FACULTÉ DES SCIENCES INFORMATIQUES
Calcul Scientifique

Minimisation d'erreurs cas d'invariance des transformations géométriques dans la reconnaissance des plaques d'immatriculation

---

Travail de fin d'études présenté et défendu en vue de l'obtention du grade de licencié en Calcul Scientifique.

Auteur : **TSHELEKA KAJILA Hassan**
Directeur : **Prof. MASAKUNA Jordan**

12 juillet 2022

RÉSUMÉ

TABLE DES MATIÈRES

LISTE DES ACRONYMES

IA Intelligence Artificielle

AI Artificial Intelligence

SA Supervised Learning

ML Machine Learning

CV Computer Vision

OCR Optical Character Recognition

ANPR Automatic Number-Plate Recognition

ALPR Automatic License Plate Recognition

GD Gradient Descent

SGD Stochastic Gradient Descent

CNN Convolutional Neural Network

ANN Artificial Neural Network

VGG Visual Geometry Group

ILSVRC ImageNet Large Scale Visual Recognition Challenge

NAG Nesterov Accelerated Gradient

Adam Adaptive Moment Estimation

Nadam Nesterov-accelerated Adaptive Moment Estimation

ReLu Rectified Linear Unit

MSE Mean Squared Error

MLP MultiLayer perceptron

SLNN Single-Layer Neural Network

$\mathbb{N}$	Ensemble des entiers naturels
$\mathbb{R}^n$	Ensemble des réels ou Espace euclidien de dimension n
$\mathbb{B}^n = \{0,1\}^n$	Espace booléen de dimension n
$\mathcal{O}(\cdot)$ ou $\Omega(\cdot)$	L'ordre de grandeur maximal de complexité d'un algorithme
$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$	Un vecteur
$x = (x_1, \vdots, x_n)^T$	Un vecteur
$x^T = (x_1, \vdots, x_n)^T$	Un vecteur transposé
$\langle xy \rangle = x^T y$	Le produit vectoriel
$\parallel x \parallel$	La norme du vecteur
M^{-1}	La matrice inverse d'une matrice M
M^T	La matrice transposée
$\frac{\partial}{\partial x} f(x,y)$	La dérivée partielle par rapport à x de al fonction f des deux variable x et y
$\nabla_A J(A,B)$	Le vecteur dérivé par rapport au vecteur A de la fonctionnelle J des deux vecteurs A et B

Les éléments en apprentissage

$\mathcal{S}$	L'échantillon d'apprentissage (un ensemble ou une suite d'exemple)
$\hat{y}$	La valeurs prédite après l'entrainement d'un modèle d'apprentissage automatique
$\hat{y}_i \in \mathcal{Y}$	La prédiction, ou sortie désirée, d'un exemple
$\mathcal{H}$	Espace des hypothèses d'apprentissages
$h \in \mathcal{H}$	Une hypothèses produite par un algorithme d'apprentissage
$y = h(x) \in \mathcal{Y}$	La prédiction faite par l'hypothèse h sur la description s d'un exemple
$\ell(f(x), h(x))$	La fonction perte ou fonction coût entre la fonction cible et une hypothèse sur x d'un exemple
$\mathcal{C}$	L'ensemble des classes
C	Le nombre de classes
$c_i \in \mathcal{C}$	Une sous classe de $\mathcal{C}$

0 INTRODUCTION

0.1 APERÇU GÉNÉRALE

L'application de cette étude dans l'apprentissage supervisé est orientée vers la reconnaissance automatique des plaques d'immatriculation (en anglais : automatic license plate recognition, ALPR) dans les images. Une des applications intéressantes parmi tant d'autres dans l'intelligence artificielle. Nous présentons une étude approfondie sur les algorithmes de minimisation de la fonction coût (en anglais : loss function) d'un modèle d'apprentissage pour l'ALPR.

Lorsque nous voulons faire une application dans le traitement de reconnaissance des formes dans des images, les ensembles de données d'entraînement pour les problèmes de détection d'objets sont généralement très volumineux et les capacités des méthodes d'apprentissage automatique statistique sont limitées par le temps de calcul plutôt que par la taille de l'échantillon [7]. Par exemple, pour entraîner une machine à reconnaître des plaques d'immatriculation de voiture, elle doit recevoir de grandes quantités d'images de plaques d'immatriculation et d'éléments liés aux plaques pour apprendre les différences et reconnaître une plaque, en particulier la voiture qui porte une plaque sans défaut. Plus nous avons des données, plus nous gagnons en précision et plus la complexité en temps augmente.
Des contraintes d'exploitation découlent des observations citées ci-dessus, parmi lesquelles nous citerons celles qui sont liées à la reconnaissance des objets dans les vidéos et images. Par exemple, de nos jours, un très grand nombre de caméras est déployé exclusivement pour la surveillance vidéo. [1]. Souvent, le contenu de ces vidéos est interprété par des opérateurs humains qui engendrent des coûts exorbitants pour le suivi et l'analyse du contenu, sans mentionner les erreurs qui peuvent être induites par la fatigue et l'inattention humaine.

