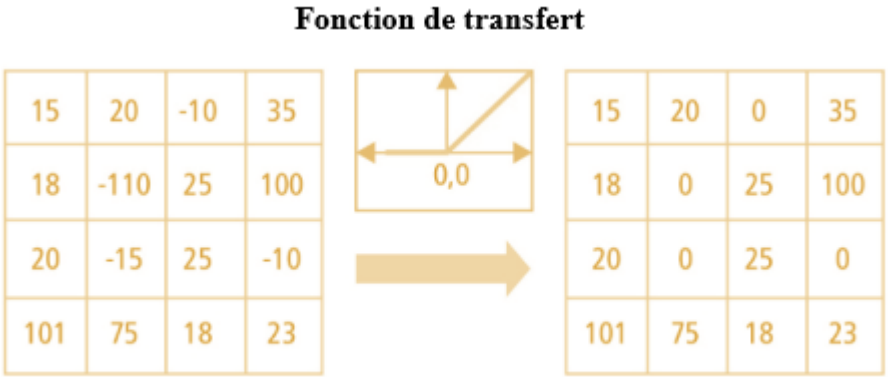


Figure 2.14 : Opération relu

5.2.4 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).

La couche que nous appelons couche FC, nous avons aplati notre matrice en vecteur et l'avons introduite dans une couche entièrement connectée comme un réseau de neurones.

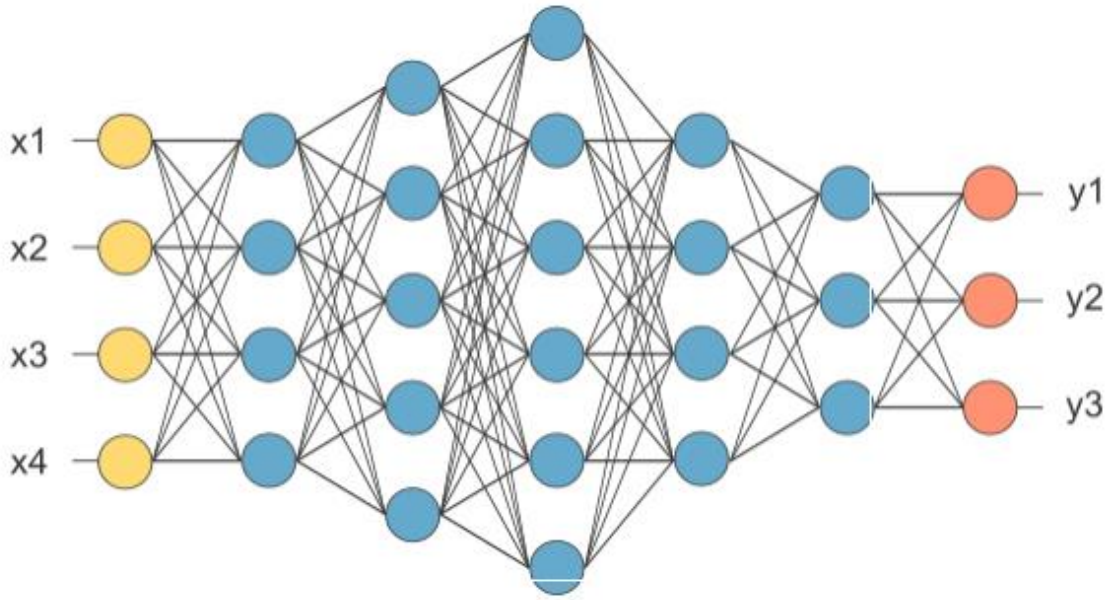


Figure 2.15 : Après couche de pooling aplatie en tant que couche FC

Dans le diagramme ci-dessus, la matrice de la carte des caractéristiques sera convertie en vecteur (x1, x2, x3…). Avec les couches entièrement connectées, nous avons combiné ces fonctionnalités pour créer un modèle. A la fin, on a une fonction d'activation telle que softmax ou sigmoïde pour classer les sorties.

5.2.5 Couche de perte (LOSS) :

La couche de perte spécifie comment l'entrainement du réseau pénalise l'écart entre le signal prévu et réel. Elle est normalement la dernière couche dans le réseau. Diverses fonctions de perte adaptées à différentes tâches peuvent y être utilisées. La fonction « Softmax » permet de calculer la distribution de probabilités sur les classes de sortie. **[18]**

6 Exemple de modèles de CNN

La forme d'une architecture la plus utilisée pour implémenter un réseau de neurones convolutifs est généralement une couche de convolution et relu, suivie par des couches pool, et cette forme va se répéter pour réduire la taille d'une image, et enfin la dernière couche empilée est la couche entièrement connectée reliée vers la sortie.

Voici quelques architectures communes de réseau de neurones convolutifs qui suivent ce modèle :

- **INPUT -> FC**. Implémente un classifieur linéaire
- **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.**

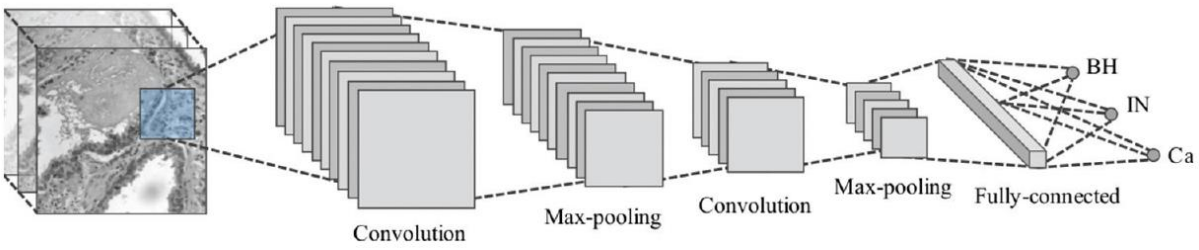


Figure 2.16 : Exemples de modèles de CNN

7 Choix des hyperparamètres

Malheureusement, tous les aspects des CNNs ne sont pas aussi intuitifs à apprendre et comprendre que ce que nous avons vu jusqu'à présent. Ainsi, il y a toujours une longue liste de paramètres qui doivent être définis manuellement pour permettre au CNN d'avoir de meilleurs résultats. Donc Il faut prendre en considération les notions de nombre de filtres, et leur forme et la forme du max pooling :

7.1 Nombre de filtres

Le nombre de filtres est le nombre de neurones, chaque neurone effectuant une convolution différente sur l'entrée de la couche (plus précisément, les poids d'entrée des neurones forment des noyaux de convolution).

Une carte de caractéristiques est le résultat de l'application d'un filtre (vous avez donc autant de cartes de caractéristiques que de filtres), et sa taille est le résultat de la taille de la fenêtre/du noyau de votre filtre et de la foulée (le pas). **[18]**

Pour mieux comprendre, des filtres de convolution différents sont appliqués à l'image d'entrée, ce qui donne des cartes de caractéristiques différentes (la sortie des filtres). Chaque pixel de chaque carte de caractéristiques est une sortie de la couche de convolution.

Par exemple, si vous avez des images d'entrée 32x32 et un calque convolutif avec 20 filtres 5x5 et la foulée (le pas) à 1, vous obtiendrez 20 cartes de caractéristiques 28x28 à la sortie de ce calque. Notez que ceci est présenté au calque suivant comme un volume de largeur = hauteur = 28 et profondeur = num_channels = 20.

7.2 Forme du filtre :

Ils sont généralement choisis en fonction de l'ensemble de données. Les meilleurs résultats sur les images (32x32) sont habituellement dans la gamme de 5x5 sur la première couche, tandis que les ensembles de données d'images naturelles, ont tendance à utiliser de plus grands filtres de première couche de 12x12 ou bien 15x15.

7.3 Forme de max pooling :

Les valeurs typiques sont 2x2. De très grands volumes d'entrée peuvent justifier un pooling 4x4 dans les premières couches.

Cependant, le choix de formes plus grandes va considérablement réduire la dimension du signal, et peut entraîner la perte de trop d'informations.

8 Méthodes de régularisation

Pour améliorer les performances et la capacité de généralisation d'un algorithme d'apprentissage et pour éviter le sur-apprentissage il y'a des méthodes de régularisation à utiliser :

8.1 Empirique :

8.1.1 Dropout :

Le concept de FC (couches entièrement connectées) crée un problème exponentiel de mémoire appelé "overfitting" c'est à dire une sur-connexion qui conduit au sur-apprentissage ralentissant le traitement de l'information. Pour prévenir cela, La méthode du dropout consiste

à « désactiver » des neurones aléatoirement (avec une probabilité prédéfinie, souvent un neurone sur deux), pendant la phase d'apprentissage, avec moins de neurones, le réseau est plus réactif et peut donc apprendre plus rapidement. Après la phase d'apprentissage les neurones "désactivés" sont "rallumés" (avec leurs poids originaux).

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.

8.1.2 DropConnect :

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

8.1.3 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. **[18]**

8.2 Explicite :

8.2.1 Taille de 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. **[18]**

8.2.2 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. **[18]**

- **La régulation 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 régulation 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).

