ROYAUME DU MAROC
UNIVERSITÉ HASSAN 1^{er}
FACULTÉ DES SCIENCES ET TECHNIQUE
SETTAT

MÉMOIRE

Pour l'obtention du Diplôme de MASTER

En : Mathématiques et Applications

Projet de fin d'études sous le thème

Classification des Panneaux de Signalisation Routière par les Réseaux de Neurones Convolutionnels

Réalisé par :

ZAGNOUNE Otmane

Soutenu le **29 juin 2022**, devant le jury :

Pr. Rachid ELHARTI	FSTS, Université Hassan Premier	Président
Pr. Abdelghani BEN TAHAR	FSTS, Université Hassan Premier	Examinateur
Pr. Jaouad DABOUNOU	FSTS, Université Hassan Premier	Encadrant

Année universitaire : 2021-2022

Remerciements

Je tiens à remercier **ALLAH** le Tout-Puissant de m'avoir donné la santé, la volonté, le courage et la patience pour mener à terme ma formation et pouvoir réaliser ce travail. Je remercie :

— Monsieur le professeur **DABOUNOU Jaouad** pour son encadrement durant toute la durée de ce travail notamment en me faisait partager son esprit de travail, et aussi pour les énormes efforts lors des séances de préparation de ce mémoire. Je le remercie également pour l'aide et les conseils concernant le sujet qu'il m'a apporté lors des différents suivis. Veuillez trouver ici l'expression de mon grand respect et ma profonde admiration pour toutes vos qualités scientifiques et humaines.

— Monsieur le professeur **ELHARTI Rachid**, votre présence me fait un grand honneur, merci d'avoir accepté de présider ce Jury.

— Monsieur le professeur **BEN TAHAR Abdelghani**, je suis très reconnaissant à vous d'avoir bien voulu porter intérêt à ce travail.

— Ma famille : ma mère, mon père, tous les mots du monde ne sauraient exprimer l'immense amour que je vous porte, ni la profonde gratitude que je vous témoigne. Merci pour votre patience, générosité.

Résumé

De nos jours, les tâches de reconnaissance d'objets sont de plus en plus résolues avec les réseaux de neurones convolutionnels (CNN). En raison de leur taux de reconnaissance élevé et de leur exécution rapide, les réseaux de neurones convolutionnels ont amélioré la plupart des tâches de vision par ordinateur, à la fois existants et nouveaux.
L'objectif principal est de mettre en œuvre un algorithme de classification des panneaux de signalisation à l'aide de réseaux de neurones convolutionnels. La formation des réseaux de neurones est réalisée à l'aide de la bibliothèque TensorFlow et d'autres bibliothèques telles que keras et une architecture massivement parallèle pour la programmation.

Table des matières

Table des figures

Liste des tableaux

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, dépasser la limite de vitesse ou manquer un panneau de dépassement crée une situation dangereuse en plus d'une pénalité pour le conducteur.

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

Le développement des derniers niveaux de technologie de processeur mobile a permis à de nombreux constructeurs automobiles d'installer des systèmes de vision par ordinateur dans les véhicules de leurs clients. Ces systèmes améliorent considérablement la sécurité et mettent en œuvre des étapes clés sur la voie de la conduite autonome.

L'une des tâches les plus connues que la vision par ordinateur résout dans cette situation est le problème de la reconnaissance des panneaux de signalisation.

Cependant, les principaux problèmes de ces systèmes sont une faible précision de reconnaissance, un besoin élevé en matériel informatique puissant et l'incapacité de certains systèmes à classer les panneaux routiers dans différents pays.

Ce système crée une aide à la conduite automobile conçue 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 prévoir et de gérer les risques à l'avance.

Un module de reconnaissance des panneaux de signalisation routière (Traffic Signs Recognition TSR) est un composant qui permet à un véhicule de reconnaître les panneaux de signalisation par un flux vidéo provenant d'une caméra installée dans un véhicule.

À la fin des années 1980, Yann LeCun 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, Yann LeCun 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, il était très difficile d'implémenter ces méthodes sur des ordinateurs à cette époque.

En 2012, un événement a soudainement changé 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 classer les panneaux de signalisation routière, on va créer un modèle de réseau de neurones convolutionnels 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, les indicateurs de performance en classification, ainsi que sur l'utilisation des réseaux de neurones pour classer ces images.
— Le deuxième chapitre, s'intéresse sur les réseaux de neurones, Deep learning et quelques algorithmes.
— Le troisième chapitre est consacré à la description des réseaux de neurones convolutionnels et de leurs intérêts dans le domaine de la classification des images.
— Le quatrième chapitre présente la partie expérimentale de notre travail donnant suite à la discussion des différents résultats obtenus.
— On termine par une conclusion générale.

Chapitre 1

Classification des images

En général, la reconnaissance d'images est une tâche facile pour un être humain au cours de son existence, et il a acquis des connaissances qui lui permettent de s'adapter aux changements provoqués par différentes conditions d'acquisition. Par exemple, il peut identifier relativement facilement des objets qui sont partiellement cachés par un dans plusieurs directions. De près et de loin, selon différents éclairages.
La classification automatique des images consiste à attribuer automatiquement une classe à une image à l'aide d'un système de classification. On retrouve ainsi la classification d'objets, de scènes, de textures, la reconnaissance de visages, d'empreintes digitale et de caractères. Il existe deux principaux types d'apprentissage : l'apprentissage supervisé et l'apprentissage non-supervisé.
Dans l'approche supervisée, chaque image est associée à une étiquette qui décrit sa classe d'appartenance, et pour l'autre approche les données disponibles ne possèdent pas d'étiquettes. Mais dans notre travail on s'intéresse de l'approche supervisée.
La classification des images consiste à répartir systématiquement des images selon des classes bien définies.

1.1 Notions de base

1.1.1 Définition d'une image

Définition

Une image est une représentation visuelle de description d'un objet ou personne par dessin, peinture, photographie, films,...etc. C'est aussi derrière l'affichage sur écran un ensemble organisé d'informations, acquises, crées, traitées ou stockées

sous forme binaire (suite de 0 et de 1). Il ne s'agit en réalité que d'une représentation spatiale de la lumière.
L'image est affichée sous la forme d'un ensemble de points associés à la grandeur physique (luminance, couleur). Ces grandeurs peuvent être continues (image analogique) ou bien discrètes (images numérique).
Mathématiquement, l'image représente une fonction continue :

$$f : \quad \mathbb{R}^2 \ \to \mathbb{R}.$$

$$(x, y) \to f(x, y).$$

appelée fonction image, mesurant la nuance du niveau de gris de l'image aux coordonnées (x, y).

Image numérique

L'image numérique est l'image dont la surface est divisée en éléments de taille fixe appelés pixels ou cellules, ayant chacun comme caractéristique un niveau de gris ou de couleurs. La numérisation d'une image est la conversion de celle-ci de son état analogique en une image numérique représentée par une matrice bidimensionnelle de valeurs numériques $f(x, y)$.

FIGURE 1.1 – Image numérique

1.1.2 Les différents types de format d'image

Image couleur RGB

L'œil humain analyse la couleur à l'aide de trois types de cellules photo 'les cônes'. Ces cellules sont sensibles aux basses, moyennes ou hautes fréquences (rouge, vert, bleu). Pour représenter la couleur d'un pixel, il faut donc donner trois nombres correspondant aux doses des trois couleurs de base : Rouge, Vert, Bleu. Par conséquent, nous pouvons représenter une image couleur avec trois matrices Chacun correspond à une couleur de base.

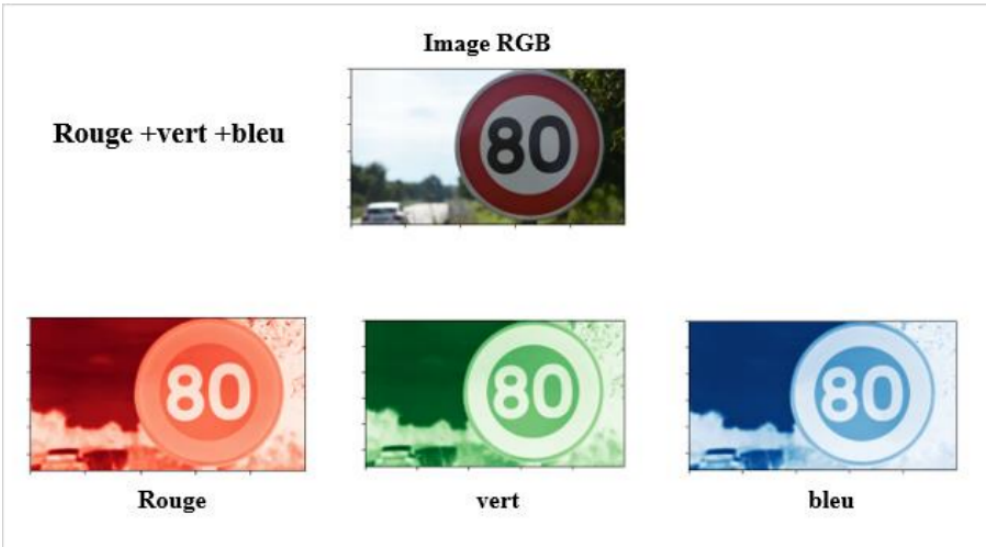


FIGURE 1.2 – Canaux RGB

Image d'intensités

C'est une matrice où chaque élément est un nombre réel compris entre 0 (noir) et 255 (blanc). On parle aussi d'image en niveaux de gris, car les valeurs comprises entre 0 et 255 représentent les différents niveaux de gris.

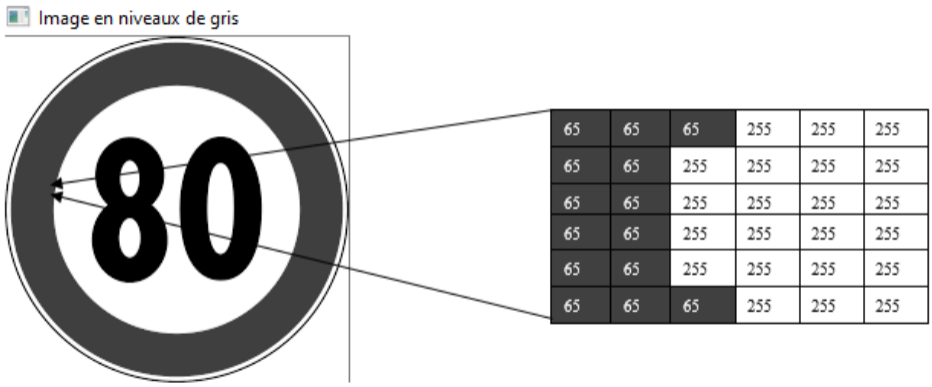


FIGURE 1.3 – Image en niveaux de gris

Image binaire

Une image binaire est une matrice rectangulaire dont les éléments sont 0 ou 1. Lors de la visualisation de telles images, 0 est représenté par le noir et 1 est représenté par le blanc.

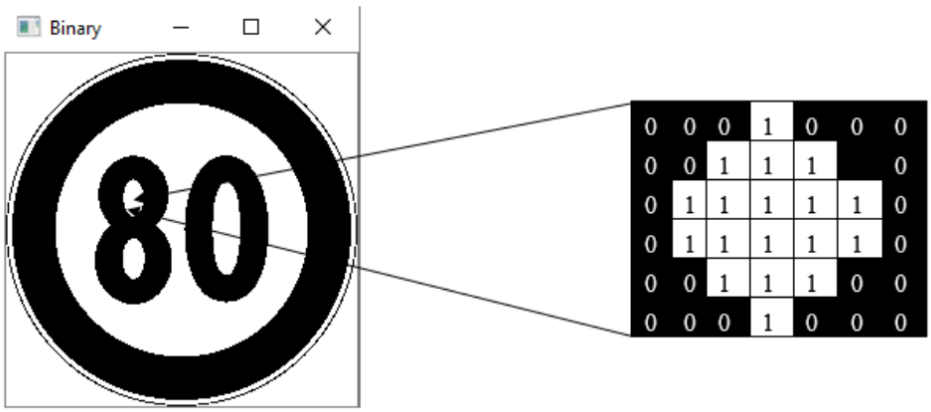


FIGURE 1.4 – Image binaire et la valeur des pixels dans un voisinage 6 * 6

1.1.3 Caractéristiques de l'image

Comme nous l'avons déjà dit, l'image est un ensemble organisé d'informations qui contient les caractéristiques suivantes :

Pixel

Une image numérique est composée d'une série de points appelés pixels (abréviation de Picture Element) qui forment une image. Par conséquent, les pixels représentent la plus petite composante d'une image numérique. L'ensemble de ces pixels est contenu dans un tableau à deux dimensions constituant l'image. Par exemple, peut être affichée comme un groupe de pixels (Figure 1.5)

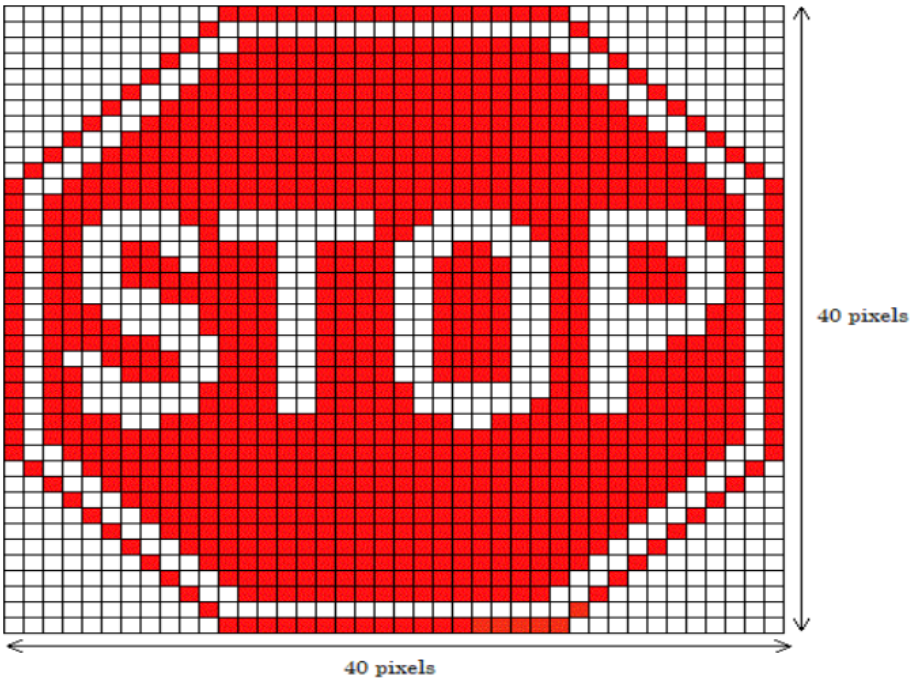


FIGURE 1.5 – Image de 1600 pixels

Résolution

La résolution d'image est le nombre de points ou de pixels par unité de surface, exprimé en points par pouce (PPP, en anglais DPI pour Dots Per Inch). Un pouce correspond à $2,54$ cm.
Nous utilisons également le terme résolution pour indiquer le nombre total de pixels horizontaux et verticaux sur le moniteur. Plus ce nombre est grand, plus la résolution est meilleure.

Dimension

La dimension est la taille de l'image. Elle se présente sous forme d'une matrice dont les éléments sont des valeurs numériques représentatives des intensités lumineuses (pixels). Le nombre de lignes de cette matrice multiplié par le nombre de colonnes nous donne le nombre total de pixels dans une image.

Voisinage

Le plan de l'image est divisé en rectangles ou en hexagones, permettant ainsi l'exploitation de la notion de voisinage (voir Tableaux 1.1 et 1.1). Un voisinage de pixel est formé par tous les pixels entourant le même pixel. On définit aussi l'assiette comme étant l'ensemble de pixels définissant le voisinage pris en compte autour d'un pixel. On distingue deux types de voisinage :

— **Voisinage à 4 :** les pixels qui ont un coté commun avec le pixel considéré.