9 Paramètres du réseau CNN

Voyons d'abord comment le nombre de paramètres pouvant être appris est calculé pour chaque type de couche, puis calculons le nombre de paramètres dans notre réseau (chapitre 3). **[19]**

- **Couche d'entrée** : Tout ce que la couche d'entrée fait est de lire l'image d'entrée. Par conséquent, aucun paramètre ne peut être appris ici.
- **Couches convolution** : considérons une couche de convolution qui prend à **l** les cartes de caractéristiques en entrée et à **k** les cartes de caractéristiques en sortie. La taille du filtre est **n x m**. On prend l'exemple de la figure 2.17 :

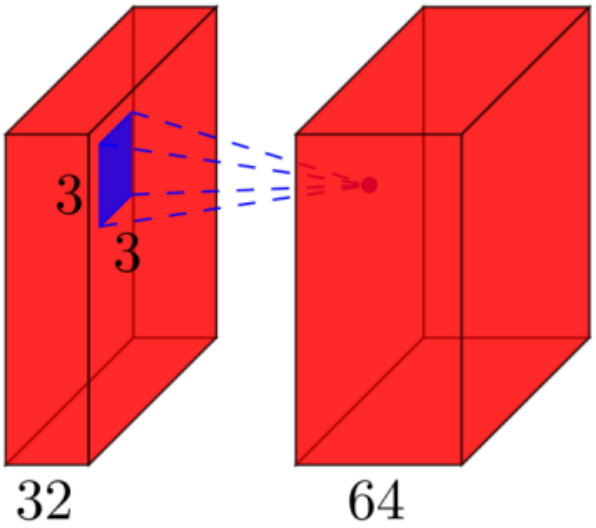


Figure 2.17 : Carte de caractéristiques en entrés et en sortie

Ici, l'entrée a des cartes de caractéristiques **l=32** en entrée, des cartes de fonction **k=64** en sortie et la taille du filtre est de **n=3 X m=3**. Il est important de comprendre que nous n'avons pas simplement un filtre 3x3, mais en réalité un filtre 3x3x32, car notre entrée est à 32 dimensions. Et nous apprenons 64 filtres 3x3x32 différents. Ainsi, le nombre total de poids est de **n*m*k*l.** Ensuite, il y a aussi un terme de biais pour chaque carte de caractéristiques, nous avons donc un nombre total de paramètres de **(n*m*l+1) *k.**

- **Couche de max pool** : par exemple. Procédez comme suit : "remplacer un quartier 2x2 par sa valeur maximale". Il n'y a donc aucun paramètre que nous pourrons apprendre dans une couche de max pool.
- **Couche entièrement connectée :** dans une couche entièrement connectée, toutes les unités d'entrée ont un poids distinct pour chaque unité de sortie. Pour **n** entrées et **m** sorties, le nombre de poids est de **n*m**. De plus, nous avons un biais pour chaque nœud de sortie, nous sommes donc à des paramètres **(n+1) *m.**
- **Couche de sortie** : la couche de sortie est une couche entièrement connectée, donc les paramètres sont au nombre de (n+1) *m, Où n est le nombre d'entrées et m est le nombre de sorties.

10 Conclusion

Nous avons consacré ce chapitre à la présentation des réseaux de neurones convolutionnels capables d'extraire des caractéristiques d'images présentées en entrée, et de les classifier. Nous avons parlé aussi sur des avantages du réseau CNN par rapport au réseau multicouche PMC. Un avantage majeur est l'utilisation d'un poids unique associé aux signaux entrants pour tous les neurones d'un même noyau de convolution (partage du poids), Cette méthode réduit l'empreinte mémoire, améliore les performances et permet une invariance du traitement par translation.

Nous avons mentionné aussi les hyper paramètres du réseau et qui sont difficiles à évaluer avant l'apprentissage, le nombre de couches, les nombre de neurones par couche ou encore les différentes connexions entre couches, on peut déterminer ces éléments par une bonne intuition ou par une succession de tests/calcul d'erreurs.

Dans le chapitre suivant, on va présenter notre modèle de CNN implémenté et appliqué pour la classification des panneaux de signalisation routière, ensuite on va interpréter les résultats obtenus dans la phase d'apprentissage et de test et les discuter.

Chapitre 3 : Implémentation

1 Introduction

Dans ce chapitre, on va définir l'architecture de notre modèle implémenté qu'on a créé et par la suite on va appliquer ce modèle sur la base d'images **GTSR**. Pour cela, on va travailler avec les bibliothèques Tensorflow et Keras pour l'apprentissage et la classification. Afin d'améliorer les performances des modèles et faciliter le travail, on va utiliser quelques techniques simples et efficaces comme data augmentation et dropout. On donnera également des informations telles que combien de temps faut-il pour s'entraîner et combien de données de chaque classe sont nécessaires pour avoir un bon modèle de classification.

Comme tout processus de création de modèle d'apprentissage automatique, nous exécuterons les étapes définies ci-dessous :

- Explorez et visualisez l'ensemble de données
- Pré-traiter et augmenter l'ensemble de données, si nécessaire
- Développer un modèle CNN
- Former et valider le modèle
- Optimiser le modèle en expérimentant différents hyper-paramètres
- Tester le modèle avec l'ensemble de données de test

2 Logiciels et librairies Utilisés dans l'implémentation

2.1 Python :

Python est un langage de programmation interprété, multiparadigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions. Python est un langage simple, facile à apprendre et permet une bonne réduction du cout de la maintenance des codes. Les bibliothèques (packages) python encouragent la modularité et la réutilisabilité des codes. Python et ses bibliothèques sont disponibles (en source ou en binaire) sans charges pour la majorité des plateformes et peuvent être redistribués gratuitement

2.1.1 Python et version utilisée :

Python est le langage préféré des développeurs de l'intelligence artificielle grâce à la simplicité de sa syntaxe. Les syntaxes développées par Python sont simples et peuvent être apprises facilement. Le langage de programmation est moins compliqué que Java ou C++ en ce qui concerne le machine learning. La version utilisée est la version Python 3.8.0.

2.2 TensorFlow :

TensorFlow est une bibliothèque open source de Machine Learning créée par GOOGLE en novembre 2015, TensorFlow n'a cessé de gagner en popularité, pour devenir très rapidement l'un des frameworks le plus utilisé , permettant de développer et d'exécuter des applications de Machine Learning et de Deep Learning, son nom est notamment inspiré du fait que les opérations courantes sur des réseaux de neurones sont principalement faite via des tables de données multidimensionnelles, appelées tenseurs, comme par exemple un Tensor à deux dimensions étant une matrice.

2.2.1 Installation TensorFlow :

Sous Windows, TensorFlow peut être installé via "pip" ou "anaconda". Python est livré avec le gestionnaire de packages pip, donc si vous avez déjà installé Python, vous devriez également avoir pip. Le package peut installer TensorFlow avec ses dépendances.

Anaconda est également une excellente option pour installer TensorFlow, mais il n'est pas livré avec Python comme pip, nous devons donc le télécharger et l'installer séparément.

Pour obtenir le gestionnaire de packages pip, nous devons tout d'abord installer Python. Nous téléchargeons la dernière version de Python (python 3.8.0) sur le site officiel de Python et nous l'installons.

Une fois l'installation terminée, on va vérifier la version de pip en cours d'exécution sur notre système. Pour ce faire, on va taper :

```
$ pip3 --version
```

Comme nous avons téléchargé la dernière version 3.8, le gestionnaire de package est pip3, au lieu de pip, qui a été utilisé avec Python 2.7.

Pour l'installation de TensorFlow, on va dans le menu démarrer de notre machine Windows, nous recherchons "cmd", et à l'aide du clic droit nous choisissons "Exécuter en tant qu'administrateur, une commande est exécutée pour installer TensorFlow.

Voici la commande :

```
$ pip3 install --upgrade tensorflow
```

2.3 Keras :

La bibliothèque Keras permet d'interagir avec les algorithmes de réseaux de neurones profonds et d'apprentissage automatique, notamment Tensorflow, Theano, Microsoft Cognitive Toolkit ou PlaidML.

Conçue pour permettre une expérimentation rapide avec les réseaux de neurones profonds, la bibliothéque Keras se concentre sur son ergonomie, sa modularité et ses capacités d'extension. Elle a été développée dans le cadre du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). Elle a été initialement écrite par François Chollet.

2.3.1 Installation keras :

Nous pouvons également installer Keras depuis PyPI:

```
$ pip3 install keras
```

2.4 Scikit-learn :

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

2.5 Matplotlib :

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique comme NumPy. Matplotlib est distribuée librement et gratuitement sous une licence de style BSD. Sa version stable actuelle est compatible avec la version 3 de Python.

2.6 Configuration utilisé dans l'implémentation :

- Un pc portable HP i5 CPU 2.40 GHZ.
- Carte graphique intel (r) hd graphics 520.
- RAM de taille 8 GO.
- Taille disque dur 256 GO.
- Système d'exploitation Windows 10 64 bits.

3 La base d'images GTSR

La base de données GTSR (mydata) représente l'ensemble de données de 35000 images et se compose de 43 classes (images de panneaux de signalisation uniques), chaque classe représente un certain type de panneaux (par exemple classe 14 : STOP). Ils sont disponibles sur le lien suivant : https://www.data.gov/ .