TABLE 1.1 – Voisinage à 4

	$f(i-1,j)$	
$f(i,j-1)$	$f(i,j)$	$f(i,j+1)$
	$f(i+1,j)$	

— **Voisinage à 8 :** les pixels qui ont au moins un point en liaison avec le pixel considéré.

TABLE 1.2 – Voisinage à 6

$f(i-1,j-1)$	$f(i-1,j)$	$f(i-1,j+1)$
$f(i,j-1)$	$f(i,j)$	$f(i,j+1)$
$f(i+1,j-1)$	$f(i+1,j)$	$f(i+1,j+1)$

Bruit

Le bruit (parasite) dans une image est considéré comme un phénomène dans lequel un pixel change brusquement d'intensité par rapport à ses voisins, il provient de l'optique du capteur et de l'éclairage de l'électronique.

Histogramme

L'histogramme des niveaux de gris ou des couleurs d'une image est une fonction qui donne la fréquence d'apparition de chaque niveau de gris (couleur) dans l'image. Pour réduire les erreurs de quantification, pour comparer deux images obtenues sous des éclairages différents, ou encore pour mesurer certaines propriétés sur une image. Cela fournit de nombreuses informations sur la répartition des niveaux de gris (couleurs) et vous permet de voir entre quelles limites la plupart des niveaux de gris (couleurs) sont répartis dans les images trop claires ou trop sombres.

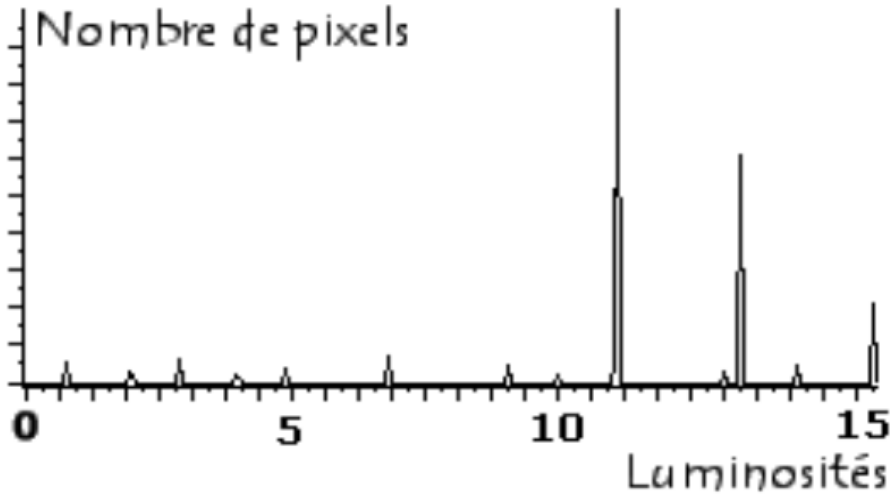


FIGURE 1.6 – Histogramme

Contraste

Il s'agit d'un contraste net entre les deux zones de l'image, plus précisément les zones sombres et claires de l'image. Le contraste est défini en fonction de la luminosité des deux zones de l'image. Si L_1 et L_2 sont les degrés de luminosité respectivement de deux zones voisines A_1 et A_2 d'une image, le contraste C est défini par le rapport :

$$C = \frac{L_1 - L_2}{L_1 + L_2}$$

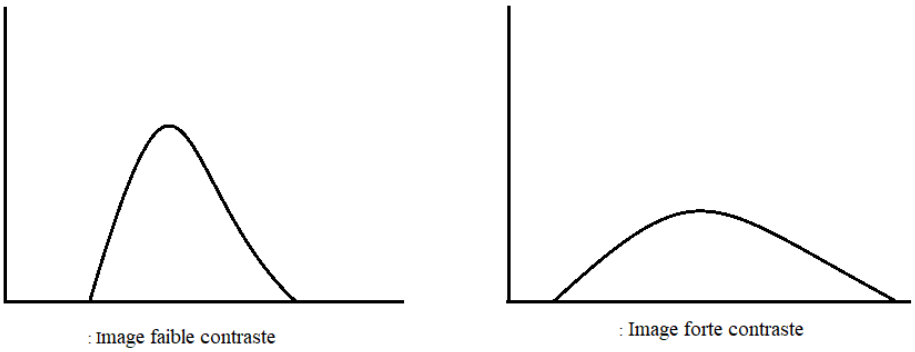


FIGURE 1.7 – image contrastée

Luminance

Il s'agit du niveau de luminosité du point dans l'image. Il est définie aussi comme étant le quotient de l'intensité lumineuse d'une surface par l'aire apparente de cette surface, pour un observateur lointain, le mot luminance est substitué au mot brillance, qui correspond à l'éclat d'un objet.
Une bonne luminance se caractérise par :

1. Des images lumineuses (brillantes).
2. Un bon contraste : il faut éviter les images où la gamme de contraste tend vers le blanc ou le noir ; ces images entraînent des pertes de détails dans les zones sombres ou lumineuses
3. L'absence de parasites.

Contour

Les contours représentent la frontière entre les objets de l'image, ou la limite entre deux pixels dont les niveaux de gris représentant une différence significative. Dans les images numériques, les contours se situent entre des pixels appartenant à des zones d'intensités moyennes différentes. Ce sont des contours de type « saut d'amplitude ». Un contour peut également correspondre à une variation locale d'intensité présentant un maximum ou un minimum ; il s'agit alors de contour « en toit ».

1.2 Méthode de classification dans un contexte supervisée

De nombreuses méthodes classiques ont été dédiées et elles peuvent être divisées en deux grandes catégories. Méthode de classification non supervisée et méthode de classification non supervisée. Cependant, ce paragraphe se concentre sur les cas supervisée.

Le but de la classification supervisée est principalement de définir des règles de classement des objets selon les variables qualitatives ou quantitatives qui les caractérisent. On dispose au départ d'un échantillon dit d'apprentissage dont le classement est connu. Cet échantillon est utilisé pour l'apprentissage des règles de classement.
Il est nécessaire d'étudier la fiabilité de ces règles pour les comparer et les appliquer, évaluer les cas de sous-apprentissage ou de sur-apprentissage (complexité du modèle). On utilise souvent un deuxième échantillon indépendant, dit de validation ou de test.

1.3 Indicateurs de performance en classification

1.3.1 Matrice de confusion

Pour mesurer les performances d'une classification, on prend un exemple d'un classifieur binaire, c'est-à-dire, qui prédit 2 classes notées classe 0 et classe 1.
Pour mesurer les performances de ce classifieur, il est d'usage de distinguer 4 types d'éléments classés pour la classe voulue :

1. Vrai positif VP : Elément de la classe 1 correctement prédit
2. Vrai négatif VN : Elément de la classe 0 correctement prédit
3. Faux positif FP : Elément de la classe 1 mal prédit
4. Faux négatif FN : Elément de la classe 0 mal prédit

Ces informations peuvent être rassemblés et visualisés sous forme de tableau dans une matrice de confusion. Dans le cas d'un classifieur binaire, on obtient (1.3)

TABLE 1.3 – Matrice de confusion

		Classe predite	
		Classe 0	Classe 1
Classe réelle	Classe 0	VN	FN
	Classe 1	FP	VP

En particulier, si la matrice de confusion est diagonale, le classifieur est parfait. Notons que la matrice de confusion est aussi généralisable lorsqu'il y a k classes à prédire ($k > 2$).
Il est possible de calculer quelques indicateurs qui résument la matrice de confusion. Par exemple, pour rendre compte de la qualité d'une prédiction de classe 1, définissez :

▷ **Précision :** Proportion d'éléments bien classés pour une classe donnée :

$$Precision_{\text{de la classe 1}} = \frac{VP}{VP + FP}$$

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

$$Rappel_{\text{de la classe 1}} = \frac{VP}{VP + FN}$$

▷ **F-mesure :** Mesure de compromis entre précision et rappel :

$$F - mesure_{\text{de la classe 1}} = \frac{2 \times Precision \times Rappel}{Precision + Rappel}$$

Il est possible de calculer tous ces indicateurs pour chaque classe. La moyenne pour chaque classe de ces indicateurs fournit un indicateur global de la qualité du classificateur.

$$Precision = \frac{1}{k} \sum_{i=1}^{k} \frac{VP_i}{VP_i + FP_i}$$

$$Rappel = \frac{1}{k} \sum_{i=1}^{k} \frac{VP_i}{VP_i + FN_i}$$

$$F - mesure = \frac{2 \times Precision \times Rappel}{Precision + Rappel}$$

1.3.2 Courbe ROC (Received Operating Characteristic)

Dans le cas d'un classifieur binaire, il est possible de visualiser les performances du classifieur sur ce que l'on appelle une courbe ROC. La courbe ROC est une représentation du taux de vrais positifs en fonction du taux de faux positifs. Son intérêt est de s'affranchir de la taille des données de test lorsque les données sont déséquilibrées.
Cette représentation met en avant un nouvel indicateur qui est l'aire sous la courbe. Plus elle se rapproche de 1, plus le classifieur est performant (voir figure 1.8).

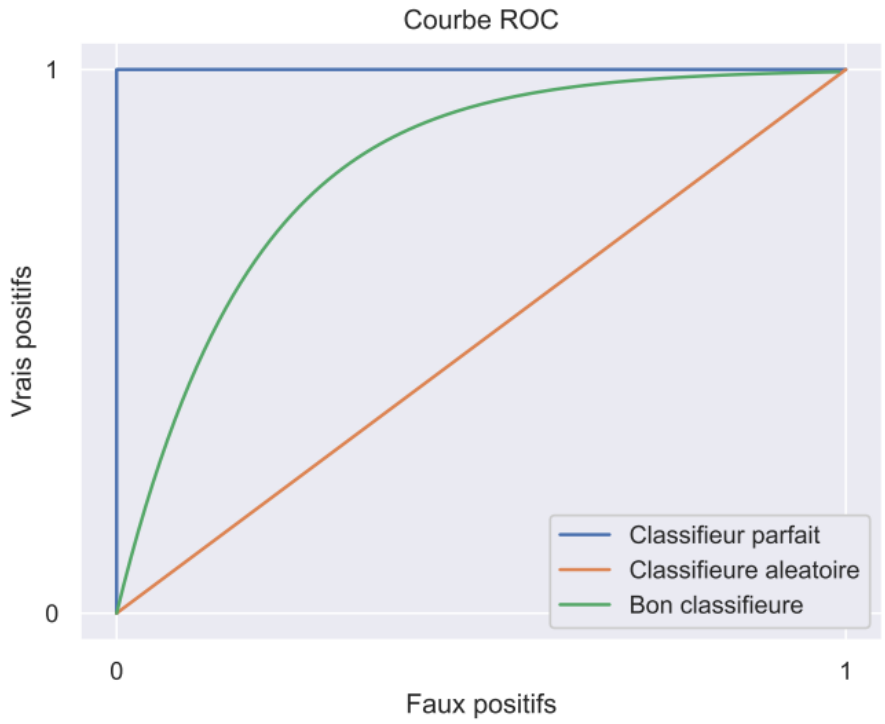


FIGURE 1.8 – Courbe ROC

1.4 Classification des images en machine Learning

Pour des tâches apparemment très simples telles que la classification d'images, la reconnaissance d'objets dans des images et la reconnaissance vocale, les méthodes manuelles se sont avérées très difficiles à appliquer. Les données venant du monde réel les échantillons d'un son ou les pixels d'une image sont complexes, variables et entachées de bruit.
Pour les machines, une image est un tableau de nombres qui indiquent la lumino-

sité (ou la couleur) 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 d'apprentissage automatique peut être considéré comme un système avec des entrées telles que des images et des sorties qui peuvent représenter les catégories d'objets dans l'image. C'est ce qu'on appelle un système de classification ou un 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 souhaitée. La machine peut alors reconnaître les images d'entraînement et les classer 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 dans la reconnaissance d'images sont l'extraction de caractéristiques (feature extraction) et un classificateur entraînable. 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 [6].

1.5 Classification des images par les réseaux de neurones

Le problème de l'approche classique de la reconnaissance d'images, le soi-disant "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 deep Learning DL s'appuie sur des réseaux de neurones artificiels inspirés du cerveau humain, tout comme les avions sont inspirés par les oiseaux.
L'architecture du réseau neuronal peut être considérée comme un réseau multicouche interconnecté par des poids entraînables. C'est ce qu'on appelle un réseau neuronal multicouche. Chacun reçoit et interprète 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 vous devez calculer un gradient qui représente la quantité d'augmentation ou de diminution des erreurs de sortie lorsque les paramètres changent. Ce gradient est calculé en utilisant la méthode de rétropropagation [5].

Conclusion

Dans ce chapitre nous avons présenté les concepts de la classification dans le domaine de l'imagerie, et nous avons également introduit machine learning et l'utilisation des réseaux de neurones dans le domaine de la classification. Dans le deuxième chapitre on va bien détailler les réseaux de neurones.

Chapitre 2

Réseaux de neurones MLP

Puisque ce mémoire porte en grande partie sur les réseaux de neurones convolutionnels, ce chapitre vise à définir les réseaux de neurones MLP (multi-layer-perceptron) ; architecture et quelque algorithmes d'apprentissage automatique. En partant du perceptron simple et comment il apprend, fonctions d'activations, fonction perte et finalement en généralise pour les MLP.

2.1 Le Perceptron Simple

2.1.1 Introduction

Le perceptron a été introduit en 1958 par Franck Rosenblatt. Il s'agit d'un neurone artificiel inspiré par la théorie cognitive de Friedrich Hayek et celle de Donald Hebb. Dans sa version la plus simple, le perceptron n'a qu'une seule sortie y à laquelle toutes les entrées x_i sont connectées (2.1).

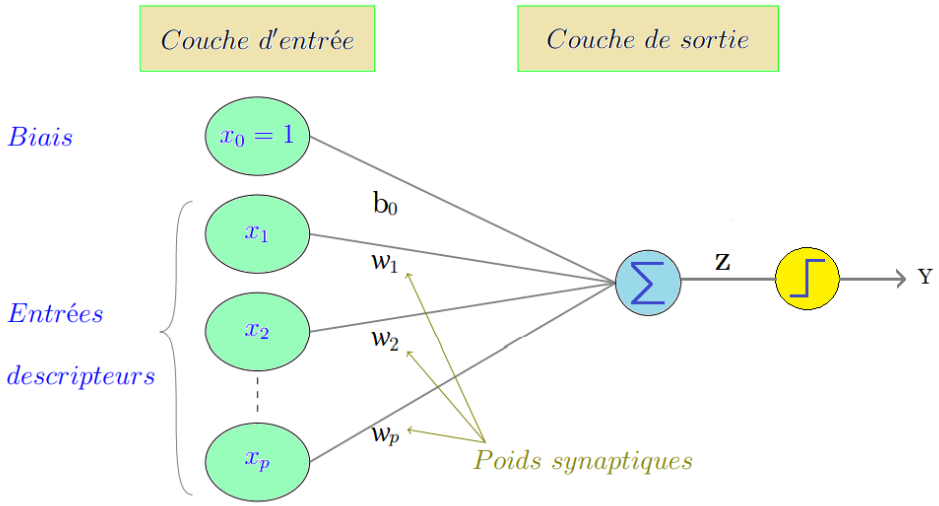


FIGURE 2.1 – Le perceptron

La propagation d'un vecteur d'entrée X à travers un perceptron s'écrit donc :

$$z = \sum_{1}^{p} w_i x_i + b_0$$

$$y = a(z)$$

Avec z s'appelle le potentiel et a désigne la fonction de seuil (ou bien d'activation) de Heavisid.
Maintenant de nombreuses variantes ont été développées pour pouvoir résoudre des problèmes de classification très complexe, et si on met l'activation linéaire on tombe sur la régression linéaire.