Figure 3.1 : Les classes des panneaux de signalisation

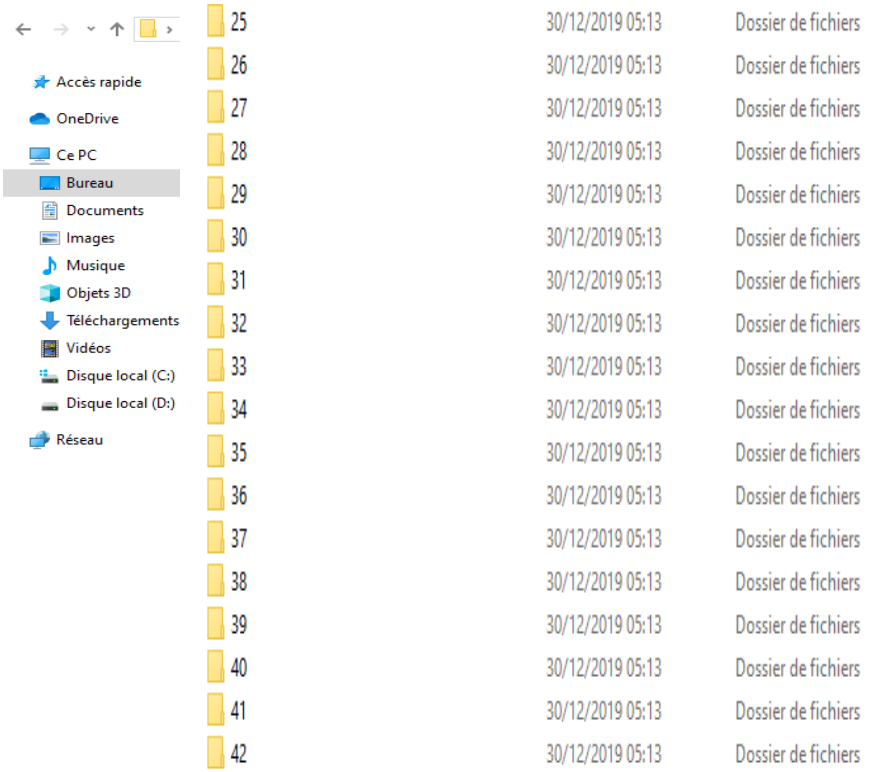


Figure 3.2 : MyData

On a créé aussi un dossier Excel qui représente les étiquettes, nous avons les noms de ces classes. Dans la base de données mydata, on a les identifiants de 0 jusqu'à 42 mais dans nos étiquettes nous avons le nom de chaque ID donc 0 représente la limite de vitesse de 20 alors que 14 représente stop et ainsi de suite.

	A
1	ClassId,Name
2	0,Limitation de vitesse (20km/h)
3	1,Limitation de vitesse (30km/h)
4	2,Limitation de vitesse (50km/h)
5	3,Limitation de vitesse (60km/h)
6	4,Limitation de vitesse (70km/h)
7	5,Limitation de vitesse (80km/h)
8	6,Fin de limitation (80km/h)
9	7,Limitation de vitesse (100km/h)
10	8,Limitation de vitesse (120km/h)
11	9,Interdiction de depasser
12	10,Interdiction de depasser pour les poids lourds
13	11,Intersection avec une route dont les usagers doivent ceder le passage
14	12,Route prioritaire
15	13,Ceder le passage
16	14,Stop
17	15,Pas de vehicules
18	16,Interdiction aux poids lourds
19	17,Entree interdite
20	18,Attention danger
21	19,Virage a gauche
22	20,Virage a droite
23	21,Deux virages

	A
22	20,Virage a droite
23	21,Deux virages
24	22,Route cahoteuse
25	23,Route glissante
26	24,La route se retrecit a droite
27	25,Travaux routiers
28	26,Feux de circulation
29	27,Pieton
30	28,Enfants traversant
31	29,Traversee de velos
32	30,Danger de la glace et de la neige a la rue
33	31,Traversee des animaux sauvages
34	32,Fin de toutes les limites de vitesse et de depassement
35	33,Tournez a droite
36	34,Tournez a gauche
37	35,Devant seulement
38	36,Allez tout droit ou a droite
39	37,Allez tout droit ou a gauche
40	38,Restez a droite
41	39,Restez a gauche
42	40,Rond-point obligatoire
43	41,Fin du non-passage
44	42,Fin du non passage-pour les poids lourds

Figure 1.3 : les étiquettes

3.1 Exploration et visualisation de l'ensemble de données :

3.1.1 Prétraitement des images :

Pour mieux faciliter l'opération d'extraction des caractéristiques par des couches de réseaux de neurones, il existe des étapes de prétraitement, donc nous appliquons dans un premier temps deux étapes de prétraitement à nos images :

- **Niveaux de gris :**

Nous convertissons des images à 3 canaux (RGB) en niveau de gris :

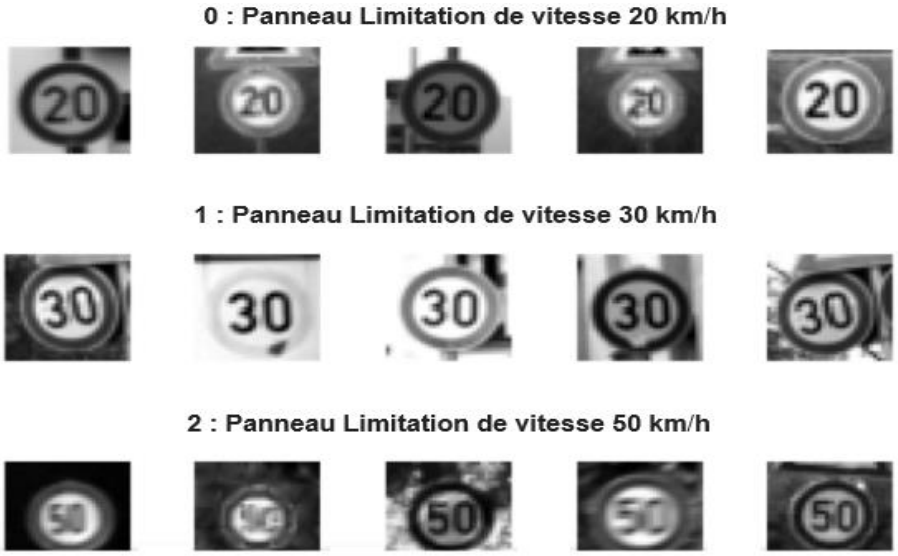


Figure 3.4 : Transformer des images RGB en niveau de gris

- **Normalisation des images :**

Nous centrons la distribution de l'ensemble des données de l'image en soustrayant chaque image par la moyenne de l'ensemble des données et en divisant par son écart type. Cela aide notre modèle à traiter les images de manière uniforme. Les images résultantes se présentent comme suit :

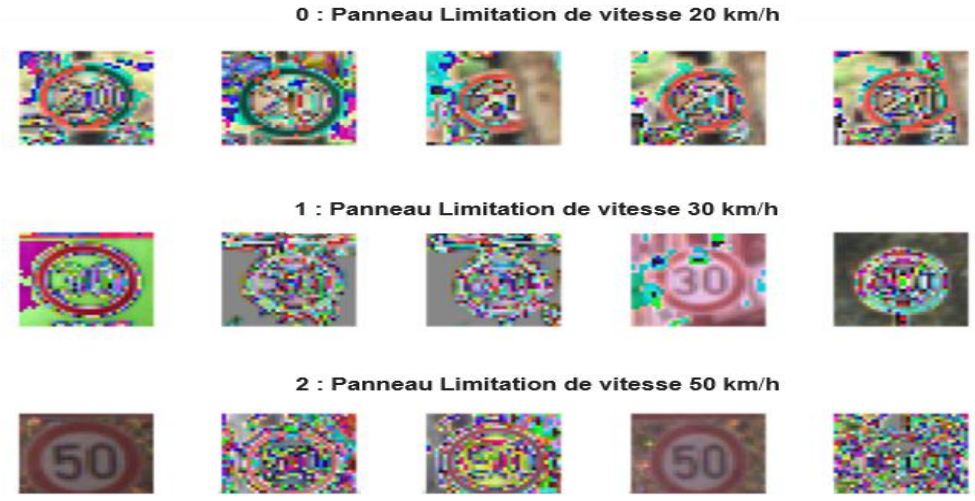


Figure 3.5 : Normalisation des images

3.1.2 L’augmentation des données :

La bibliothèque matplotlib nous permet de voir la distribution de nos images de panneaux de signalisation dans l’ensemble des classes, la distribution montre qu'ils sont inégalement répartis (certaines classes ont moins de 200 images, tandis que d’autres ont plus de 1200). Cela signifie que notre modèle pourrait être biaisé vers des classes surreprésentées, surtout lorsqu’il n'est pas sûr de ses prédictions. Cela peut nous faire un grand problème dans la classification ou bien la reconnaissance. L’histogramme suivant représente le nombre d'échantillons pour chaque classe.

Figure 3.6 : Distribution d'échantillons pour chaque classe.

La quantité de données dont nous disposons n'est pas suffisante pour qu'un modèle se généralise correctement. Il est également assez déséquilibré, et certaines classes sont représentées dans une moindre mesure que les autres. Nous allons résoudre ce problème avec l'augmentation des données.

Nous devons donc le faire pour rendre le modèle plus générique comme la rotation des images, zoomer des images, déplacer des images de gauche vers la droite et de droite vers la gauche. On peut faire l’augmentation par une fonction ImageDataGenerator.

4 Architecture de notre réseau

Au cours de nos expérimentations, nous avons créé un modèle proche de leNet5 (Lenet5 modifié), où on a appliqué le modèle sur la base d'images GTSR.

Le modèle implémenté que nous présentons dans la figure suivante est composé de quatre couches de convolution, deux couches de maxpooling et de deux couches entièrement connectées (fully connected).

L’image en entrée est de taille 32*32*3, après la transformation en niveaux de gris la taille sera 32*32*1, l’image passe d’abord à la première couche de convolution. Cette couche est composée de 60 filtres de taille 5*5, la fonction d’activation ReLU est utilisée, cette fonction d’activation force les neurones à retourner des valeurs positives, après cette convolution 60 features maps seront créés de taille 28*28 (la taille spatiale du volume de sortie).

$$28 = \frac{32-5}{1} + 1$$

Ensuite, les 60 feature maps obtenus, sont donnés en entrée à la deuxième couche de convolution qui est composée aussi de 60 filtres. La fonction d’activation ReLU est appliquée sur cette couche. Le Maxpooling est appliqué après pour réduire la taille de l’image. À la sortie de cette couche, nous aurons 60 feature maps de taille 12*12.