2.1.2 les fonctions d'activation

Tout apprentissage d'un modèle de réseau de neurones se fait en déterminant les poids du réseau et en les mettant à jour de sorte que la sortie soit proche de la sortie souhaitée, cela se fait via un algorithme de rétropropagation de gradient qui repose sur le calcul du gradient de sortie et puis après sont rétro-propagés à travers la fonction de seuil, suivis des poids. C'est pourquoi la fonction de seuil doit être dérivable. Par conséquent, la fonction d'activation Heaviside est remplacée par une fonction d'activation dérivable. La figure (2.2) présente les fonction d'activations les plus classiquement utilisées :

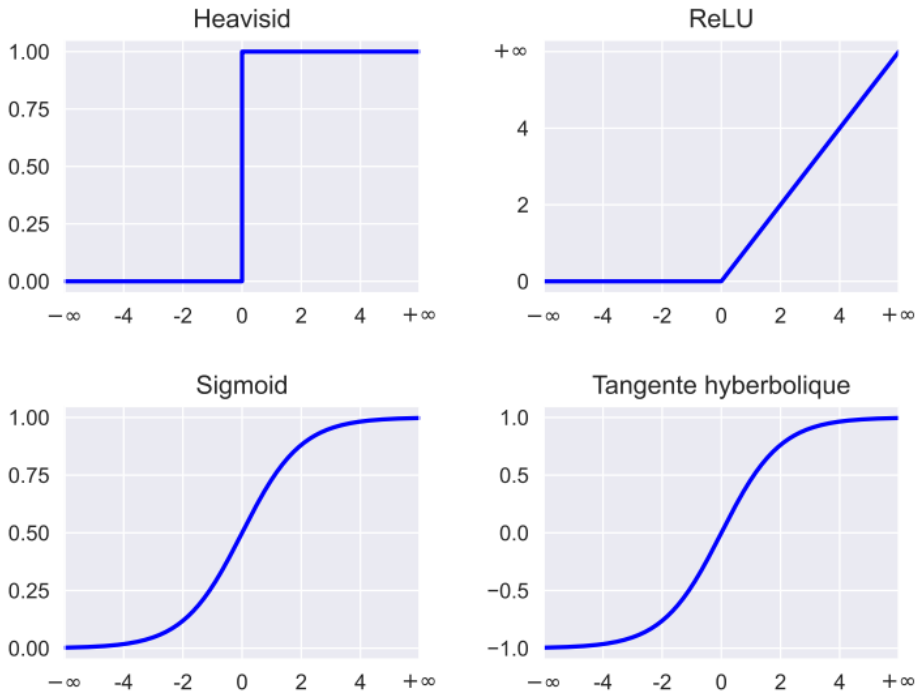


FIGURE 2.2 – Fonctions d'activation

2.1.3 La notion de rétropropagation du gradient

On définit l'apprentissage classique comme la minimisation de la fonction coût suivante :

$$\mathcal{L} = \frac{\|y - y_d\|^2}{2}$$

Où y est la sortie obtenu de perceptron et y_d est la sortie désirée.
La rétropropagation du gradient, c'est l'estimation des gradients de $\mathcal{L}$ par rapport aux poids en progressant de la sortie vers l'entrée du réseau de neurone :
Le gradient de l'erreur en sortie est donc exprimé par :

$$\frac{\partial \mathcal{L}}{\partial y} = y - y_d$$

Ainsi le gradient de l'erreur pour le potentiel z est :

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial y}\frac{\partial y}{\partial z} = \frac{\partial \mathcal{L}}{\partial y} \times a'(z)$$

Le gradient de l'erreur pour un poid w_i est :

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial z} \times x_i, \quad i = 1, \cdots, p$$

Le gradient de l'erreur pour une biais b_0 est :

$$\frac{\partial \mathcal{L}}{\partial b_0} = \frac{\partial \mathcal{L}}{\partial z}\frac{\partial z}{\partial b_0} = \frac{\partial \mathcal{L}}{\partial z} \times 1$$

Dans un apprentissage supervisé, les poids vont être corrigés de façon itérative. A l'itération k, étant données les poids w_i^k, composantes du vecteur w^k, on sélectionne aléatoirement un exemple x^k dont on connait la sortie y^k, l'utilisation de l'algorithme de descente du gradient implique que :

$$w_i^{k+1} = w_i^k - \alpha^k \frac{\partial \mathcal{L}(x^k, w^k)}{\partial w_i}$$

le taux d'apprentissage α^k peut améliorer significativement la convergence de l'algorithme. En générale, il est choisi 0.001. On définira en détail dans la sous section 2.2.2.
le problème du perceptron simple échoue pour des problèmes de classification où les classes ne sont pas linéairement séparables, et on peut pas classifier sur plusieurs classes ou voir même les relations profonds entre les données.

2.2 Le perceptron multicouche MLP

Dans le modèle du Perceptron Multicouches, les perceptrons sont organisés en couches. Les perceptrons multicouches sont capables de traiter des données qui ne sont pas linéairement séparables. Avec l'arrivée des algorithmes de rétropropagation, ils deviennent le type de réseaux de neurones le plus utilisé. Les MLP sont généralement organisés en trois couches, la couche d'entrée, la couche intermédiaire (dite couche(s) cachée(s)) et la couche de sortie. L'utilité de plusieurs couches cachées n'a pas été démontrée. La figure (2.3) illustre la structure d'un MLP présentant 4 neurones en entrée, $K = 2$ classes distinctes pour la sortie, et une seule couche cachée $L = 1$. Lorsque tous les neurones d'une couche sont connectés aux neurones de la couche suivante, on parle alors de couches complètement connectées.

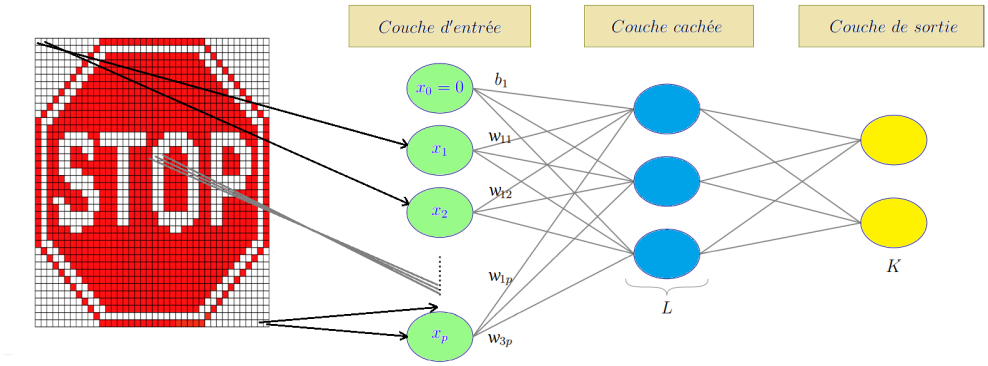


FIGURE 2.3 – Le perceptron multicouche

Quand il s'agit de la classification avec un réseau de neurones, la variable expliquée $y \in \mathbb{R}^K$ contient l'information pour la classe observée et $x \in \mathbb{R}^p$ est le vecteur d'entrants contenant les p variables explicatives. Une observation de la variable expliquée y prend la forme d'un vecteur où la k^{ime} composante prend la valeur 1 si la classe correspondante est observée.
C'est-à-dire que y a l'une des formes suivantes selon la classe associée :

$$\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix} \quad \cdots \quad \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

Ce problème est en fait modélisé à l'aide de la loi multinomiale. Un jeu de données contenant les N observations de la variable dépendante y est représenté au niveau probabiliste par les réalisations de N épreuves indépendantes d'une loi multinomiale.
Le modèle pour un réseau de neurones (MLP) est défini récursivement de la façon suivante :

$$x_{l+1} = \begin{cases} a(\theta_l x_l) & \text{si} \quad l = 1, \ldots, L-1 \\ f(\theta_L x_L) & \text{si} \quad l = L \end{cases}$$

Où L le nombre de couches cachées.
On définit la matrice de poids partitionnée $\theta_l = (b_l W_l)$ comme étant la matrice comprenant la matrice de poids $W_l \in \mathbb{R}^{n_{l+1} \times n_l}$ entre les neurones et le vecteur biais $b_l \in \mathbb{R}^{n_{l+1}}$. On prend les notations suivantes $n_0, n_1, \ldots, n_L, n_{L+1}$ pour signifier le nombre de neurones respectif dans chacune des $L+2$ couches du réseau allant de la couche d'entrée jusqu'à la couche de sortie.

Ainsi, x_0 est vu comme un vecteur de variables explicatives (entrées) pour le réseau, alors que x_{L+1} représente le vecteur ou scalaire de variable(s) expliquée(s) (sortie) par celui-ci. Les vecteurs $x_1, \ldots, x_L$ contiennent les neurones pour chacune des couches cachées du réseau.
On choisit la fonction d'activation a afin de faire les connexions entre les différentes couches cachées d'un réseau de neurones. On choisira une fonction d'activation pouvant être distincte pour la sortie du réseau que l'on dénote par f. Cette dernière transforme les neurones de la dernière couche en sortie pour le réseau.
Par exemple, pour une tâche de classification binaire, on aura le choix d'utiliser f égale à la fonction sigmoïde ou softmax selon que l'on utilise un ou deux neurones respectivement pour la couche de sortie. Dans le cas où l'on a une tâche de classification où le nombre de catégories est $K \geq 2$, la fonction softmax est souvent utilisée en pratique puisqu'elle permet d'associer un probabilité respective à chacune des différentes classes.

2.2.1 Inférence

On dénote l'ensemble des paramètres du modèle comme étant $\theta = \{\theta_0, \ldots, \theta_L\}$. Ainsi, on utilise la fonction :

$$S_\theta(x_0) = f(\theta_L a(\theta_{L-1} a(\theta_{L-2} \circ \circ \circ \theta_1 a(\theta_0 x_0))))$$

et on cherche

$$argmin_\theta \left(l(S_\theta(x_0), y) + \lambda R(\theta) \right)$$

Où y est une observation de la variable expliquée et R le régularisateur pour le modèle. On se sert donc du jeu d'entrainement. Supposons que ce dernier contienne N paires d'observations d'un vecteur de variables explicatives et expliquées. Alors on a :

$$(X_{0j}, y_j)\ j = 1, \ldots, N$$

Ce problème revient à minimiser

$$\mathcal{L}(\theta) = \frac{1}{N} \left[\sum_{j=1}^{N} l(S_\theta(x_{0j}), y_j) + \lambda R(\theta) \right]$$

2.2.2 Estimation du modèle

Algorithme de descente de gradient

L'approximation des paramètres pour un réseau de neurones se fait de la même façon que l'on aurait pu le faire pour un problème standard de minimisation en statistique ou en apprentissage automatique. On minimise une fonction de perte

par rapport a ses paramètres a laquelle s'ajoute un paramètre de régularisation. On cherche a minimiser une fonction sous forme sommation

$$\mathcal{L}(\theta) = \frac{1}{N}\left[\sum_{j=1}^{N} l(S_\theta(x_{0j}), y_j) + \lambda R(\theta)\right] \tag{2.1}$$

où θ^* est le paramètre qui minimise (2.1) à estimer. Une façon naturelle d'approcher θ^* est d'utiliser l'algorithme de descente de gradient (GD).
L'idée est donc de partir d'un point de départ pour θ_0 pour la fonction de perte afin de se diriger à petits pas vers un minimum en espérant que celui-ci soit un minimum global. Ainsi, on a une séquence de points $\theta_0, \theta_1, \theta_2, \ldots$, tels que pour :

$$\theta_{n+1} = \theta_n - \alpha\nabla\mathcal{L}(\theta_n), \quad n \geq 0$$

On a que :

$$\mathcal{L}(\theta_0) \geq \mathcal{L}(\theta_1) \geq \mathcal{L}(\theta_2) \geq \ldots$$

En pseudo-code, on a les étapes suivantes pour la minimisation :

1. On initialise les paramètres pour le réseau (θ_0).
2. Pour n itérations :
$$\theta_{n+1} = \theta_n - \alpha\nabla\mathcal{L}(\theta_n)$$

Il est connu que si la fonction objective est convexe, cet algorithme mène a une solution optimale.

Algorithme du gradient stochastique

Lorsque le nombre d'observations N est grand pour un certain jeu de données (dataset), évaluer le gradient de la somme défini en (2.1) peut devenir très lourd au niveau computationnel. C'est pourquoi, en pratique, l'algorithme du gradient stochastique (SGD) est davantage utilisé. Contrairement au GD, l'algorithme met à jour les paramètres pour chaque observation dans l'ensemble d'entraînement. Afin d'introduire l'algorithme, on prend ici $\mathrm{J}(\theta)_j$ la fonction de perte évaluée pour l'observation j, c'est-à-dire

$$\mathrm{J}(\theta)_j = l(S_\theta(x_{0j}), y_j)$$

Le SGD approxime la vraie valeur du gradient par sa valeur évaluée sur une observation. La convergence de l'algorithme repose en grande partie sur deux conditions. Si α_t est le pas de l'itération pour la $t^{\text{ième}}$ mise a jour des paramètres, on a besoin que $\alpha_t \to 0$, $\sum_t \alpha_t^2 < \infty$. Par exemple, on aurait que $\alpha_t = \frac{1}{t}$ serait un bon choix étant donné qu'il satisfait ces trois conditions. Essentiellement, lorsque α_t décroît

selon un taux approprié et que ces conditions sont respectées, le SGD converge vers le minimum global si la fonction de perte est convexe ou vers un minimum local si celle-ci est pseudoconvexe. Ces résultats proviennent en fait du théorème de Robbins-Siegmund (Robbins, 1971). Un choix couramment utilisé pour la vitesse de convergence $\alpha_t = \frac{\alpha_0}{1+\delta}$. On a ici que α_0 représente la valeur initiale pour le pas d'itération et δ est appelée constante de décroissance. On choisit habituellement $\delta < 10^{-3}$.
En pseudo-code, on a les étapes suivantes pour la minimisation :

1. On initialise les paramètres pour le réseau.
2. Pour n itérations :
3. Mélanger aléatoirement les observations dans l'ensemble d'entraînement.
4. pour chacune des observation dans l'ensemble d'entrainement définie à l'étape 3 :

$$\theta_{n+1} = \theta_n - \alpha_t(\nabla \mathrm{J}(\theta_n)_j + \nabla \frac{\lambda}{2} R(\theta_n))$$

On utilise le terme période «epoch» pour désigner n le nombre de fois où l'on passe sur l'ensemble d'entraînement. Lorsque N est grand, on peut apprendre de nos données en passant une seule fois sur le jeu de données. Par contre, plus N est petit, plus on devra passer de fois sur le jeu données pour que l'algorithme de descente de gradient puisse apprendre des données. Afin de déterminer le nombre optimal de périodes, on utilise souvent une procédure d'arrêt forcé «early stopping». L'idée derrière cette procédure est d'arrêter l'algorithme avant convergence afin d'éviter le sur-ajustement du modèle sur l'ensemble d'entraînement. En pratique, on peut représenter sur un graphique l'erreur moyenne sur l'ensemble d'entraînement à laquelle on compare l'erreur moyenne sur l'ensemble de validation. Au départ, l'erreur moyenne va diminuer sur les deux ensembles jusqu'à un certain point où l'on aura que l'erreur sur l'ensemble d'entraînement va diminuer alors que celle sur l'ensemble validation va commencer à augmenter. On peut percevoir ce point comme étant une indication de sur-ajustement pour le modèle sur l'ensemble d'entraînement.

Algorithme du gradient stochastique «Mini-Batch»

Un compromis entre le GD et le SGD est d'utiliser le gradient stochastique «Mini-Batch». Au lieu d'approximer la valeur du gradient pour chaque observation dans le jeu de données, on peut utiliser à chaque fois certain nombre d'observation choisi aléatoirement que l'on regroupe en «mini-batch» d'une certaine grandeur, disons b, pour calculer le gradient. On arrive ainsi à réduire la variance de nos paramètres lorsque ceux-ci sont réévalués. Ce qui peut mener à une convergence relativement plus stable des paramètres. L'autre avantage par rapport au SGD

est que l'on peut accélérer le calcul du gradient en utilisant certaines librairies permettant l'optimisation matricielle. Avec une bonne vectorisation, le gradient stochastique «mini-batch» permet de traiter plusieurs observation en parallèle et par conséquent, il peut être plus efficace que le SGD.
En pseudo-code, on a les étapes suivantes pour la minimisation :

1. On initialise les paramètres pour le réseau.
2. Pour n itérations :
3. Mélanger aléatoirement les observations dans l'ensemble d'entraînement.
4. pour chacune des «mini-batch» c-à-d, pour la séparation suivante de l'ensemble d'entraînement de l'étape 3 : $(i = 1, b + 1, 2b + 1, \ldots, N - b + 1)$;

$$\theta_{n+1} = \theta_n - \alpha_t(\nabla \frac{1}{b} \sum_{j=1}^{i+b-1} \mathrm{J}(\theta_n)_j + \nabla \frac{\lambda}{2} R(\theta_n))$$

Conclusion

Dans ce chapitre nous avons donné les principes des réseaux de neurones et leur architecture générale. Pratiquement il y a différentes architectures ça dépend du problème et de type de données. Ce qui est le cas dans notre mémoire, le MLP échoue lorsque on a des images en entrée car si on associe à chaque pixel un neurone on aura donc une explosion des paramètres du modèles. Pour remédier à ce problème nous allons présenté dans le chapitre suivant les réseaux de neurones convolutionnels.