On répète la même chose avec les couches de convolutions trois et quatre qui sont composées de 30 filtres, la fonction d’activation ReLU est appliquée toujours sur chaque convolution. Une couche de Maxpooling est appliquée après la couche de convolution quatre. À la sortie de cette couche, nous aurons 30 feature maps de taille 4*4. Le vecteur de caractéristiques issu des convolutions a une dimension de 480.

Après ces quatre couches de convolution, nous utilisons un réseau de neurones composé de deux couches fully connected. La première couche est composée de 500 neurones où la fonction d’activation utilisée est la fonction ReLU, la deuxième couche utilise la fonction softmax qui permet de calculer la distribution de probabilité des 43 classes (nombre de classes dans la base d’image GTSR).

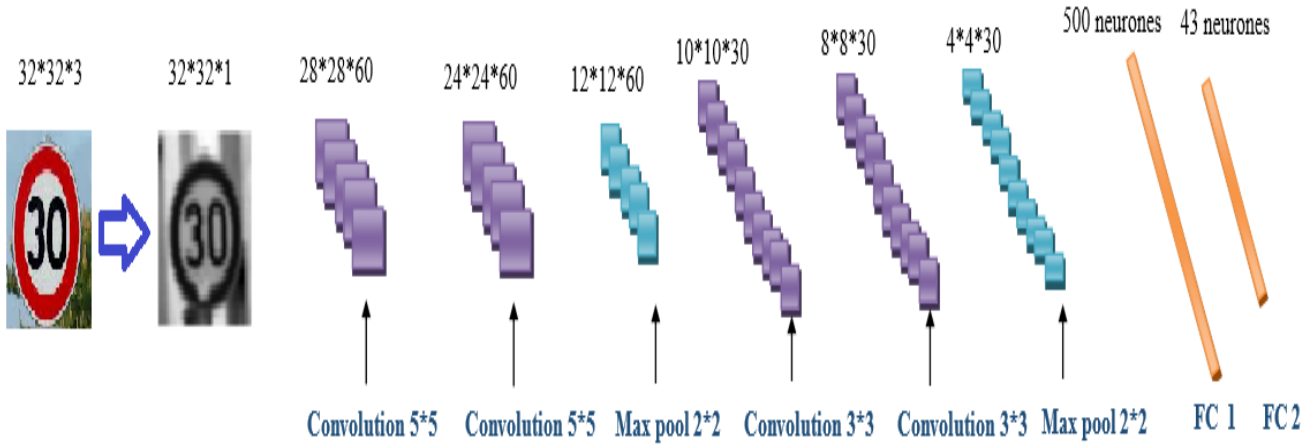
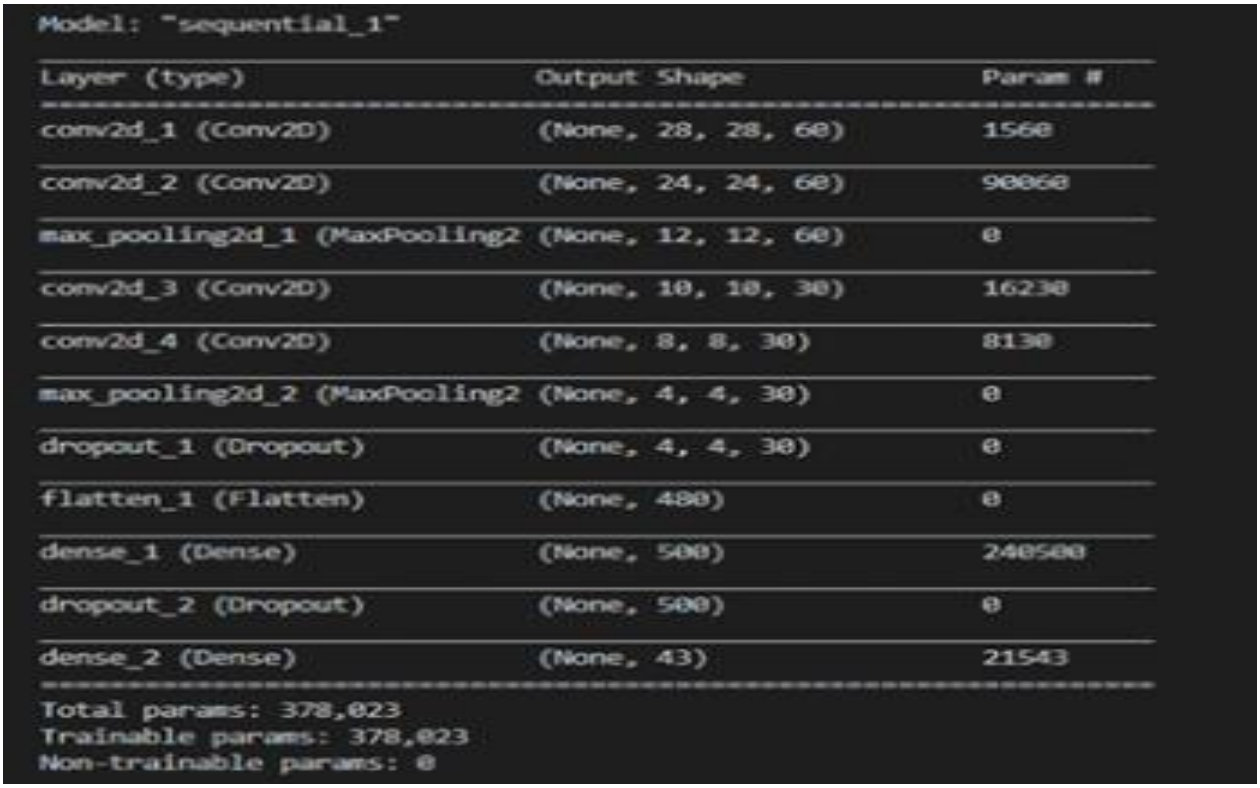