Chapitre 3

Les réseaux de neurones convolutionnels

Les réseaux de neurones convolutionnels (CNNs) sont des algorithmes du deep learning spécialisés dans les tâches de reconnaissance d'image et parmi ces tâches on trouve la classification. Ainsi ils ont une méthodologie similaire à celle des méthodes traditionnelles d'apprentissage supervisé : ils reçoivent des images en entrée, détectent les features automatiquement de chacune d'entre elles, puis entraînent un classifieur dessus. Donc les CNNs réalisent eux-mêmes tout le boulot fastidieux d'extraction et description de features.

3.1 Réseaux de neurones convolutionnels et MLP

Les perceptrons multicouches (MLP) ont des difficultés à gérer des images de grande taille, en raison de le nombre de connexions augmente de façon exponentielle avec la taille de l'image, du fait que chaque neurone est « totalement connecté » à chacun des neurones de la couche précédente et suivante.
A l'inverse, les réseaux de neurones convolutionnels limitent le nombre de connexions entre un neurone et les neurones des couches adjacentes, ce qui réduit fortement le nombre de paramètres à apprendre.
Les réseaux de neurones convolutionnels sont conçus pour limiter le nombre d'entrées tout en préservant les fortes corrélations "spatialement locale" des images naturelles. Par rapport à MLP, CNN présente les principales caractéristiques suivantes :

— **Champ récepteur** Dans un réseau neuronal traditionnel, chaque neurone cachée est connecté à chaque neurone de la couche précédente. Les réseaux de neurones convolutionnels, cependant, ont une architecture de champ récepteur, c'est-à-dire que chaque neurone cachée ne peut se connecter qu'à une petite région de l'entrée appelée champ récepteur. Pour cela, le filtre

(poids) est plus petit que l'entrée. Avec le champ récepteur, les neurones peuvent extraire des caractéristiques visuelles élémentaires comme les bords, les coins etc...

— **Volumes 3D de neurones :** La couche de neurones n'est plus seulement une surface (perceptron), mais un volume profond. Étant donné un seul champ récepteur sur le CNN, les neurones associés (profondeurs) correspondent à la première couche de MLP.

— **Connectivité locale :** Étant donné que le champ récepteur limite le nombre d'entrées aux neurones, tout en préservant l'architecture MLP, le réseau neuronal convolutionnel garantit ainsi que le "filtre" produit la réponse la plus forte aux modèles d'entrée spatialement locaux, ce qui entraîne une représentation minimaliste de l'entrée. Cette représentation prend moins de place en mémoire. De plus, leurs estimations (statistiques) sont plus robustes à une quantité fixe de données (par rapport à MLP) en raison du nombre réduit de paramètres à estimer.

— **Poids partagés :** Dans un réseau de neurones convolutionnels, les paramètres de filtrage d'un neurone (pour un champ récepteur donné) sont les mêmes pour tous les autres neurones du même noyau (tous les autres champs récepteurs qui traitent l'image). Le partage de poids fait référence à l'utilisation du même filtre pour tous les champs récepteurs d'une couche. Dans un CNN, puisque les filtres sont plus petits que l'entrée, chaque filtre est appliqué à chaque position de l'entrée.

— **Invariance à la translation :** Comme tous les neurones d'un même noyau (filtre) sont identiques, les motifs détectés par ce noyau sont indépendants de la localisation spatiale dans l'image.

Ces propriétés permettent aux réseaux de neurones convolutionnels d'obtenir une meilleure robustesse dans l'estimation des paramètres pour les problèmes d'apprentissage, puisque pour une taille de corpus d'apprentissage fixe, la quantité de données par paramètre est plus importante. Le partage de poids peut également réduire considérablement le nombre de paramètres libres à apprendre, réduisant ainsi les besoins en mémoire pour les opérations du réseau. L'empreinte mémoire réduite permet l'apprentissage de réseaux plus grands, et donc généralement plus puissants.

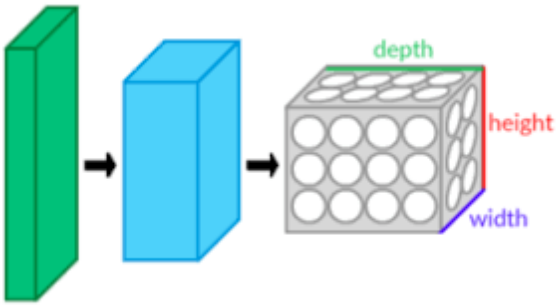


FIGURE 3.1 – Une couche du CNN en 3 dimensions. (Vert = volume d'entrée, bleu = volume du champ récepteur, gris = couche de CNN, cercles = neurones artificiels indépendants).

3.2 Architecture du réseau de neurone convolutionnel

On utilise habituellement trois types de couches pour former un réseau convolutionnel qui a comme entrée une image, celui-ci est composé de couche(s) de convolution (Conv), de couche(s) de pooling (Pool) et de couche(s) entièrement connectées (FC).
La ou les couches entièrement connectées son souvent utilisées pour la sortie du réseau. Habituellement, une couche de convolution est suivie d'une fonction d'activation puis d'une couche de pooling, cette séquence peut être répétée plusieurs fois jusqu'à la couche FC pour former un réseau convolutionnel que l'on dénote souvent sous la notation CONVNET. Notre modèle aura donc une architecture comme montre la figure (3.2). Où les trois étapes intermédiaires Conv - Activation - Pool peuvent se répéter plusieurs fois avant la couche FC. Aussi il est commun d'utiliser plus qu'une couche FC avant la sortie du réseau [8].

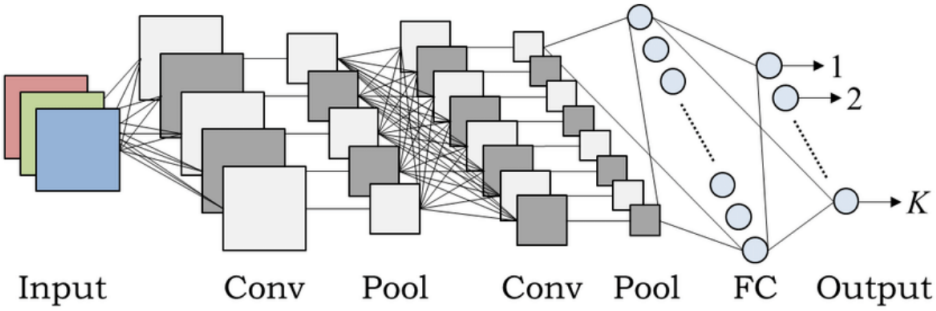


FIGURE 3.2 – CONVNET.

Il existe plusieurs architectures dans la domaine de CNNs, par exemple LeNet, AlexNet, ResNet ...
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.

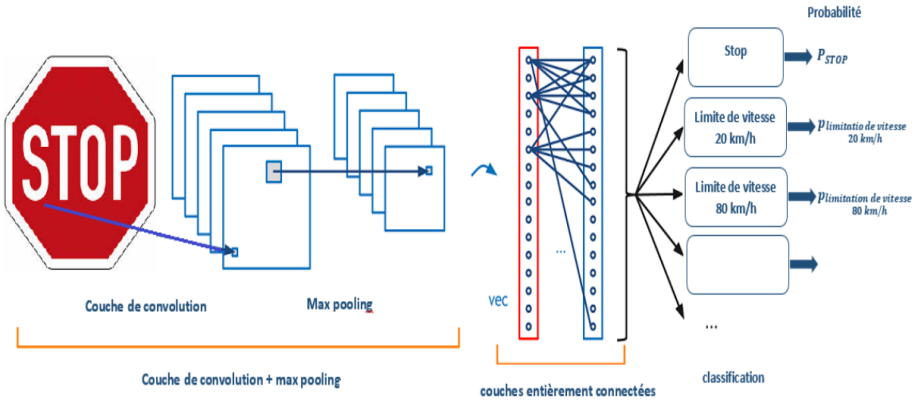


FIGURE 3.3 – Architecture d'un réseau de neurone convolutionnel.

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

Input : Puisque on utilise un réseau convolutionnel pour la classification d'images, l'entrant sera représenté sous forme d'un volume. Pour le traitement à l'intérieur du réseau, l'image est habituellement représentée sous forme d'un volume dont la largeur est égale a la hauteur. La profondeur du volume représente le nombre de canaux. Pour les images couleur, il existe trois canaux qui représentent les intensités du rouge, du vert et du bleu. Dans les images

en niveaux de gris, il n'y a qu'un seul canal. Nous utilisons également le terme d'entrée dans un sens plus large pour désigner une couche d'entrée particulière, quel que soit son emplacement dans le réseau.

Conv : Pour passer d'une couche d'entrée ou de pooling a une couche de convolution, on applique une convolution au sens mathématique. D'où le nom réseau de neurones convolutionnels. La convolution peut transformer la hauteur, la largeur et la profondeur du volume d'entrée. On définira la convolution plus en détail dans la section 3.3.

Activation : la fonction d'activation est appliquée après la convolution et laisse la taille du volume inchangé. Par exemple, si on a une fonction d'activation ReLU, on applique la fonction $f(x) = max(0, x)$ à chaque élément du volume associé à la couche [4].

Pool : la couche pooling réduit les dimensions (hauteur et largeur) du volume entrant sans affecter la profondeur. Les opérations les plus courante sont le max-pooling et le moyenne-pooling «average pooling».

FC : les neurones d'une couche entièrement connectée sont connectés avec chacun des neurones du volume correspondant à son entrant, c'est-à-dire avec chacun des neurones de la couche précédente. Si on décompose notre volume d'entrants sous forme vectorielle, on a exactement un réseau de neurones MLP.

3.3 Convolution

La convolution est l'une des opérations les plus importantes du traitement signaux, qui sont utilisés dans plusieurs applications de traitement du langage naturel, Vision par ordinateur et traitement d'images. L'opération de convolution peut également Appliqué aux fonctions multidimensionnelles.
Nous notons que I est l'image d'entrée, K est le filtre $2D$ dont la dimension est $m \times n$, et F est la carte de caractéristiques, qui représente le résultat de la convolution de l'image avec le filtre. Cette opération peut être exprimée mathématiquement comme suit :

$$F(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Dans la convolution le filtre K doit être inversé par rapport à l'image en entrée. Dans le cas où K n'est pas inversé, l'opération de convolution sera la même formule que la corrélation croisée, et cela peut être formulé mathématiquement par :

$$F(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Plusieurs bibliothèques d'apprentissage profond utilisent la corrélation croisé au lieu de la convolution car son implémentation est plus pratique.
En termes simples, la convolution est l'application d'un filtre mathématique à une image. Techniquement, il s'agit de faire glisser une matrice par-dessus une image, et pour chaque pixel, utiliser la somme de la multiplication de ce pixel par la valeur de la matrice. Cette technique nous permet de trouver des parties de l'image qui pourraient nous être intéressantes. Prenons la Figure ci-dessous à gauche comme exemple d'image et la Figure à droite comme exemple de filtre.

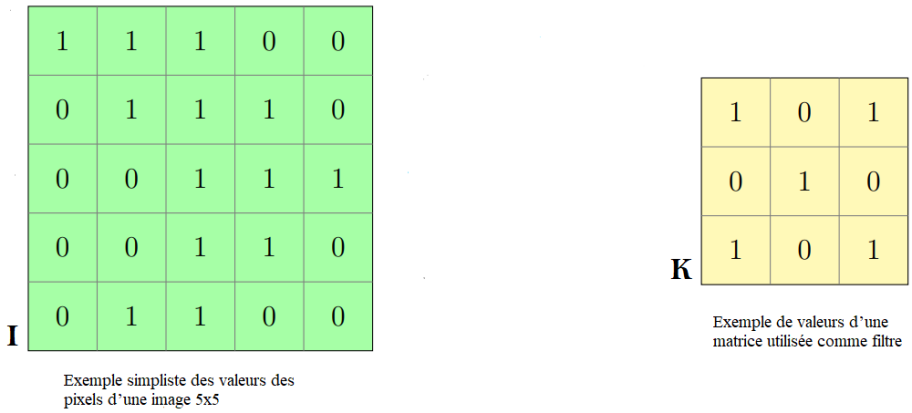


FIGURE 3.4 – Image et filtre.

Appliquer le filtre sur l'image : dans la matrice image I, nous pouvons voir que chaque valeur des pixels de l'image tuile (les cases orange) est multipliée par chaque valeur correspondante du filtre ($1 \times 1, 1 \times 0, 1 \times 1 \ldots$). Puis additionner tous ces valeurs pour obtenir une seule valeur '4' qui fera partie d'une nouvelle image convoluée.

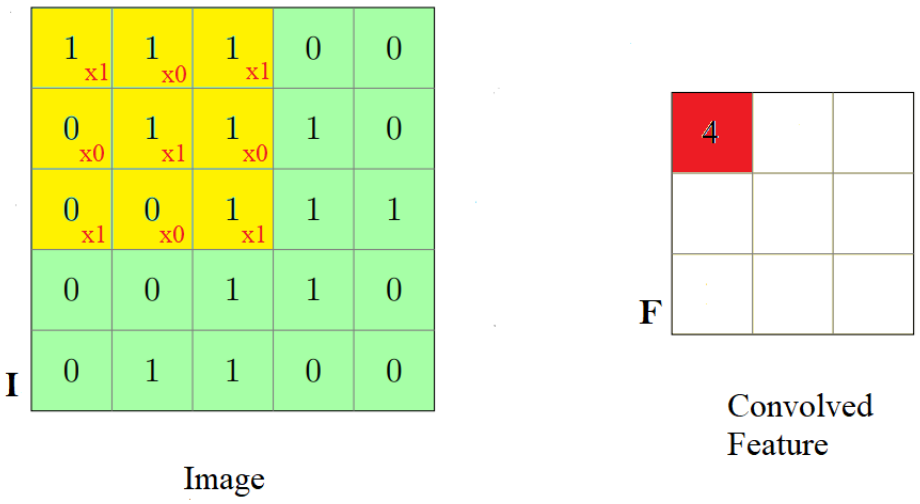


FIGURE 3.5 – Image et Convolved Feature.

Le filtre doit se déplacer d'une case à chaque itération jusqu'à ce que la première ligne soit finie. Lorsque nous avons fini la première ligne, le filtre « descend » d'une case et la même procédure se répète pour chaque ligne et colonne. Voir l'animation suivante :

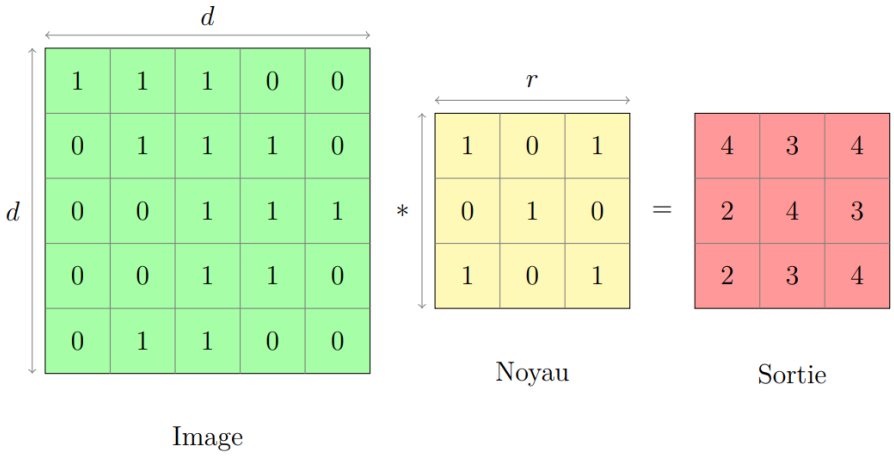


FIGURE 3.6 – Exemple de convolution en 2 dimensions avec un pas s = 1 et sans marge à zéros.

Noté qu'une convolution 3×3 de profondeur 1 effectuée sur une carte de caractéristiques d'entrée 5×5, également de profondeur 1. Comme il y a neuf emplacements 3×3 possibles pour extraire les tuiles de la carte de caractéristiques

5×5, cette convolution génère une carte de caractéristiques de sortie 3×3. Un réseau de neurones à convolution contient de multiples filtres et ces filtres sont appliqués sur l'image d'origine. Après la première étape nous avons donc autant de nouvelles images que de filtres. La phase de convolution peut aussi être vue comme des couches de neurones cachées où chaque neurone n'est connecté qu'à quelques neurones de la couche suivante.

Il est nécessaire d'introduire quelques concepts pour bien comprendre comment la convolution fonctionne dans le cadre d'un réseau convolutionnel. Les trois hyperparamètres suivants détermineront la dimension de la sortie associée à notre couche de convolution.

— **le pas :** lors de l'exécution d'une convolution, on choisit ce qu'on appelle pas «stride», c'est à dire le pas auquel on déplace le noyau à travers l'entrant. Le pas horizontal représente le déplacement horizontal du noyau, ainsi que le pas vertical. Dans la suite on utilisera le pas horizontalement égale à le pas verticalement.

— **La profondeur :** le nombre de filtres est en fait le nombre de noyaux (matrices de poids) choisis par couche de convolution. Le nombre de filtres déterminera la profondeur de la sortie associée à notre couche.

— **La marge à zéro :** (Padding) il est commun d'utiliser une certaine épaisseur de marge de zéros autour de l'entrant afin de contrôler la hauteur et la largeur du volume de sortie.

La dimension du volume sortant (largeur et hauteur) est donnée par la formule suivante :

$$\frac{d - r + 2p}{s} + 1$$

Où p est le margin à zéro qui égale 0 dans cet exemple, et s le pas de déplacement du noyau sur l'image.

Notée bien que la profondeur de chaque entrée est aussi varie celon le nombre de filtres (nombre de neurones par analogie avec un MLP).

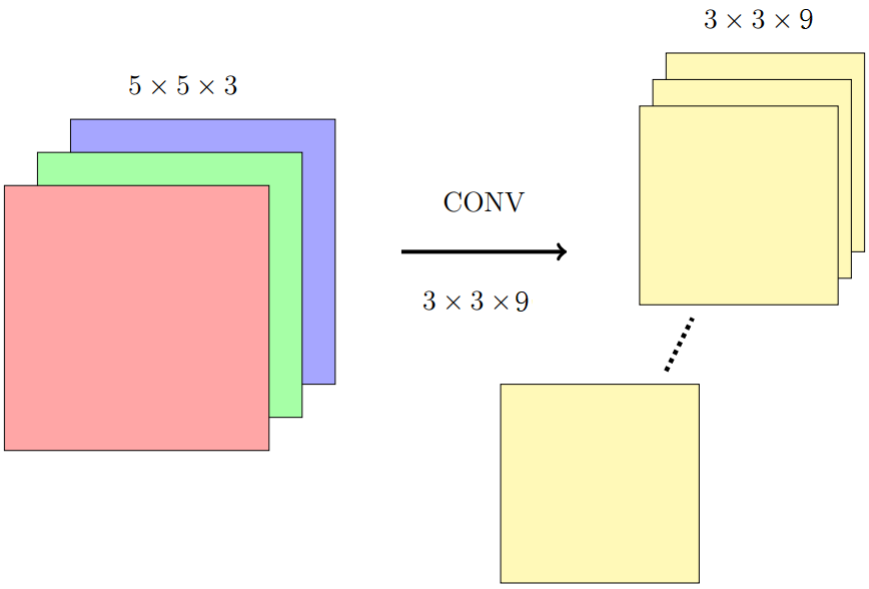


FIGURE 3.7 – La sortie de la couche Conv avec 9 filtres.

Dans notre exemple nous avons un entrée de taille $5 \times 5 \times 3$ avec une couche de convolution qui contient 9 filtres chaqun de taille $3 \times 3 \times 3$, alors la sortie est de taille $3 \times \times 3 \times 9$.
Notée bien que chaque filtre doit avoir la même profondeur que de l'entrée.

3.4 Pooling

Ce type de couche est souvent placé entre deux couches de convolution : elle reçoit en entrée plusieurs feature maps, et applique à chacune d'entre elles l'opération de pooling. L'opération de pooling consiste à réduire la taille des images, tout en préservant leurs caractéristiques importantes. Pour cela, on découpe l'image en cellules régulière, puis on garde au sein de chaque cellule la valeur maximale.

La méthode utilisée consiste à imaginer une fenêtre de 2 ou 3 pixels qui glisse au-dessus d'une image, comme pour la convolution. Mais, cette fois-ci, nous faisons des pas de 2 pour une fenêtre de taille 2, et des pas de 3 pour 3 pixels. La taille de la fenêtre est appelée « kernel size » et les pas s'appellent «strides ». Pour chaque étape, nous prenons la valeur la plus haute parmi celles présentes dans la fenêtre et cette valeur constitue un nouveau pixel dans une nouvelle image. Ceci s'appelle Max-Pooling.

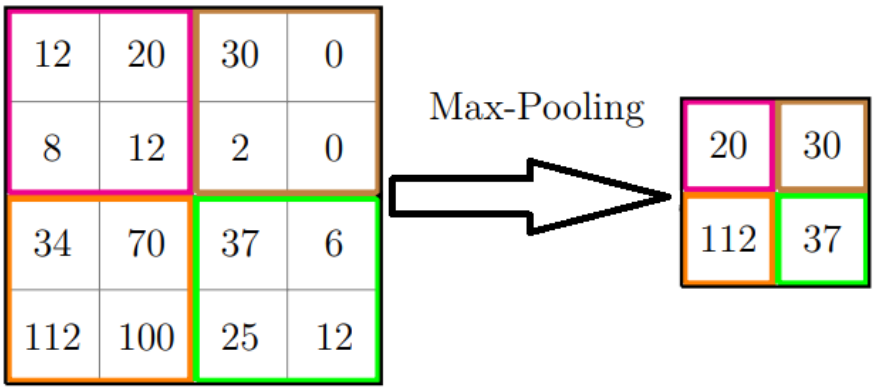


FIGURE 3.8 – Max-Pooling.

La couche de pooling permet de réduire le nombre de paramètres et de calculs dans le réseau. On améliore ainsi l'efficacité du réseau et on évite le surapprentissage.

Les valeurs maximales sont repérées de manière moins exacte dans les feature maps obtenues après pooling que dans celles reçues en entrée - c'est en fait un grand avantage ! En effet, lorsqu'on veut reconnaître un chien par exemple, ses oreilles n'ont pas besoin d'être localisées le plus précisément possible : savoir qu'elles se situent à peu près à côté de la tête suffit !
Ainsi, la couche de pooling rend le réseau moins sensible à la position des features : le fait qu'une feature se situe un peu plus en haut ou en bas, ou même qu'elle ait une orientation légèrement différente ne devrait pas provoquer un changement radical dans la classification de l'image.

Il existe aussi d'autres types de pooling comme "average pooling" qui prend la moyenne entre les valeurs de patch.

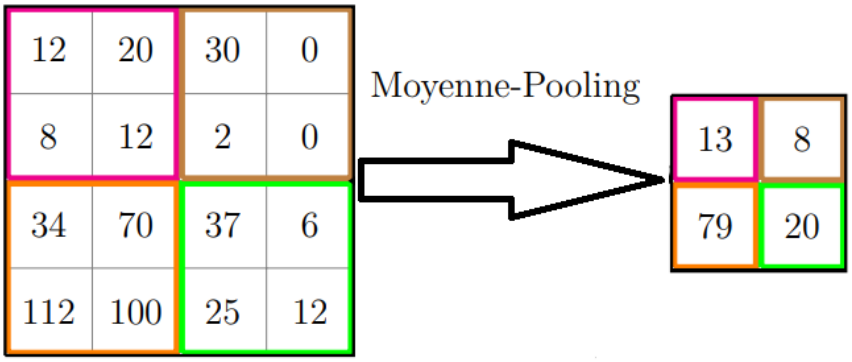


FIGURE 3.9 – Average-Pooling.

Notée bien que l'opération pooling ne change pas la profondeur de l'entrant, or il réduit la la hauteur et la largeur [3].

3.5 Fully-Connected

La couche Fully-Connected constitue toujours la dernière couche d'un réseau de neurones. Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie. Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée. La dernière couche Fully-Connected permet de classifier l'image en entrée du réseau : elle renvoie un vecteur de taille N, où N est le nombre de classes dans notre problème de classification d'images. Chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir à une classe. Par exemple, si le problème consiste à distinguer les chats des chiens, le vecteur final sera de taille 2 : le premier élément (respectivement, le deuxième) donne la probabilité d'appartenir à la classe "chat" (respectivement "chien"). Ainsi, le vecteur $[0.9, 0.1]$ signifie que l'image a 90% de chances de représenter un chat.
Pour calculer les probabilités, la couche Fully-Connected multiplie donc chaque élément en entrée par un poids, fait la somme, puis applique une fonction d'activation (logistique si $N = 2$, softmax si $N > 2$) : Ce traitement revient à multiplier le vecteur en entrée par la matrice contenant les poids. Le fait que chaque valeur en entrée soit connectée avec toutes les valeurs en sortie explique le terme Fully-Connected.

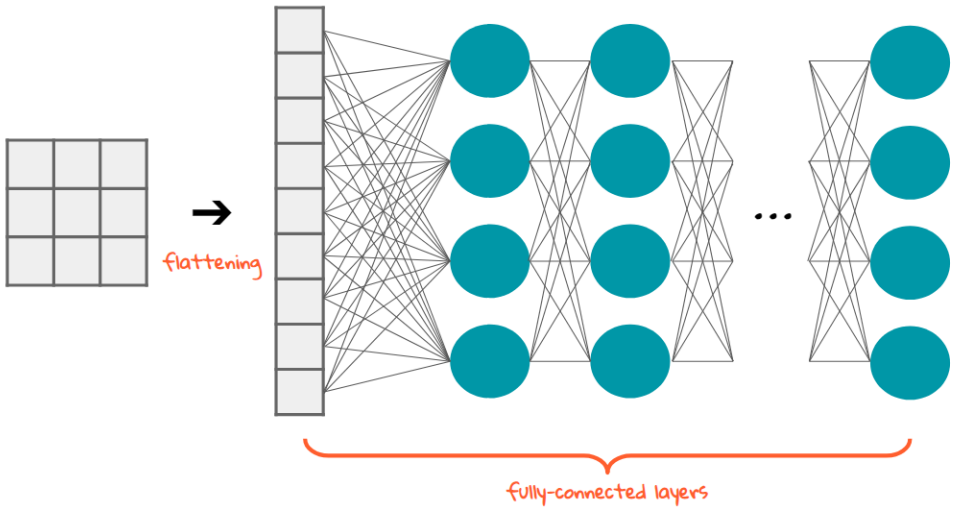


FIGURE 3.10 – La couche Fully-Connected.

3.6 Choix des hyperparamètres

Malheureusement, tous les aspects de CNN ne sont pas aussi intuitifs à apprendre et à comprendre que nous l'avons vu. Par conséquent, il y a toujours une longue liste de paramètres qui doivent être définis manuellement pour que le CNN obtienne de meilleurs résultats. Par conséquent, il est nécessaire de considérer la notion de nombre de filtres et leur forme, ainsi que la forme de max-pooling.

3.6.1 Nombre de filtres

Le nombre de filtres est le nombre de neurones, où chaque neurone effectue une convolution différente sur l'entrée de la couche (plus précisément, le poids de l'entrée du neurone forme le noyau de convolution). Pour une meilleure compréhension, différents filtres de convolution sont appliqués à l'image d'entrée et différentes cartes de caractéristiques (sortie de filtre) sont générées. Chaque pixel de chaque carte de caractéristiques est la sortie de la couche de convolution.

3.6.2 Forme du filtre

Ils sont généralement sélectionnés en fonction de l'ensemble de données. Ils sont généralement choisis en fonction de l'ensemble de données. Les meilleurs résultats pour les images (32×32) se situent généralement dans la gamme de 5×5 sur la première couche, mais les ensembles de données d'images naturelles ont tendance à utiliser des filtres de de première couche de 12×12 ou bien 15×15.

3.6.3 Forme de max-pooling :

La valeur typique est 2×2. Une très grande taille d'entrée peut justifier un pooling 4×4 dans la première couche.
Cependant, le choix d'une forme plus grande peut réduire considérablement la taille du signal et entraîner la perte de beaucoup d'informations.

3.7 Méthodes de régularisation

Pour améliorer les performances et la généralisation de l'algorithme d'apprentissage et éviter le sur-apprentissage, des méthodes de régularisation peuvent être utilisées :

3.7.1 Empirique

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 qui ralentit le traitement de l'information. Pour éviter cela, la méthode du dropout consiste à « désactiver » aléatoirement des neurones (avec une probabilité prédéfinie, généralement 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 augmente la vitesse d'apprentissage, mais en déconnectant les neurones, elle limite également les effets marginaux, rendant le réseau plus robuste et capable de mieux généraliser les concepts appris.

DropConnect

Le DropConnect est une évolution du dropout, où on n'éteint pas non plus les neurones, le DropConnect consistant à inhiber une connexion (l'équivalent de la synapse), et toujours de manière aléatoire. Les résultats sont similaires au dropout (rapidité, capacité de généralisation de l'apprentissage), mais montrent des différences dans l'évolution des poids de connexion. Une couche « complètement connectée » avec un DropConnect peut s'apparenter à une couche à connexion « diffuse ».

3.7.2 Explicite

Taille de réseau

Le moyen le plus simple de limiter le sur-apprentissage est de limiter le nombre de couches dans le réseau et de libérer les paramètres libres (connexions) du réseau.

Dégradation des poids

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

— **La 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).

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 MLP. 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 hyperparamètres 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 4

Résultats expérimentaux

Dans ce chapitre, on va définir l'architecture de notre modèle implémenté qu'on a crée 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.

✓ Prétraiter et augmenter l'ensemble de données, si nécessaire.

✓ Développer un modèle CNN.

✓ Former et valider le modèle.

✓ Optimiser le modèle en expérimentant différents hyperparamètres.

✓ Tester le modèle avec l'ensemble de données de test.

4.1 Outils utilisés dans l'implémentation

4.1.1 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.

4.1.2 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.

4.1.3 Keras

Keras est une API de réseaux de neurones de haut niveau, écrite en Python et capable de fonctionner sur TensorFlow ou Theano. Il a été développé en mettant l'accent sur l'expérimentation rapide. Être capable d'aller de l'idée à un résultat avec le moins de délai possible est la clé pour faire de bonnes recherches. Il a été développé dans le cadre de l'effort de recherche du projet ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), et son principal auteur et mainteneur est François Chollet, un ingénieur Google.
En 2017, l'équipe TensorFlow de Google a décidé de soutenir Keras dans la bibliothèque principale de TensorFlow. Chollet a expliqué que Keras a été conçue comme une interface plutôt que comme un cadre d'apprentissage end to end. Il présente un ensemble d'abstractions de niveau supérieur et plus intuitif qui facilitent la configuration des réseaux neuronaux indépendamment de la bibliothèque informatique de backend. Microsoft travaille également à ajouter un backend CNTK à Keras aussi [2].

4.1.4 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 NumPy et SciPy. Matplotlib est distribuée librement et gratuitement sous une licence de style BSD.