Figure 3.7 : Architecture de notre réseau



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 60)        1560
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 60)        90060
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 60)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 30)        16230
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 30)          8130
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 30)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 30)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 480)               0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               240500
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 43)                21543
=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
```

Figure 3.8 : Configuration de notre modèle

5 Calcul des paramètres de notre modèle :

- **Couche d'entrée :** Aucun paramètre ne peut être appris dans la couche d'entrée.
- **Couche de convolution 1 :**

Pour calculer le nombre de paramètres dans la couche de convolution, on applique la loi suivante : (n*m*l+1) *k**.***

N*m : la taille de filtre, dans notre exemple pour la couche de conv2D_1 est choisi à 5*5.

L : Les cartes de caractéristiques en entrée, on a une image donc l = 1.

K : Carte des caractéristiques en sortie (nombre de filtres) K = 60

Donc le nombre de paramètres est équivalent à : (5*5*1 + 1) *60 = **1560 paramètres.**

- **Couche de convolution 2 :**

Nombre de paramètres : (5*5*60+1) *60 = **90 060 paramètres.**

- **Couche de max pool 1 :**

Aucun paramètre à apprendre dans une couche de max pool.

- **Couche de convolution 3 :**

La taille du filtre est 3*3 et K = 30.

Nombre de paramètres : (3*3*60+1) *30 = **16230 paramètres.**

- **Couche de convolution 4 :**

Nombre de paramètres : (3*3*30+1) *30 = **8130 paramètres.**

- **Couche de max pool 2 :**

Aucun paramètre à apprendre dans une couche de max pool.

- **Couche entièrement connectée :**

Pour calculer le nombre de paramètres dans la couche entièrement connectée, on applique la loi suivante : (**n+1) *m.** Pour **n** entrées et **m** sorties :

Nombre de paramètres : (480+1) *500 = **240 500 Paramètres.**

- **Couche de sortie** : la couche de sortie est une couche entièrement connectée, le nombre de paramètres est alors (n+1) *m :

Le nombre de paramètres de la couche de sortie s'élève à : (500+1) *43 = **21 543 paramètres.**

Le Nombre total de paramètres entrainés est : 1560 + 90 060 + 16230 + 8130 + 240 500 + 21 543 = **378023 paramètres.**

6 Résultats obtenus et discussion

6.1 Discutions :

Les résultats obtenus en terme de précision et d'erreur de notre modèle, sont illustrés sur la Figure 3.9.

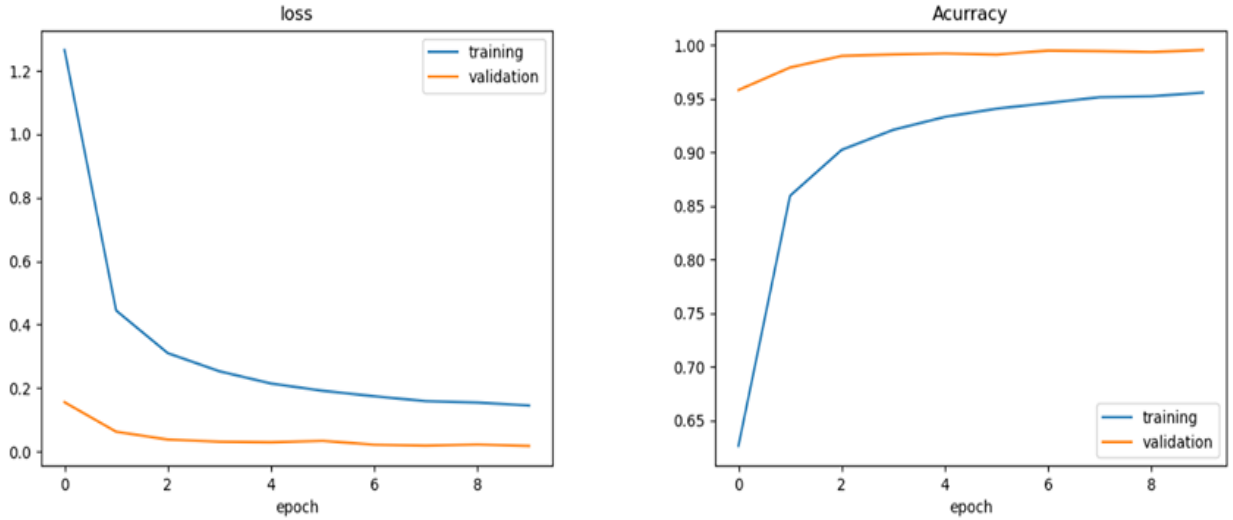


Figure 3.9 : Précision et Erreur pour le Modèle implémenté

Après l'analyse des résultats obtenus, On constate les remarques suivantes :