Sa version stable actuelle (la 2.0.1 en 2017) est compatible avec la version 3 de Python [4].

4.2 Présentation sur les panneaux de la signalisation routière

4.2.1 La base d'image GTSR

La base de données GTSR (mydata) représente l'ensemble de données de 51800 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 4.1 – Les classes des panneaux de signalisation.

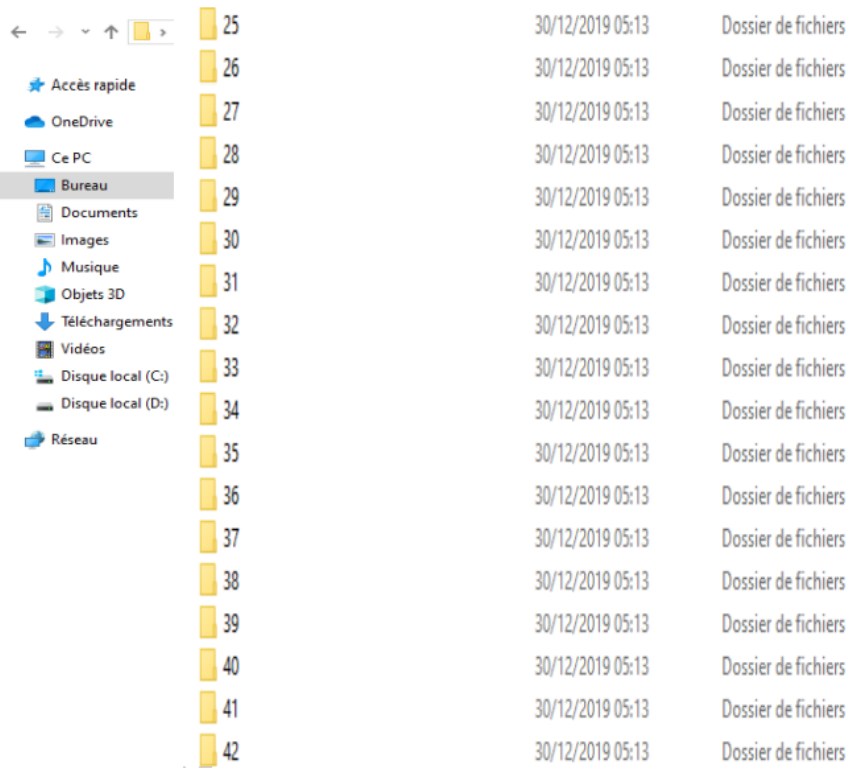
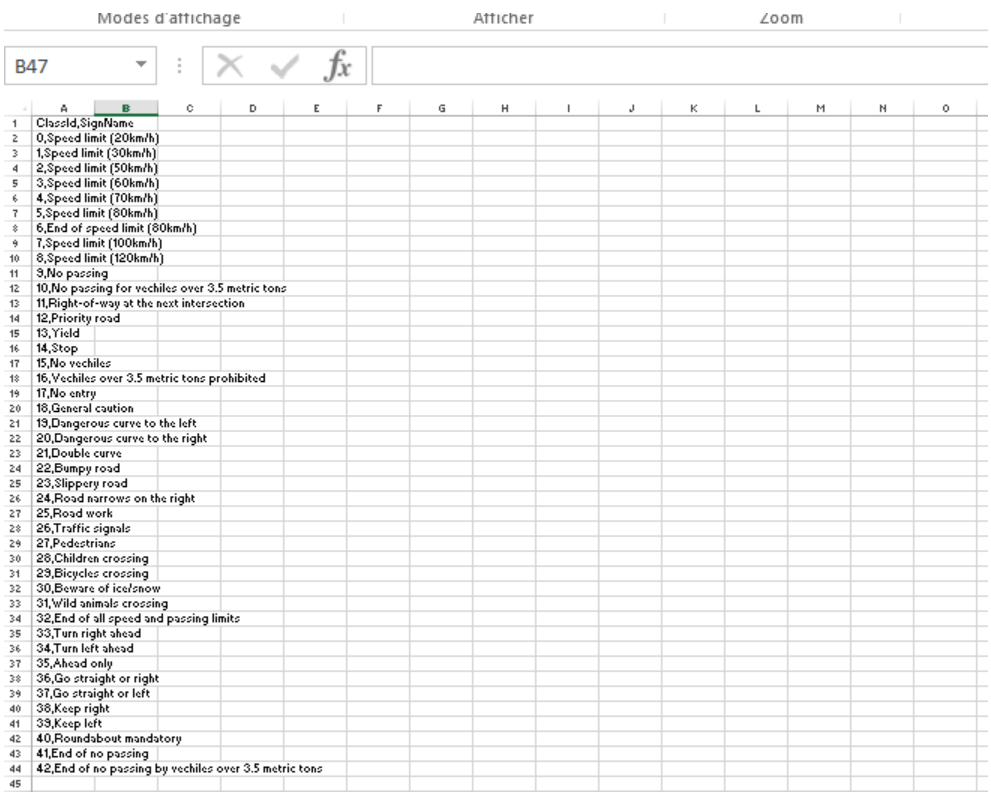


FIGURE 4.2 – MyData GTSR.

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 ClassId donc 0 représente la limite de vitesse de 20 alors que 14 représente stop et ainsi de suite.



ClassId	SignName
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
4	Speed limit (70km/h)
5	Speed limit (80km/h)
6	End of speed limit (80km/h)
7	Speed limit (100km/h)
8	Speed limit (120km/h)
9	No passing
10	No passing for vechiles over 3.5 metric tons
11	Right-of-way at the next intersection
12	Priority road
13	Yield
14	Stop
15	No vechiles
16	Vechiles over 3.5 metric tons prohibited
17	No entry
18	General caution
19	Dangerous curve to the left
20	Dangerous curve to the right
21	Double curve
22	Bumpy road
23	Slippery road
24	Road narrows on the right
25	Road work
26	Traffic signals
27	Pedestrians
28	Children crossing
29	Bicycles crossing
30	Beware of ice/snow
31	Wild animals crossing
32	End of all speed and passing limits
33	Turn right ahead
34	Turn left ahead
35	Ahead only
36	Go straight or right
37	Go straight or left
38	Keep right
39	Keep left
40	Roundabout mandatory
41	End of no passing
42	End of no passing by vechiles over 3.5 metric tons

FIGURE 4.3 – les étiquettes.

4.2.2 Exploration et visualisation de l'ensemble de données

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 :

Prétraitement des images

Niveaux de gris :

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

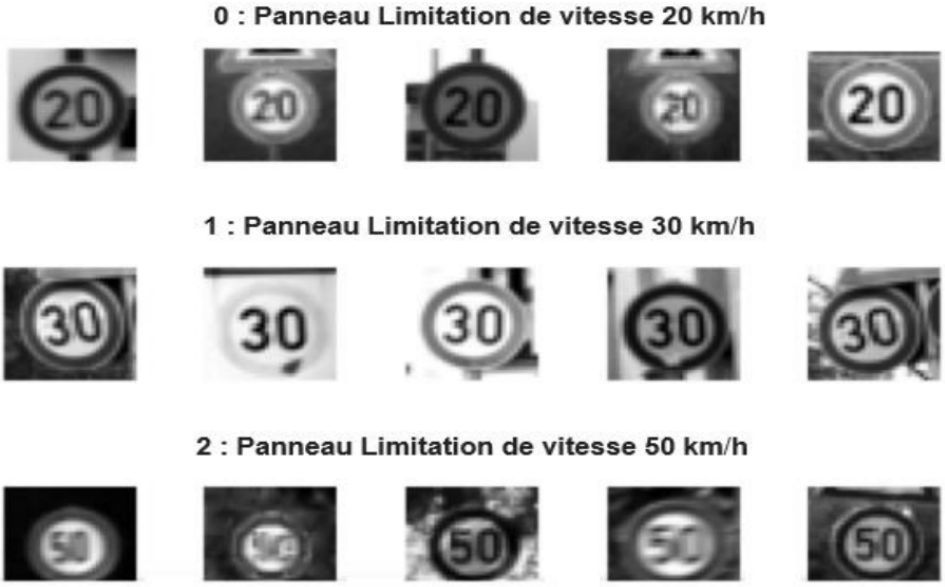


FIGURE 4.4 – Transformer des images RGB en niveau de gris.

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 :

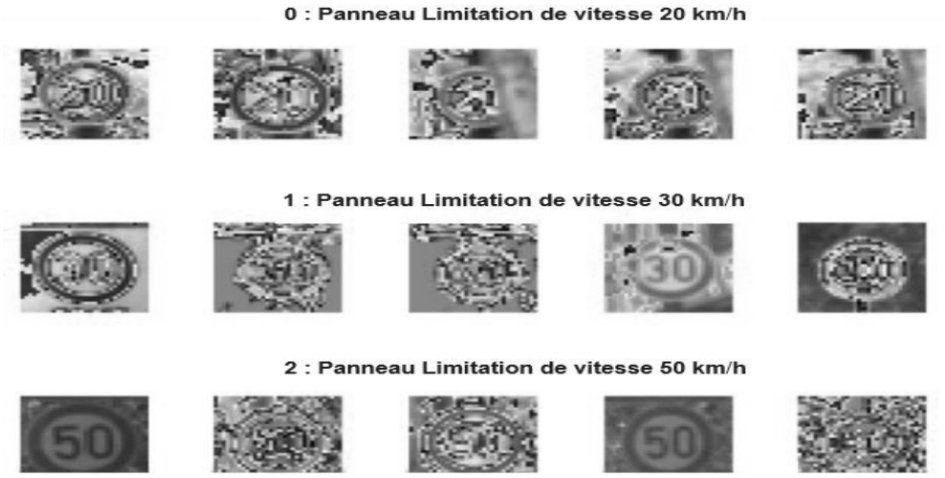


FIGURE 4.5 – Normalisation des images.

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 sur représentées, 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.

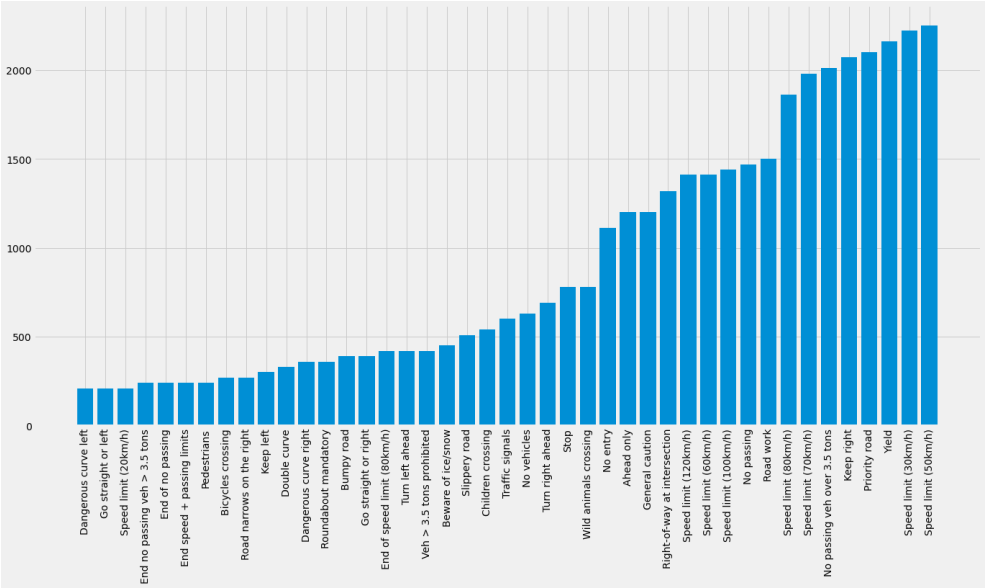


FIGURE 4.6 – Distribution d'échantillons pour chaque classe.

La quantité de données dont nous disposons n'est pas suffisante pour que le modèle se généralise correctement. C'est aussi assez déséquilibré, certaines catégories étant moins représentées que d'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. Nous pouvons l'augmenter grâce à la fonction ImageDataGenerator.

4.3 Architecture de notre modèle implémenté

Dans nos expériences, nous avons créé un modèle proche de leNet5 (Lenet5 modifié), où nous avons appliqué le modèle basé sur l'image GTSR.

Le modèle implémenté que nous montrons dans la figure ci-dessous se compose de quatre couches de convolution, deux couches de max-pooling et de deux couches entièrement connectées (Fully-Connected).
Dans la phase d'entraînement nous avons choisi le nombre d'epochs jusqu'à l'apprentissage n'augmente plus, et nous avons éviter le phénomène de sur-apprentissage à cause des couches Dropout et Batch-Normalization.

— La couche Batch-normalization a but de normaliser les valeur et réduire le temps d'exécution [5].

— La couche Dropout a but d'éviter le sur-apprentissage, i.e l'efficacité du réseau augmente.

— Softmax : c'est une fonction d'activation qui fait associer à chaque classe une probabilité par entrant, définit on (4.1).

$$Softmax(f)_i = \frac{\exp^{f_i}}{\sum_{j=1}^{C} \exp^{f_i}}, \text{ pour } i = 1, 2, \ldots, C \tag{4.1}$$

L'image en entrée est de taille $32 \times 32 \times 3$, après la transformation en niveaux de gris la taille sera $32 \times 32 \times 1$, l'image passe d'abord à la première couche de convolution. Cette couche se compose de 32 filtres de taille 3×3, la fonction d'activation ReLU est utilisée, cette fonction d'activation force les neurones à retourner des valeurs positives, après cette convolution 32 features maps seront créés de taille 30×30 (la taille spatiale du volume de sortie).

$$\frac{32 - 3 + 2 \times 0}{1} + 1 = 30$$

Ensuite, les 32 cartes de caractéristiques (feature maps) obtenues sont utilisées comme entrée dans la deuxième couche de convolution qui est composée de 64 filtres. La fonction d'activation ReLU est appliquée sur cette couche. Le Max-pooling est appliqué après pour réduire la taille de l'image. À la sortie de cette couche, nous aurons 64 feature maps de taille 14×14.
On répète la même chose avec les couches de convolutions 3 et 4 composées respectivement de 64 et 128 filtres, la fonction d'activation ReLU est toujours appliquée à chaque convolution. Une couche de Max-pooling est appliquée après la couche de convolution quatre. À la sortie de cette couche, nous aurons 128 feature maps de taille 5×5. Le vecteur de caractéristiques issu des convolutions a une dimension de 3200.
On applique la couche Batch-normalization($axis = -1$) après chaque couche de Max-pooling.
Après ces quatre couches de convolution, nous utilisons un réseau de neurones composé de deux couches Fully-Connected. La première couche est composée de

512 neurones où la fonction d'activation utilisée est la fonction ReLU suivie par deux couche (Batch-normalization() et Dropout(0.5)), 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).

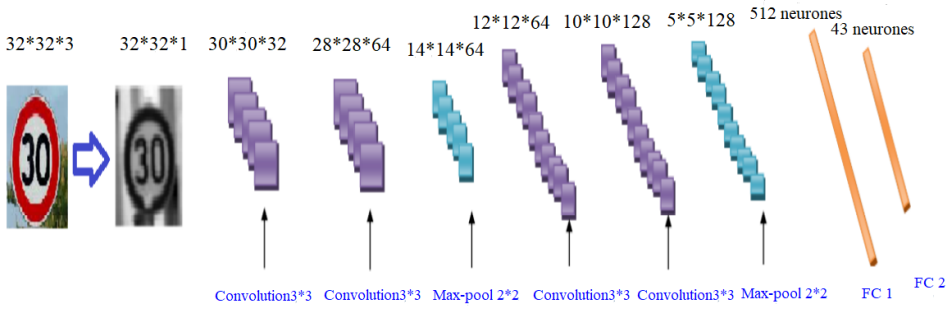


FIGURE 4.7 – Architecture de notre réseau.

Finalement nous donnons l'architecture de nos modèles avec plus de détails.

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_104 (Conv2D)          (None, 30, 30, 32)        320
_________________________________________________________________
conv2d_105 (Conv2D)          (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_54 (MaxPooling (None, 14, 14, 64)        0
_________________________________________________________________
batch_normalization_74 (Batc (None, 14, 14, 64)        256
_________________________________________________________________
conv2d_106 (Conv2D)          (None, 12, 12, 64)        36928
_________________________________________________________________
conv2d_107 (Conv2D)          (None, 10, 10, 128)       73856
_________________________________________________________________
max_pooling2d_55 (MaxPooling (None, 5, 5, 128)         0
_________________________________________________________________
batch_normalization_75 (Batc (None, 5, 5, 128)         512
_________________________________________________________________
flatten_21 (Flatten)         (None, 3200)              0
_________________________________________________________________
dense_42 (Dense)             (None, 512)               1638912
_________________________________________________________________
batch_normalization_76 (Batc (None, 512)               2048
_________________________________________________________________
dropout_21 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_43 (Dense)             (None, 43)                22059
=================================================================
Total params: 1,793,387
Trainable params: 1,791,979
Non-trainable params: 1,408
```

FIGURE 4.8 – Configuration de notre modèle.

4.4 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 : $(N \times M \times L + 1) \times K$.
$N \times M$: la taille de filtre, dans notre exemple pour la couche de $conv2D\_1$ est choisi à 3×3.
L : Les cartes de caractéristiques en entrée, on a une image donc $L = 1$.
K : Carte des caractéristiques en sortie (nombre de filtres) $K = 32$.
Donc le nombre de paramètres est équivalent à : $(3 \times 3 \times 1 + 1) \times 32 = 320$ paramètres.

Couche de convolution 2 **:** La taille du filtre est 3×3 et $K = 64$.
Nombre de paramètres : $(3 \times 3 \times 32 + 1) \times 64 = 18\ 496$ 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 $K = 64$.
Nombre de paramètres : $(3 \times 3 \times 64 + 1) \times 64 = 36\ 928$ paramètres.

Couche de convolution 4 : La taille du filtre est 3×3 et $K = 128$.
Nombre de paramètres : $(3 \times 3 \times 64 + 1) \times 128 = 73\ 856$ 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 : $(N + 1) \times M$.
Pour N entrées et M sorties :
Nombre de paramètres : $(3200 + 1) \times 512 = 1\ 638\ 912$ Paramètres.

Couche de sortie : la couche de sortie est une couche entièrement connectée, le nombre de paramètres est alors $(N + 1) \times M$:
Le nombre de paramètres de la couche de sortie s'élève à : $(512 + 1) \times 43 = 137\ 643$ paramètres.

Couche Batch-Normalization 1 : Pour calculer le nombre de paramètres dans la couche Batch-Normalization, on applique la loi suivante : $4 \times N$. Pour N entrées :
Nombre de paramètres : $4 \times 64 = 256$ Paramètres.

Couche Batch-Normalization 2 : Nombre de paramètres : $4 \times 128 = 512$ Paramètres.

Couche Batch-Normalization 3 : Nombre de paramètres : $4 \times 512 = 2\ 048$ Paramètres.

Le Nombre total de paramètres entrainés est : $320 + 18\ 496 + 36\ 928 + 73\ 856 + 1\ 638\ 912 + 137\ 643 + 256 + 512 + 2\ 048 = 1\ 793\ 387$ paramètres.

4.5 Résultats obtenus et discussion

4.5.1 Augmentation des données

La figure (4.9) présente les résultats de classification de GTSR sans augmentation des données et (4.10) avec augmentation.

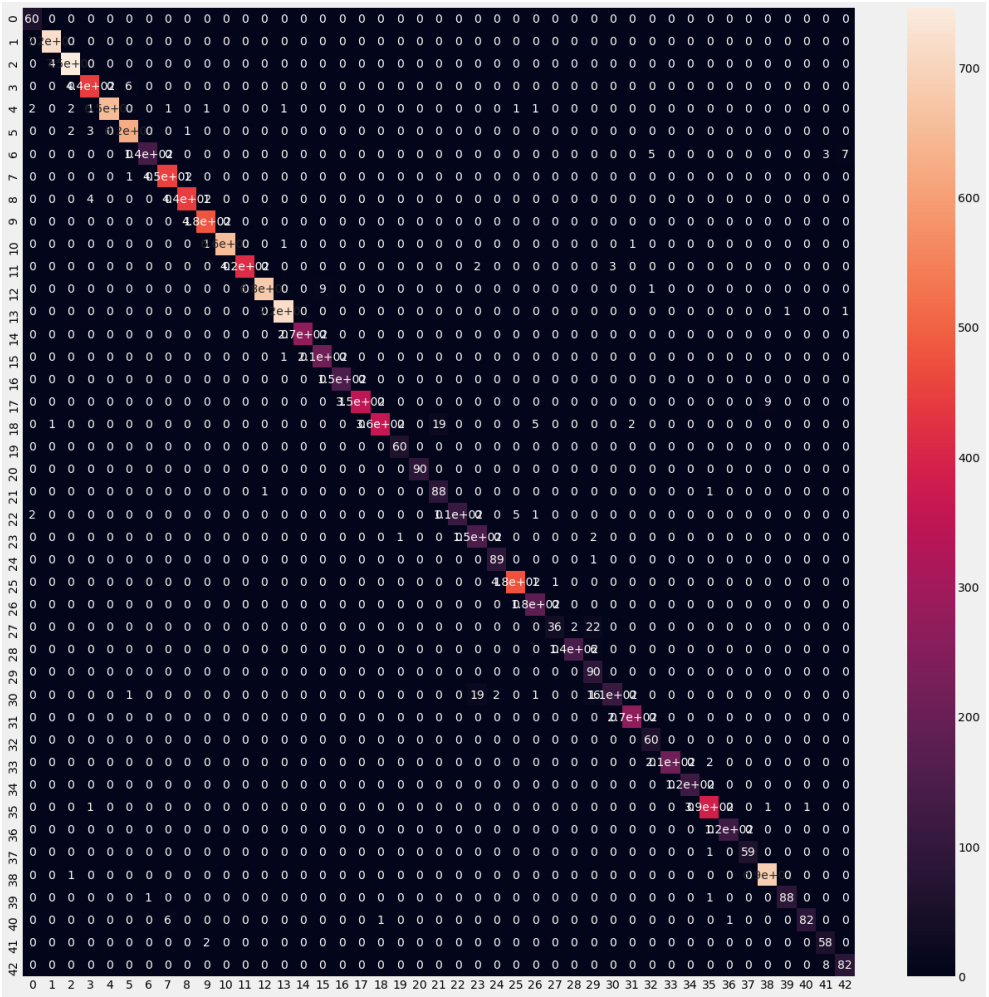


FIGURE 4.9 – Matrice de confusion de GTSR sans augmentation.

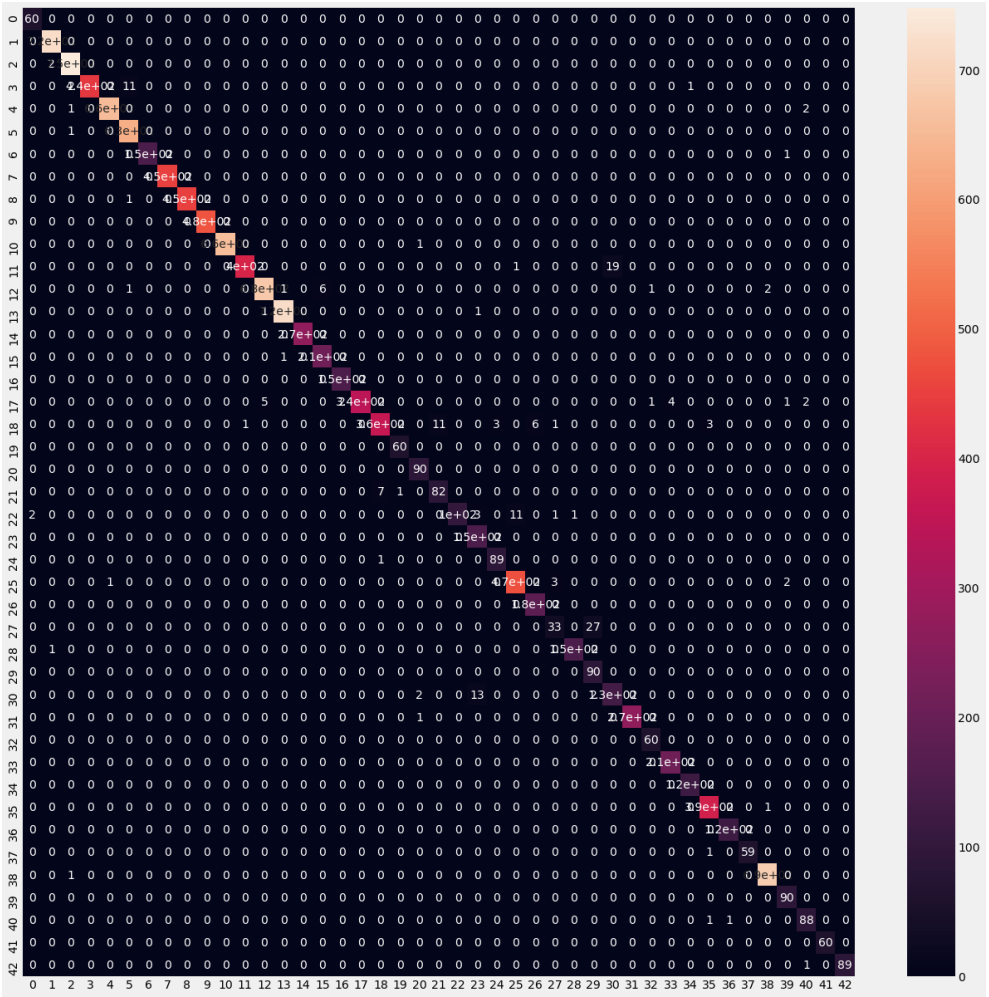
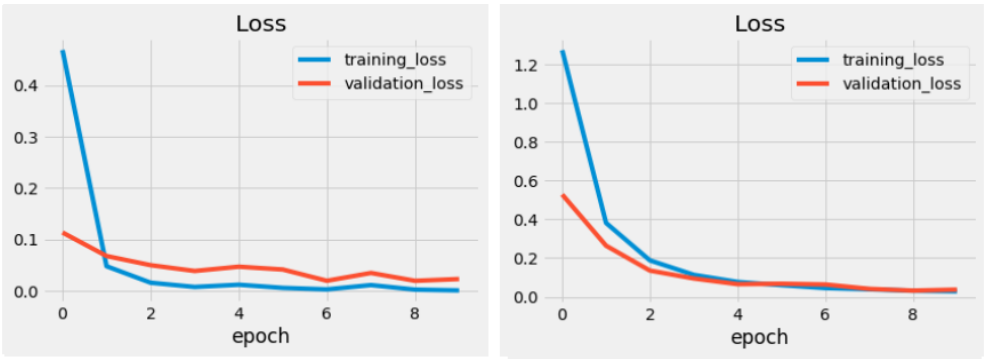


FIGURE 4.10 – Matrice de confusion de GTSR avec augmentation.

Nous remarquons que l'utilisation de la technique d'augmentation améliore la performance du modèle avec un cout très petit (le temps d'exécution s'augmente), sans perte de ressources ou augmenter les paramètres du réseau.

Les figures (4.11) et (4.12) présentent une comparaison d'erreur et de précision en fonction des épochs par rapport a l'utilisation ou non de la technique d'augmentation



(a) Sans augmentation (b) Avec augmentation

FIGURE 4.11 – Erreur pour le Modèle implémenté.

(a) Sans augmentation (b) Avec augmentation

FIGURE 4.12 – Précision pour le Modèle implémenté.

TABLE 4.1 – Tester le Modèle implémenté.

	Test accuracy	Test loss
Sans augmentation	0.9604	0.1463
Avec augmentation	0.9819	0.0633

D'après la comparaison des résultats trouvés montrent que l'augmentation des données a un grand influence sur les performances de classifieur.

4.5.2 Nombre d'époques

Les résultats obtenus en terme de précision et d'erreur de notre modèle, sont illustrés sur la Figure (4.13).

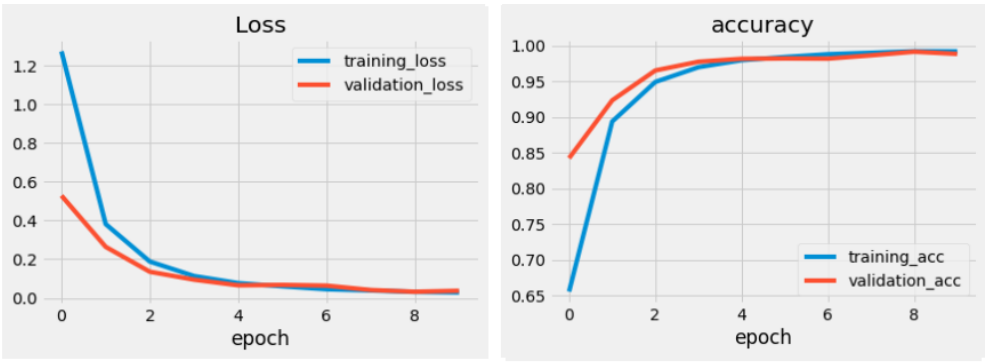


FIGURE 4.13 – Précision et Erreur pour le Modèle implémenté.

D'après la Figure (4.13) 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 que notre modèle apprenne et par conséquent on doit donc 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 0.9918, une perte d'entraînement de 0.0285, une perte de validation de 0.0317 et une précision de validation de 0.9883.
La durée totale d'apprentissage pour 10 époques est de 1310 secondes (CPU) presque 21.83 min, chaque époque prend environ 2.18 min.

```
Epoch 1/10
1103/1103 [==============================] - 128s 116ms/step - loss: 1.2731 - accuracy: 0.6557 - val_loss: 0.5284 - val_accuracy: 0.8424
Epoch 2/10
1103/1103 [==============================] - 129s 117ms/step - loss: 0.3816 - accuracy: 0.8936 - val_loss: 0.2641 - val_accuracy: 0.9232
Epoch 3/10
1103/1103 [==============================] - 129s 117ms/step - loss: 0.1880 - accuracy: 0.9490 - val_loss: 0.1354 - val_accuracy: 0.9651
Epoch 4/10
1103/1103 [==============================] - 130s 118ms/step - loss: 0.1134 - accuracy: 0.9696 - val_loss: 0.0946 - val_accuracy: 0.9773
Epoch 5/10
1103/1103 [==============================] - 133s 121ms/step - loss: 0.0768 - accuracy: 0.9794 - val_loss: 0.0654 - val_accuracy: 0.9814
Epoch 6/10
1103/1103 [==============================] - 130s 118ms/step - loss: 0.0601 - accuracy: 0.9839 - val_loss: 0.0674 - val_accuracy: 0.9819
Epoch 7/10
1103/1103 [==============================] - 129s 117ms/step - loss: 0.0450 - accuracy: 0.9878 - val_loss: 0.0637 - val_accuracy: 0.9816
Epoch 8/10
1103/1103 [==============================] - 131s 119ms/step - loss: 0.0388 - accuracy: 0.9898 - val_loss: 0.0410 - val_accuracy: 0.9862
Epoch 9/10
1103/1103 [==============================] - 130s 118ms/step - loss: 0.0315 - accuracy: 0.9920 - val_loss: 0.0321 - val_accuracy: 0.9913
Epoch 10/10
1103/1103 [==============================] - 130s 118ms/step - loss: 0.0285 - accuracy: 0.9918 - val_loss: 0.0371 - val_accuracy: 0.9883
```

FIGURE 4.14 – Détails sur l'apprentissage de notre modèle sur CPU.

Remarque : On a débuté l'implémentation avec 30 époques comme montré sur la Figure (4.15), mais d'après (4.15) (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 0.9918)

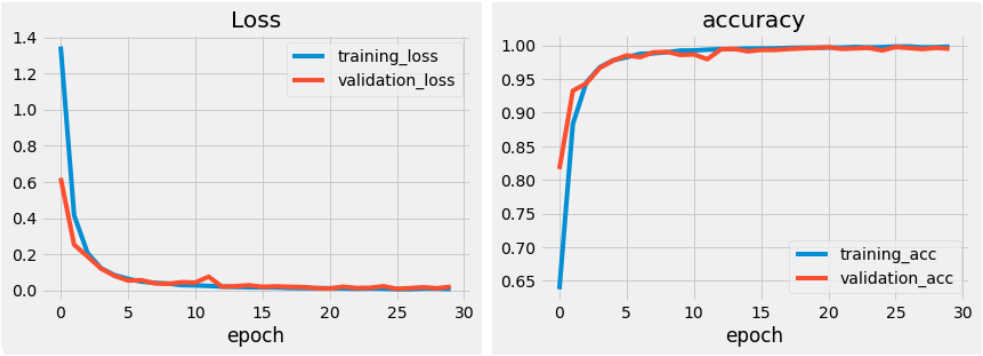


FIGURE 4.15 – Précision et Erreur pour le Modèle implémenté 30 époques.

Nous remarquons une bonne précision d'entrainement de 0.9979, une perte d'entraînement de 0.0083, une perte de validation de 0.0110 et une précision de validation de 0.9969.
La durée totale d'apprentissage pour 30 époques est de 3930 secondes (CPU) soit 65.5 min, chaque époque prend environ 131 secondes.
Les résultats obtenus montrent également que le nombre d'époques sont des facteurs importants pour l'obtention de meilleurs résultats.

4.5.3 Implémentation sur un GPU

La Fiqure 4.16 présente Détails sur l'apprentissage de notre modèle sur GPU .