D'après la Figure 3.9 la précision de l'apprentissage et de la validation augmente dans le temps en fonction du nombre d'époques (une époque est un ensemble d'itérations en avant et en arrière à travers le réseau neuronal seulement une fois), ceci reflète que le modèle apprend au fur et mesure plus d'informations. Si la précision est moindre alors on aura besoin de plus d'informations pour faire apprendre notre modèle et par conséquent on doit augmenter le nombre d'époques et vice versa. De même, l'erreur d'apprentissage et de validation diminue en fonction du temps. Les résultats nous mènent à une précision d'entrainement de 0,9558, une perte d'entraînement de 0,1450, une perte de validation de 0,0176 et une précision de validation de 0,9955.

La durée totale d'apprentissage pour 10 époques est de 10 198 secondes (CPU) presque 3 heures, chaque époque prend environ 16 min.

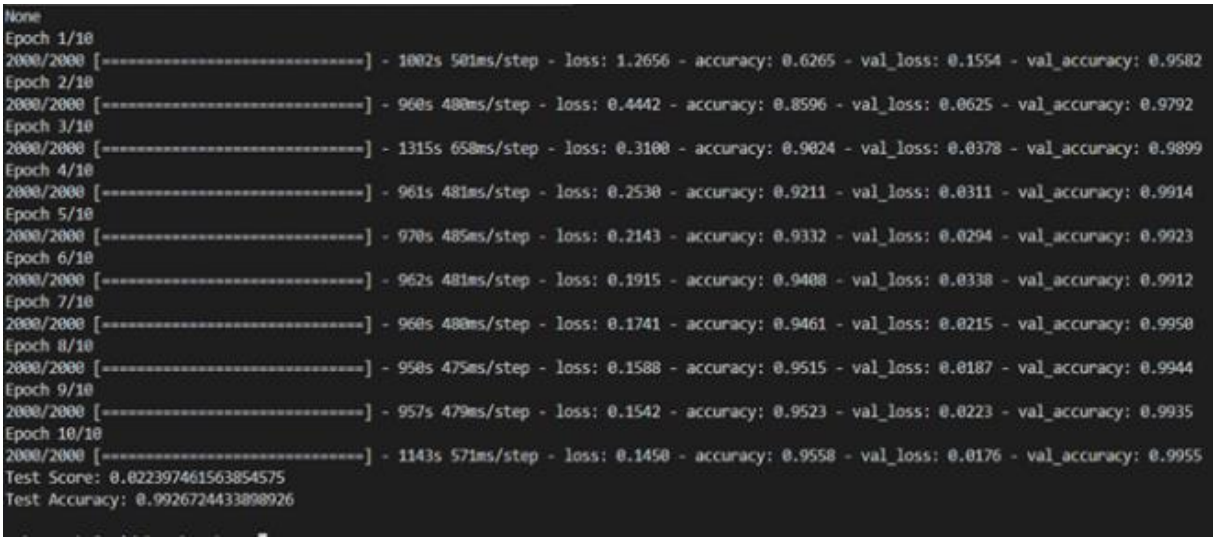


Figure 3.10 : Détails sur l'apprentissage de notre modèle sur CPU

Remarque : on a débuté l'implémentation avec 30 époques comme montré sur la Figure 3.11, mais d'après la Figure 3.11 (accuracy en fonction d'époques) le nombre 10 époques était suffisant pour obtenir un bon résultat, on l'a réduit pour gagner en temps (À 10 époques accuracy est plus que 0 ,95)

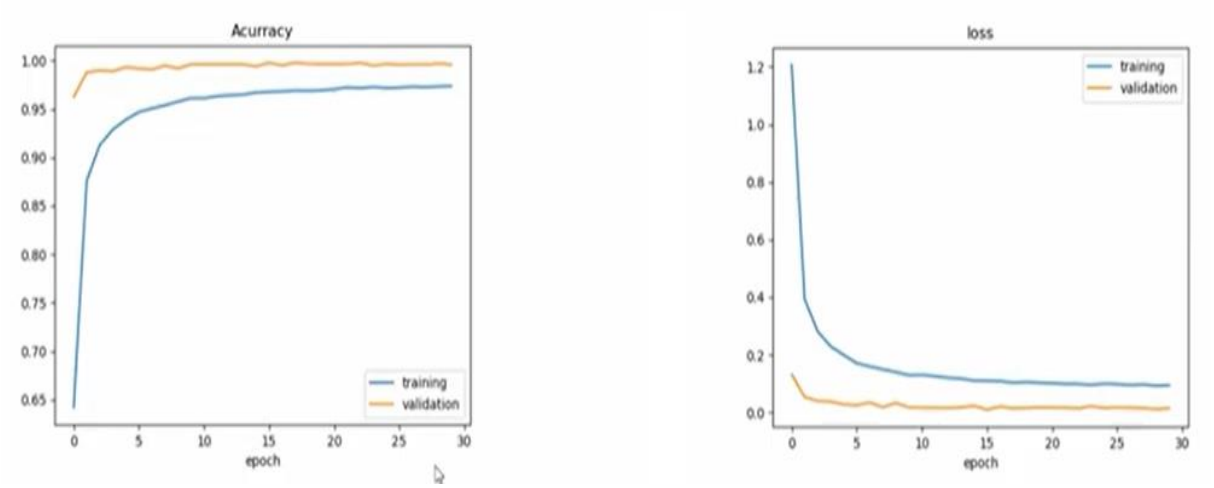


Figure 3.11 : Précision et Erreur pour le Modèle implémenté 30 époques

6.2 Implémentation sur un GPU :

La compatibilité GPU de TensorFlow nécessite un ensemble de pilotes et de bibliothèques surtout qu'on a travaillé sur une GPU de la famille nvidia**. [20]**

- **Etape 1 :** vérifier si le pilote de carte graphique est installé sur le pc sinon il faut le télécharger du site officiel nvidia et installé.
- **Etape 2 :** Télécharger une version CUDA Toolkit **compatible** avec Tensorflow qu'on doit utiliser. CUDA est une technologie de programmation parallèle et un langage de programmation NVIDIA propriétaires pour leurs GPU.
- **Etape 3** : Télécharger la bibliothèque CUDNN **compatible** avec la version de CUDA. Pour cela, il est nécessaire de créer un compte nvidia.

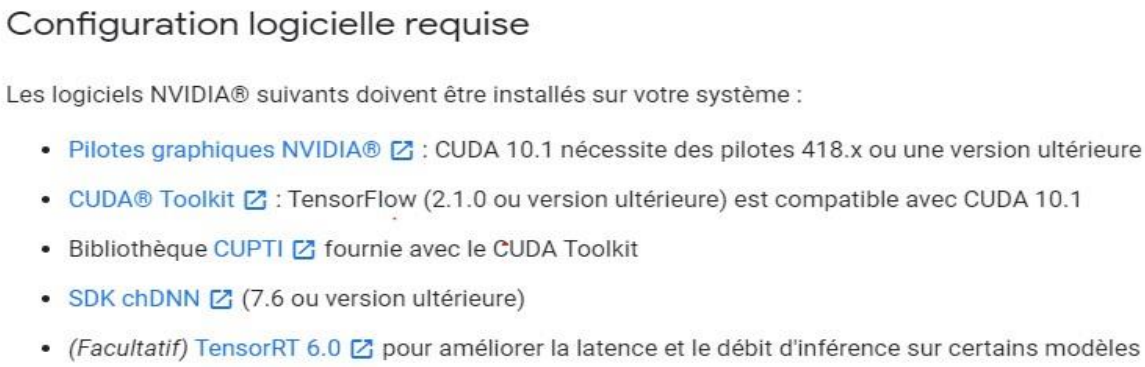


Figure 3.12 : Les pilotes et les bibliothèques utilisés avec ses versions

Une fois le téléchargement de CUDA et CUDNN terminé, on procède à l'installation automatique de CUDA en suivant le guide d'installation et en extrayant le fichier de CUDNN

du fichier de type. rar, par la suite tous les fichiers seront copiés dans l'emplacement approprié de CUDA

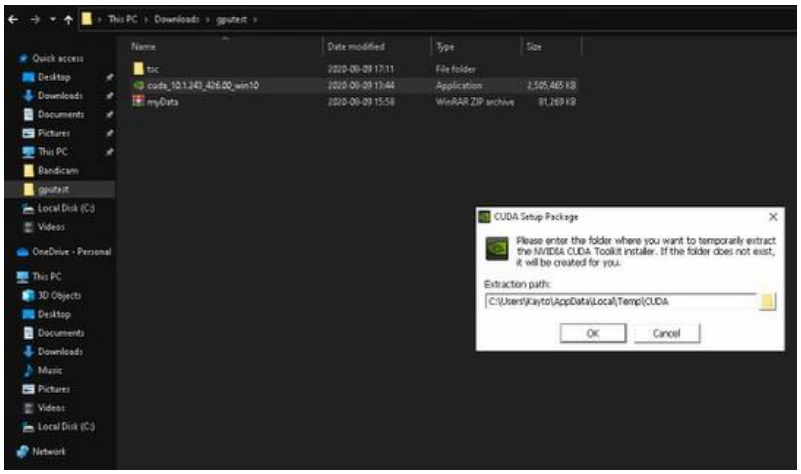


Figure 3.13 : Installation de CUDA

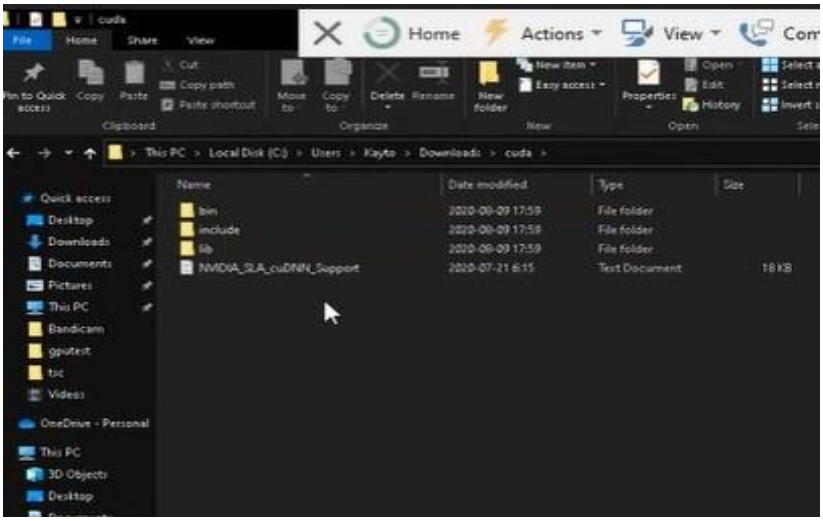


Figure 3.14 : Copie des bibliothèques de cuDNN

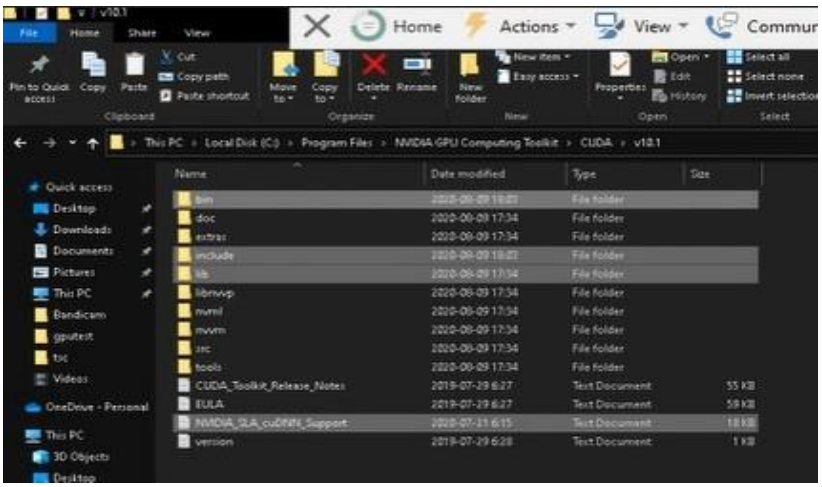


Figure 3.15 : Remplacement des bibliothèques par défaut de cuda avec les bibliothèques de cuDNN

Par la suite les répertoires d'installation de CUDA, et de cuDNN ont été ajoutés à la variable d'environnement

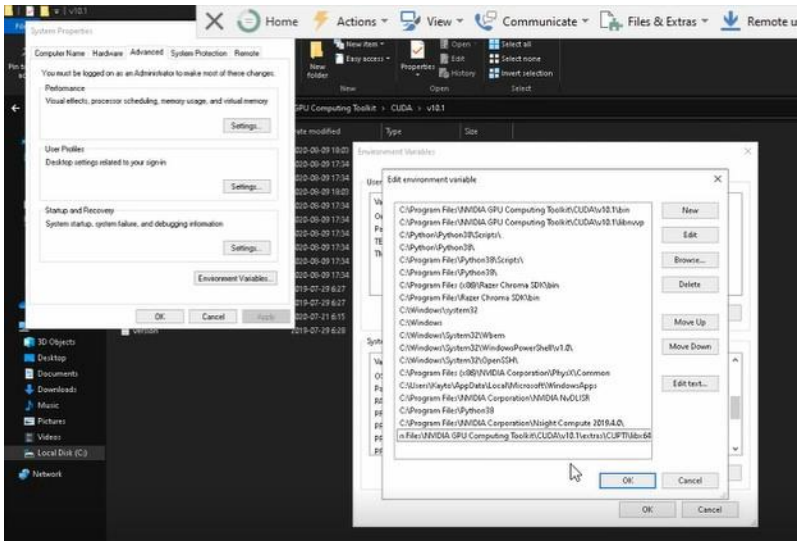


Figure 3.16 : Ajout des répertoires d'installation de CUDA, cuDNN à la variable d'environnement.

Il est également nécessaire d'installer le compilateur Microsoft visual c++ 2015-2019

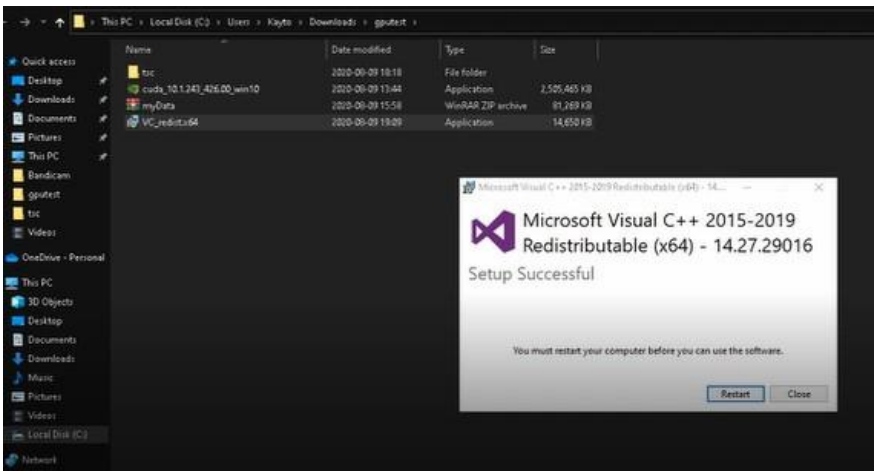


Figure 3.17 : Installation de Microsoft Visual c++

On termine ici la préparation de l'environnement pour l'implémentation sur le GPU. L'implémentation sur un GPU consomme une forte mémoire. Pour cela on a ajouté des lignes de code pour limiter un peu l'utilisation de mémoire par le Tensorflow