```
Epoch 1/10
1103/1103 [==============================] - 16s 14ms/step - loss: 1.3156 - accuracy: 0.6410 - val_loss: 0.6637 - val_accuracy: 0.7995
Epoch 2/10
1103/1103 [==============================] - 16s 15ms/step - loss: 0.4200 - accuracy: 0.8804 - val_loss: 0.3361 - val_accuracy: 0.9056
Epoch 3/10
1103/1103 [==============================] - 15s 14ms/step - loss: 0.2124 - accuracy: 0.9415 - val_loss: 0.2030 - val_accuracy: 0.9403
Epoch 4/10
1103/1103 [==============================] - 16s 15ms/step - loss: 0.1281 - accuracy: 0.9649 - val_loss: 0.1081 - val_accuracy: 0.9676
Epoch 5/10
1103/1103 [==============================] - 15s 14ms/step - loss: 0.0879 - accuracy: 0.9765 - val_loss: 0.0955 - val_accuracy: 0.9755
Epoch 6/10
1103/1103 [==============================] - 16s 15ms/step - loss: 0.0645 - accuracy: 0.9826 - val_loss: 0.0591 - val_accuracy: 0.9829
Epoch 7/10
1103/1103 [==============================] - 16s 14ms/step - loss: 0.0341 - accuracy: 0.9910 - val_loss: 0.0357 - val_accuracy: 0.9903
Epoch 8/10
1103/1103 [==============================] - 15s 14ms/step - loss: 0.0247 - accuracy: 0.9921 - val_loss: 0.0294 - val_accuracy: 0.9921
Epoch 9/10
1103/1103 [==============================] - 15s 14ms/step - loss: 0.0182 - accuracy: 0.9946 - val_loss: 0.0222 - val_accuracy: 0.9952
Epoch 10/10
1103/1103 [==============================] - 15s 14ms/step - loss: 0.0150 - accuracy: 0.9958 - val_loss: 0.0150 - val_accuracy: 0.9949
```

FIGURE 4.16 – Détails sur l'apprentissage de notre modèle sur GPU .

Nous remarquons une bonne précision d'entrainement de 0.9958, une perte d'entraînement de 0.0150, une perte de validation de 0.0150 et une précision de validation de 0.9949. La durée totale d'apprentissage pour 10 époques est de 180 secondes (GPU) soit 3 min, chaque époque prend environ 18 secondes.

La Fiqure (4.17) présente Précision et Erreur pour le Modèle implémenté (GPU).

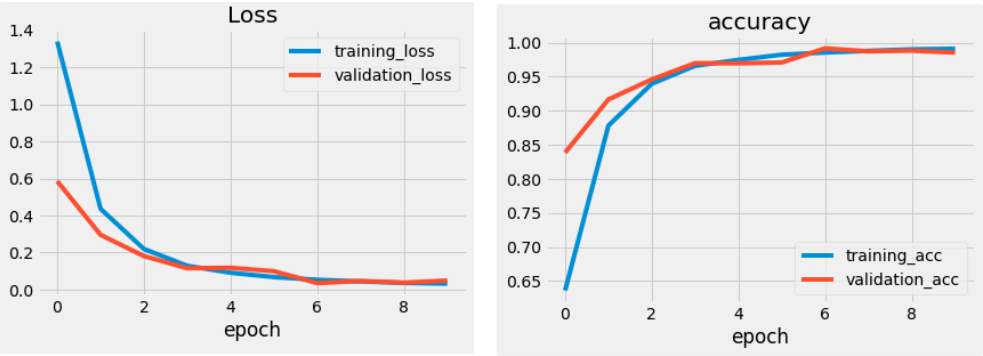


FIGURE 4.17 – Précision et Erreur pour le Modèle implémenté (GPU).

	Architecture Utilisé			Nombre Epoques	La base d'apprentissage		La base de validation		Temps d'exécution
	Couches de Convolution	Couches de Pooling	Fully Connected		Précision	Erreur	Précision	Erreur	
CPU	04	02	02	10	99,18%	02,85%	98,83%	03,71%	1310 secondes
GPU	04	02	02	10	99,58%	01,50%	99,49%	01,50%	180 secondes

FIGURE 4.18 – Résultats sur GPU et CPU.

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 préci-

sion 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.

4.5.4 Tester le Model

La fiqure (4.19) montre les résultats de la classification de notre système.

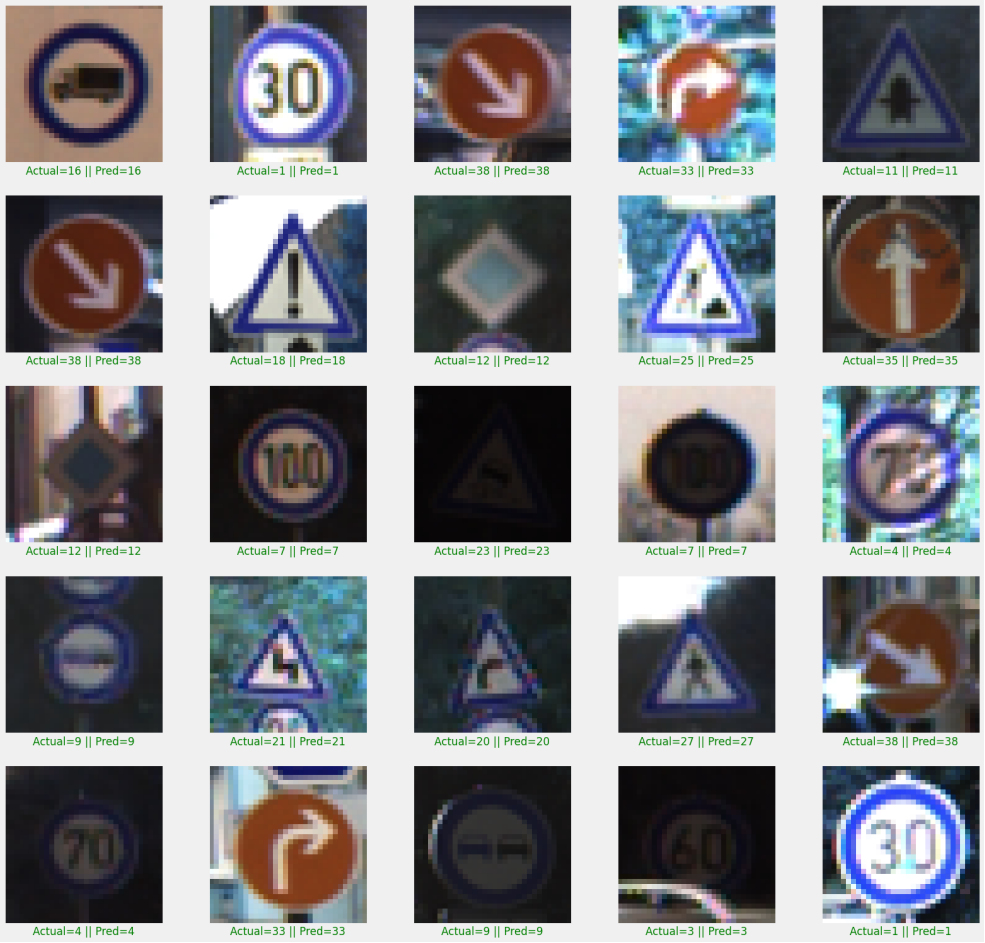


FIGURE 4.19 – Prédictions sur Test Data

Conclusion

Dans ce chapitre, nous introduisons une architecture de classification basée sur des réseaux de neurones convolutionnels, similaire à 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 l'augmentation des données, 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

Les systèmes de sécurité des véhicules modernes disposent de nombreuses technologies qui peuvent vous aider à conduire et à avertir 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 un système 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 concepts de base des réseaux de neurones en général et des réseaux de neurones convolutionnels en particulier. Nous avons introduit ces réseaux de neurones convolutionnels en présentant les divers 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 distribuées dans 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, il est souhaitable de déployer un réseau de neurones convolutionnels plus profonds sur des bases plus importantes sur un GPU est souhaitable.
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, afin d'aider les routiers et les transporteurs.
Au cours de ce projet, nous avons eu l'opportunité de se rapprocher du monde de l'intelligence artificielle et découvrir l'importance de ce domaine actuellement.

Bibliographie

[1] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2921-2929, Las Vegas, NV, USA, 2016.

[2] I. Vasilev, D. Slater, G. Spacagna, P. Roelants, V. Zocca. Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow. 2019.

[3] J. Ker, L. Wang, J. Rao, T. Lim. Deep Learning Applications in Medical Image Analysis. IEEE Access. DOI : 10.1109/ACCESS.2017.2788044. 2017.

[4] N. Buduma. Fundamentals of Deep Learning Designing Next-Generation Machine Intelligence Algorithms. 2017.

[5] S. Ioffe, C. Szegedy, "Batch normalization : accelerating deep network training by reducing internal covariate shift," in Proceedings of the International Conference on Machine Learning (ICML), pp. 448-456, Lille, France, 2015.

[6] Y. 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.

[7] Q. Zheng, M. Yang, X. Tian, N. Jiang, D. Wang. A Full Stage Data Augmentation Method in Deep Convolutional Neural Network for Natural Image Classification, ID. 4706576, 2020.

[8] Y. Moualek Djaloul. Deep learning pour la classification des images. Université Abou Bekr Belkaid- Tlemcen, Juillet 2017.

Web Bibliographie

1. **Python**
   https://python.org/
2. **Tensorflow**
   https://Tensorflow.org/
3. **Keras**
   https://keras.io/
4. **Kaggle**
   https://kaggle.com/
5. **Matplotlib**
   https://matplotlib.org

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.**

— $train\_path$ = '../input/myData/Train'

— $test\_path$ = '../input/myData/'

— étiquettes = '../input/myData/étiquette.csv'

— $Num\_class = len(os.listdir(train\_path))$

— $Epochs\_val = 10$: **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.**

— $Img\_h,\ Img\_w,\ channels = 32,\ 32,\ 3$: **image d'entrée avec dimension** $32 \times 32 \times 3$ **(Taille** 32×32 **avec** 3 **channel de RGB) 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 :

— $path = train\_path + '/' + str(i)\ images = os.listdir(path)$ **pour tout** i **dans** Num_class

— $image = cv2.imread(path + '/' + img)\ image\_data.append(image)$ $image\_labels.append(i)$: **pour tout** img **dans** $images$

— $image\_data = np.array(image\_data)\ image\_labels = np.array($ $image\_labels)$: **changer la liste en tableau** $numpy$

C)Diviser data :

— $X\_train,\ X\_test,\ y\_train,\ y\_test = train\_test\_split(image\_data,$ $Num\_class,\ test\_size = testRatio)$: **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.**

D)Lire un dossier csv :

— $data = pd.read\_csv$(étiquettes)
$classes = data["SignName"]$: **lire des étiquettes** 0 **jusqu'à** 42**.**

E)Pré-traitement 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)Encoder les étiquettes en One hot encoding

— $y\_train = keras.utils.to\_categorical(y\_train,\ Num\_class)$

— $y\_val = keras.utils.to\_categorical(y\_val,\ Num\_class)$

— $y\_test = keras.utils.to\_categorical(y\_test,\ Num\_class)$

G) L'augmentation des données :

— $aug = ImageDataGenerator(width\_shift\_range = 0.1,$
$height\_shift\_range = 0.1,\ zoom\_range = 0.2,\ shear\_range = 0.1,$
$rotation\_range = 10)$: **rotation des images, déplacement de gauche à droite, agrandissement des images, de sorte qu'il crée un ensemble de données différentes.**

H) Création de notre modèle de neurone :

— $Num\_filtre1 = 32$: **nombre de filtre utilisé dans la couche de convolution** 1 **et** 2**.**

— $Num\_filtre2 = 64$: **nombre de filtre utilisé dans la couche de convolution** 2 **et** 3**.**

— $Num\_filtre3 = 128$: **nombre de filtre utilisé dans la couche de convolution** 4**.**

— $Size\_filtre = (3, 3)$: **Taille des filtres.**

— $Size\_pool = (2, 2)$: **Taille de pool.**

— $Num\_nds = 512$: **Couche cachée avec une taille de** 512 **neurones.**

— Couche de convolution 1 : $model.add((Conv2D(nombre\_de\_filtre1,$
$Size\_filtre,\ input\_shape = (Img\_h,\ Img\_w, 1),\ activation =$ 'relu')))
: **Cette commande permet de créer** 32 **cartes de caractéristiques en**

utilisant un filtre de taille 3 **par** 3 **pixels, appliqué sur une image d'entrée de taille** $32 \times 32 \times 3$ **et une fonction d'activation de type RELU.**

— Couche de convolution 2 : $model.add((Conv2D(Num\_filtre2, Size\_filter, activation = \text{'relu'})))$: **Cette commande permet de créer** 64 **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 1 : $model.add(MaxPooling2D(pool\_size = Size\_pool))$: **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 Batch-Normalization 1 : $model.add.BatchNormalization(axis = -1)$: **Cette commande a but de normaliser les valeur et réduire le temps d'exécution**

— Couche de convolution 3 : $model.add((Conv2D(Num\_filtre2, Size\_filter, activation = \text{'relu'})))$: **Cette commande permet de créer** 64 **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 : $model.add((Conv2D(Num\_filtre3, Size\_filter, activation = \text{'relu'})))$: **Cette commande permet de créer** 128 **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 : $model.add(MaxPooling2D(pool\_size = Size\_pool))$: **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 Batch-Normalization 2 : $model.add.BatchNormalization(axis = -1)$

— $model.add(Flatten())$: **Cette commande permet de créer un seul vecteur 1D puis connecter avec la première couche cachée pour commencer la classification.**

— $model.add(Dense(Num\_dnds, activation = \text{'relu'}))$: **Cette commande permet de créer une couche cachée avec une taille de** 512 **neurones, la fonction RELU est utilisée comme fonction d'activation.**

— Couche Batch-Normalization 3 : $model.add.BatchNormalization()$

— Couche Dropout : $model.add(Dropout(0.5))$: **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.

— $model.add(Dense(Num\_class,\ activation =$ '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.

I) L'entrainement du modèle :

— $model.fit(aug.flow(X\_train,\ y\_train,\ batch\_size = 32),\ epochs = Epochs\_val,\ validation\_data = aug.flow(X\_val,\ y\_val,\ batch\_size = 8))$: **Cette commande permet de lancer l'apprentissage puis la validation.**

J) L'évaluation du modèle :

— $score = model.evaluate(x\_test,\ y\_test,\ verbose = 0)$ print('Test loss :', score[0]) print('Test accuracy :', score[1]) : **Cette commande permet d'évaluer notre modèle sur la base de test.**