```
16  ############## MEMORY LIMIT
17  import tensorflow as tf
18
19  config = tf.compat.v1.ConfigProto()
20  config.gpu_options.per_process_gpu_memory_fraction = 0.7
21  tf.compat.v1.keras.backend.set_session(tf.compat.v1.Session(config=config));
```

Figure 3.18 : Limitation de mémoire

6.2.1 Configuration utilisée dans l'implémentation GPU :

- Un pc de bureau i5 4460 3.2 GHZ.
- Carte Graphique Nvidia GTX 960.
- RAM de taille 8GO.
- Taille disque dur SSD 128 GO.
- Système d'exploitation Windows 10 64 bits.

6.2.2 Résultats obtenus sur GPU :

```
2000/2000 [==============================] - 35s 18ms/step - loss: 1.3029 - accuracy: 0.6154 - val_loss: 0.1228 - val_accuracy: 0.9695
Epoch 2/10
2000/2000 [==============================] - 33s 17ms/step - loss: 0.4299 - accuracy: 0.8643 - val_loss: 0.0545 - val_accuracy: 0.9820
Epoch 3/10
2000/2000 [==============================] - 33s 16ms/step - loss: 0.2905 - accuracy: 0.9091 - val_loss: 0.0355 - val_accuracy: 0.9878
Epoch 4/10
2000/2000 [==============================] - 31s 15ms/step - loss: 0.2345 - accuracy: 0.9268 - val_loss: 0.0248 - val_accuracy: 0.9916
Epoch 5/10
2000/2000 [==============================] - 29s 15ms/step - loss: 0.1983 - accuracy: 0.9390 - val_loss: 0.0283 - val_accuracy: 0.9926
Epoch 6/10
2000/2000 [==============================] - 30s 15ms/step - loss: 0.1732 - accuracy: 0.9462 - val_loss: 0.0205 - val_accuracy: 0.9937
Epoch 7/10
2000/2000 [==============================] - 33s 16ms/step - loss: 0.1564 - accuracy: 0.9519 - val_loss: 0.0218 - val_accuracy: 0.9941
Epoch 8/10
2000/2000 [==============================] - 32s 16ms/step - loss: 0.1456 - accuracy: 0.9554 - val_loss: 0.0128 - val_accuracy: 0.9969
Epoch 9/10
2000/2000 [==============================] - 33s 16ms/step - loss: 0.1388 - accuracy: 0.9579 - val_loss: 0.0154 - val_accuracy: 0.9952
Epoch 10/10
2000/2000 [==============================] - 30s 15ms/step - loss: 0.1275 - accuracy: 0.9615 - val_loss: 0.0147 - val_accuracy: 0.9959
Test Score: 0.014812318817269118
Test Accuracy: 0.9958333373069763
```

Figure 3.19: Détails sur l'apprentissage de notre modèle sur GPU

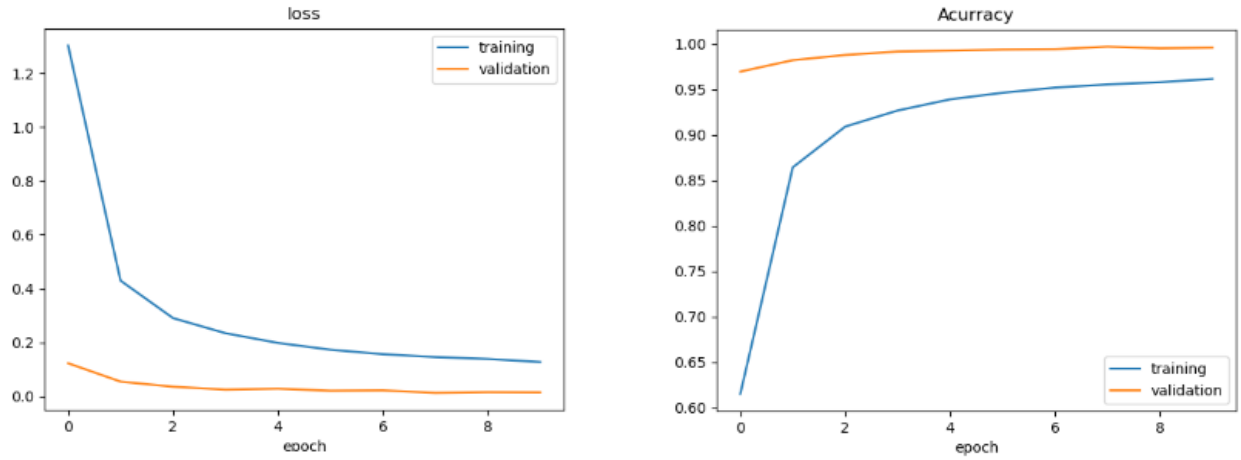


Figure 3.20 : Précision et Erreur pour le Modèle implémenté (GPU)

Nous remarquons une bonne précision d'entrainement de 0,9615, une perte d'entraînement de 0,1275, une perte de validation de 0,0147 et une précision de validation de 0,9959.
La durée totale d'apprentissage pour 10 époques est de 319 secondes (GPU) soit 5,31 min, chaque époque prend environ 32 secondes.

6.3 Tableau de comparaison des résultats :

	Architecture Utilisé			Nombre Epoque	Précision obtenu sur la base d'apprentissage	Précision obtenu sur la base de validation	erreur	Temps d'exécution
	Couche de Convolution	Couche de Pooling	Fully connected					
CPU	04	02	02	10	95%	99%	14%	10 198 secondes
GPU	04	02	02	10	96%	99%	12%	319 Secondes

Tableau 3.1 : Résultats sur CPU & GPU

Le tableau montre l'architecture utilisée dans notre modèle le nombre d'époques et les résultats obtenus exprimés en termes de précision d'apprentissage, de précision de validation,

de test d'erreur et enfin de temps d'exécution. Le temps d'exécution étant trop couteux, Ceci nécessite l'utilisation d'un GPU au lieu d'un CPU.

6.4 Exécution en temps réel par une webcam :

La classification d'une plaque de signalisation en temps réel est la dernière station dans un système de vison par ordinateur, pour cela faut-il un bon appareil d'acquisition d'image pour que l'image à classifier soit nette et avec un minimum de bruit.

Un appareil par défaut d'un ordinateur portable (0.3MegaPixel) n'est pas assez bon pour l'application de classification d'images (surtout celle qui est basée sur les petits détails de l'image). En revanche un appareil d'un smartphone de 13 Mégapixels assure une meilleure acquisition.

Pour utiliser un smartphone en tant que webcam, il y a de nombreuses applications multimédia qui transforment le téléphone mobile en webcam haute définition pour l'ordinateur. Pour cela on a utilisé l'application iVcam. Cette dernière a été installée sur le smartphone(Android) et l'ordinateur (Windows), la connexion entre les deux dispositifs se fait via un réseau wifi. Les deux dispositifs sont connectés entre eux à l'aide d'une connexion radio.

6.5 Résultats :

Les figures 3.21, 3.22, 3.23, 3.24, et 3.25 montrent les résultats de la classification de notre système.

Figure 3.21: Reconnaissance d'une image "Entrée interdite"

Figure 3.22: Reconnaissance d'une image "limitation de vitesse 70km/h"

Figure 3.23: Reconnaissance d'une image "limitation de vitesse 30 Km/h"

Figure 3.24: Reconnaissance d'une image "Interdiction de depasser"

Figure 3.25: Reconnaissance d'une image "Interdiction aux poids lourds"

6.6 Observation :

Lors de l'évaluation de notre modèle, on a constaté quelques observations concernant la précision de la reconnaissance comme le montre la Figure 3.27 et la Figure 3.28.

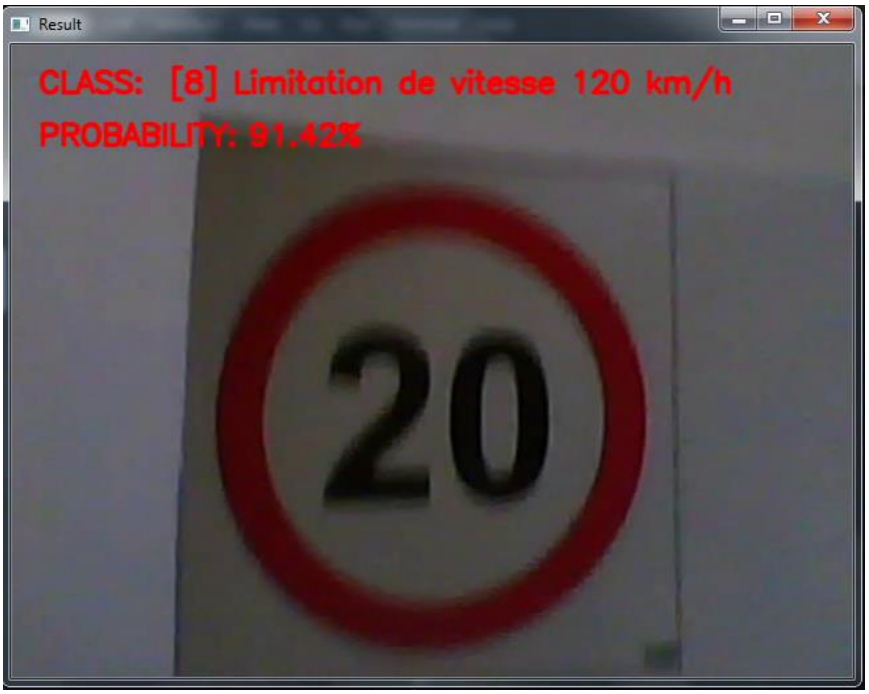


Figure 3.26 : Reconnaissance d'une image "Limitation de vitesse 120 km/h"

Figure 3.27 : Reconnaissance d'une image "STOP"

La classification d'un panneau de signalisation comme « limitation de vitesse 20 km/h » a connu une mauvaise reconnaissance, cela est dû au nombre d'images qui existe dans la classe [0] (représentant la limitation de vitesse 20 km/h), comme le montre la figure 3.6 le nombre d'images dans la classe [0] est inférieure à 200.

En revanche la classification d'un panneau de signalisation « STOP » a connu une bonne précision à cause du nombre d'images qui existe dans la classe [14] soit 500 images, cela signifie que le choix du nombre d'images joue un rôle important dans la précision ainsi que la profondeur et la complexité de l'architecteure du réseau de neurone.

7 Code Source du modèle

Après l'nstallation des bibliothéques necessaires TensorFlow et d'autres bibliothèques comme keras, matplotlib ect...., on a implementé notre système de la classification des panneaux routiers à l'aide d'une architecture massivement parallèle pour la programmation python (on trouve plus informations dans l'annexe) le code source est divisé comme suit:

A) Paramètres
B) Importation data
C) Diviser data
D) Lire dossier csv
E) Prétraitement d'image

F) L’augmentation des données

G) Création de notre modèle de neurones

8 Conclusion

Nous avons présenté dans ce chapitre l’architecture utilisée pour la classification basée sur les réseaux de neurones convolutionnels, proche de l’architecture LeNet-5. Pour cela on a utilisé le modèle avec différentes couches de convolution, de max pool et de couches entièrement connectées et on a montré les résultats obtenus en termes de précision et d’erreur. Le temps d’exécution étant trop couteux, ceci nécessite au mieux l’utilisation d’un GPU au lieu d’un CPU.

On a constaté aussi que le nombre de données de chaque classe nécessaire pour avoir un bon modèle de classification est de 500 images. Les résultats obtenus montrent également que le nombre d’époques, la taille de la base et la profondeur du réseau, sont des facteurs importants pour l’obtention de meilleurs résultats.

Conclusion Générale

Dans les systèmes modernes de sécurité des véhicules, de nombreuses technologies pourraient nous aider à conduire et informer le conducteur en cas de danger.

La capacité à surveiller en permanence les panneaux de signalisation de restrictions et les avertissements sur la route, conduit à ce que le conducteur est souvent distrait du contrôle du véhicule. Ainsi, cela augmente les risques d'accidents.

La solution consiste à développer activement des systèmes de classification des panneaux de signalisation pour informer le conducteur.

À la suite de ce travail, on a développé une application de classification des panneaux de signalisation en introduisant les notions fondamentales des réseaux de neurones en général et des réseaux de neurones convolutionnels en particulier. Nous avons introduit ces réseaux de neurones convolutifs en présentant les différents types de couches utilisées dans la classification, la couche de convolution, la couche de rectification, la couche de pooling et la couche entièrement connectée (fully connected). On a parlé aussi des méthodes de régularisation (dropout et data augmentation) utilisées pour éviter le problème de sur apprentissage.

Le nombre d'images réparties sur les classes sont importantes pour obtenir un bon résultat, dans notre cas et en fonction de notre réseau la moyenne d'image nécessaire est de 500 images.

Nous avons rencontré quelques problèmes dans la phase d'implémentation, l'utilisation d'un CPU a fait que le temps d'exécution était trop couteux. Cependant pour un gain en temps, le déploiement des réseaux de neurones convolutionnels plus profonds sur des bases plus importantes sur un GPU est souhaitable.

La mise en œuvre logicielle de l'application a été réalisée à l'aide de l'environnement de développement PYTHON car cet environnement est multiplateforme avec l'utilisation de la librairie OpenCV.

Les résultats de notre projet montrent que l'application de classification des panneaux de signalisation est efficace et peut s'appliquer plus tard à des systèmes de reconnaissance des panneaux de signalisation visant le code de la route d'Algérie, afin d'aider les routiers et les transporteurs

Durant ce projet, nous avons eu l'occasion de se rapprocher du monde de l'intelligence artificielle, et découvrir l'importance de ce domaine actuellement.

Références Bibliographiques

[1] Chesner Desir, Classification Automatique d'Images, Application à l'Imagerie du Poumon Profond, Université de Rouen, 2013.

[2] Expert Vision, Different Types of images, Octobre 2014 https://xpertsvision.wordpress.com/2014/10/25/different-types-of-images/.

[3] https://www.canson-infinity.com/fr/faq/qu-est-ce-que-la-resolution-d-une-image#:~:text=La%20r%C3%A9solution%20d'une%20image,'image%20(plus%20pr%C3%A9cise)

[4] Jonathan Weber, Segmentation morphologique interactive pour la fouille de séquences vidéo, Sept2011, https://www.researchgate.net/figure/Le-4-voisinage-et-le-8-voisinage_fig5_233862731

[5] Z YAZID, O YAHI, Contours actifs Paramétriques pour la Segmentation d'images, université de bouira ,2017.

[6] D.Boukhlouf ,Généralités sur le traitement d'images,2015 http://thesis.univ-biskra.dz/2271/6/Chapitre%2003.pdf

[7] https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-Intro-ApprentStat.pdf

[8] Rémi Eyraud , Classification-Apprentissage-Décision.octobr2017. http://pageperso.lif.univ-mrs.fr/~remi.eyraud/CAD/theorie.pdf

[9] Yann LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/.

[11] D. H. Hubel and T. Wiesel. Receptive fields, binocular interaction and functional architecturein the cat's visual cortex. Physiol, 160 :106–154, 1962.

[12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, Neural Information Processing Systems, volume 2. Morgan Kaufman, 1990.

[13] Yann LeCun,THE MNIST DATABASE of handwritten digits,1999

[14] D.E. Rumelhart, et J.L.Mc Clelland, Explorations in the Microstructure of Cognition,1986, https://www.becoz.org/these/memoirehtml/ch06s04.html.

[15] Florent SIMON,Deep Learning, les fonctions d'activation, Avril 2018 https://www.supinfo.com/articles/single/7923-deep-learning-fonctions

[16] Antoine Cornuéjols, Les réseaux de neurones, Université d'Orsay https://www.lri.fr/~antoine/Courses/ENSTA/Tr-ensta-nnx9.pdf

[17] Mr. MIMOUNE Zakarya, Développement d'une Architecture Basée sur l'Apprentissage Profond (Deep Learning) pour la Détection d'Intrusion dans les Réseaux, Université Ahmed Draia – Adrar ,2018.

[18] https://fr.wikipedia.org/wiki/R%C3%A9seau_neuronal_convolutif.

[19] baderts , onv-neural-network , mars 2017 https://www.it-swarm.dev/fr/conv-neural-network/.

[20] https://www.tensorflow.org/install/gpu

Annexe

A) Paramètres :

- **Path = "myData" :** #après l'installation des packages nécessaires, nous avons nos paramètres dans lequel on va mentionner l'endroit des données, le dossier dans lequel elles sont stockées.
- **LabelFile = 'étiquettes .csv' :** #fichier csv avec tous les noms de classes.
- **Epochs_val=10 :** #nombre d'époques ou bien nombre d'itérations, 10 est suffisant pour obtenir un bon résultat mais allez jusqu'à 20 à 30, cela prendra plus de temps
- **ImageDimensions = (32,32,3) :** #image d'entrée avec dimension 32*32*3 (Taille 32*32 avec 3 channel de RGB)
- **Testratio = 0.2 .validationRatio = 0.2 :**#nombre d'images que nous prenons pour l'entrainement et nombre d'images pour le test et de validation , on a pris 20% pour le test et 20% du reste pour la validation et le restant final destiné à l'entrainement.

B) Importation data : une fois la lecture de notre data, le code détectera automatiquement le nombre de classes**.**

```
Total Classes Detected: 43
Importing Classes.....
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
Data Shapes
```

Donc le nombre total de classes détecté est 43, après il importe toutes ces images de classes dans une matrice images, et le numéro d'identification de classes classé dans une matrice classNo.

- **Images = np.array(images)** : #matrice comporte des images de classes.
- **classNo = np.array(classNo)** : #matrice comporte nombre ID de classes.

C) Diviser data :

- **X_train, X_test, y_train, y_test = train_test_split(images, classNo,test_size=testRatio) :** #divisé data en notre test.
- **X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train, test_size=validationRatio) : #divisé data en validation.**

```
Total Classes Detected: 43
Importing Classes.....
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
Data Shapes
Train(22271, 32, 32, 3) (22271,)
Validation(5568, 32, 32, 3) (5568,)
Test(6960, 32, 32, 3) (6960,)
```

D) Lire un dossier csv :

- **data=pd.read_csv(étiquettes) : #lire des étiquettes 0 jusqu'à 42.**

E) Prétraitement d'image :

- **img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)** : **#convertir en Grayscale.**
- **img =cv2.equalizeHist(img) : #standardiser l'éclairage dans une image**
- **img = img/255** : **#normaliser les valeurs entre 0 et 1 au lieu de 0 à 255.**

F) L'augmentation des données :

- **dataGen= ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, zoom_range=0.2, shear_range=0.1, rotation_range=10) :** #rotation des images , déplacement de gauche à droite , agrandissement des images , de sorte qu'il crée un ensemble de données différentes.

G) Création de notre modèle de neurone :

- **Nombre _de_ filtre1 = 60 :** #nombre de filtre utilisé dans la couche de convolution 1 et 2.
- **Nombre_ de_ filtre2 = 30 :** #nombre de filtre utilisé dans la couche de convolution 3 et 4
- **Size_Filtre1 = 5 :** #Taille de filtre 1.
- **Size_Filtre2 = 3 :** #Taille de filtre 2.

- **Pool _size = 2 :** # Taille de pool.
- **Nombre _de_ nœuds = 500 :** #Couche cachée avec une taille de 500 neurones.
- **Couche de convolution 1 :**

  model.add((Conv2D(nombre_ de_ filtre1,size_filtre1, ,input_shape = (imageDimnsion [0] , image Dimension[1],1) , activation='relu'))) : # Cette commande permet de créer 60 cartes de caractéristiques en utilisant un filtre de taille 5 par 5 pixels , appliqué sur une image d'entrée de taille 32*32*3 et une fonction d'activation de type RELU.
- **Couche de convolution 2 :**

  model.add((Conv2D(nombre_de_filtre1, size_Filter1, activation='relu'))) :#Cette commande permet de créer 60 cartes de caractéristiques en utilisant un filtre de taille 5 par 5 pixels et une fonction d'activation de type RELU.
- **Couche de max pooling1 :**

  model.add(MaxPooling2D(pool_size=size_of_pool)) : #Cette ligne de commande permet de réduire la taille de l'image, La méthode Max Pooling est utilisée et la taille de l'image sera divisée sur 2.
- **Couche de convolutio 3 :**

  model.add ((Conv2D(Nombre_ de_ filtre2, size_of_Filter2,activation='relu'))) : #Cette commande permet de créer 30 cartes de caractéristiques en utilisant un filtre de taille 3 par 3 pixels et une fonction d'activation de type RELU.
- **Couche de convolution 4 :**

  model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu'))) : # Cette commande permet de créer 30 cartes de caractéristiques en utilisant un filtre de taille 3 par 3 pixels et une fonction d'activation de type RELU.
- **Couche de max pooling 2 :**

  model.add(MaxPooling2D(pool_size=size_of_pool)) : #Cette ligne de commande permet de réduire la taille de l'image, La méthode Max Pooling est utilisée et la taille de l'image sera divisée sur 2.

- **model.add(Dropout(0.5)) :** #Pour ne pas tomber dans le problème de sur apprentissage il faut utiliser dropout , très efficace pour les réseaux de neurones pour régulariser et n'a besoin que de deux paramètres pour être défini, dont :
    - Le paramètre type avec pour valeur 'dropout'
    - Le paramètre rate avec pour valeur 0.5.
- **model.add(Flatten()) :** #Cette commande permet de créer un seul vecteur 1D puis connecter avec la première couche cachée pour commencer la classification.
- **model.add(Dense(Nombre_de_ nœuds,activation='relu')) :** #Cette commande permet de créer une couche cachée avec une taille de 500 neurones, la fonction RELU est utilisée comme fonction d'activation.
- **model.add(Dense(noOfClasses,activation='softmax') :** : #Cette commande permet de créer une couche de sortie composée de 43 neurones (nombre de classes) la fonction softmax est utilisé pour calculer la probabilité de chaque classe.

- **model.compile(Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy']** #Cette commande permet de compiler notre modèle, elle prend deux paramètres la fonction loss et Opimizer. On a choisi la fonction categorical_crossentropy comme fonction loss et Adam comme optimizer avec un taux d'apprentissage de 0.001.