0.2 CHOIX ET INTÉRÊT DU SUJET

0.3 ÉTAT DE L'ART

0.4 PROBLÉMATIQUE

La complexité de calcul de l'algorithme d'apprentissage devient le facteur limitant critique lorsque l'on envisage un grand ensemble de données, un ensemble d'image par exemple. C'est à ce point critique qu'entre en jeu cette étude, la minimisation des erreurs sans alourdir **la complexité en temps et espace** de l'algorithme d'apprentissage. Les ensembles de données d'entraînement pour les problèmes de détection d'objets dans des images sont généralement très volumineux. Minimiser les erreurs dans ces modèles

d'apprentissage est une tâche très importante pour renforcer la fiabilité de notre **modèle entraîné** [19].

Les modèles entraînés doivent être invariants sous des transformations géométriques et qualités des objets observés.

- Invariance à la rotation : l'objet dans une image doit être reconnue même après une rotation, sous différent angle.
- Invariance à l'échelle : le même objet dans une image doit être reconnue même sous une échelle différente.

Établir un algorithme d'apprentissage qui s'adapte au mieux à notre modèle, pour en reproduire un modèle entraîné et invariant à la transformation géométrique de l'objet dans l'image. Selon la nature du problème métier traité, il existe différentes approches qui varient selon le type de modèle à entraîné et la quantité des paramètres du modèle en question.

L'un des piliers de l'apprentissage automatique est l'optimisation mathématique [9] qui, dans ce contexte, implique le calcul numérique de minimisation des paramètres d'un système conçu pour prendre des décisions en fonction des données disponibles. Ces paramètres sont choisis pour être optimaux par rapport à notre problème d'apprentissage.

Dans l'ensemble, ce document tente d'apporter des réponses aux questions suivantes.

1. Comment les problèmes de minimisation surviennent-ils dans les applications d'apprentissage automatique ?
2. Quelles ont été les méthodes de minimisation les plus efficaces pour la reconnaissance automatique de plaque d'immatriculation ?
3. Quel sera le comportement du modèle entraîné par rapport à la transformation géométrique des objets ?
4. Comment des algorithmes d'apprentissage supervisé arrivent-t-ils résoudre le problème de l'invariance des transformations géométriques des objets ?

0.5 HYPOTHÈSE

Le cas des problèmes d'apprentissage à grande ou à petite échelle implique la complexité de calcul de l'algorithme d'optimisation sous-jacent de manière non triviale.
En effet, dans ce travail, nous analysons les algorithmes de descente de gradient stochastique parce qu'ils montrent des performances d'optimisation incroyables pour les problèmes à grande échelle. [7].

Le travail de Léon Bottou et al (e. g., dans [7] [31] [8]), présente *la descente de gradient stochastique comme un algorithme d'apprentissage fondamental*.
Une analyse plus précise révèle des compromis qualitativement différents pour le cas des problèmes d'apprentissage à grande échelle [9]. Des algorithmes d'optimisation improbables, tels que la SDG, montrent des performances étonnantes pour les problèmes à grande échelle, lorsque l'ensemble d'apprentissage est volumineux. En particulier, le gradient stochastique du second ordre et le gradient stochastique moyennée sont asymptotiquement efficaces après un seul passage sur l' ensemble d'entraînement [7].

Les optimiseurs axés sur la SGD n'utilisent qu'un seul nouvel échantillon d'apprentissage à chaque itération. De plus, ces optimiseurs sont utilisés pour faire une rétropropagation dans un réseau des neurones pour alléger les poids et biais de celui-ci.

0.6 OBJECTIFS

Nous nous proposons dans ce mémoire d'aborder sur l'utilisation des algorithmes d'optimisation numérique, précisément de minimisation. Ils seront appliqués à l'apprentissage automatique qui permettra aux ordinateurs et aux systèmes informatiques de dériver des informations significatives à partir d'images numériques, avec un coût plus bas que possible.

En fait, nous faisons la reconnaissance des plaques d'immatriculation des véhicules à l'aide d'un classificateur et optimiseurs de la famille de descente de gradient stochastique implémentés dans un réseau de neurones convolutifs (CNN). On s'en servira pour mesurer l'efficacité des optimiseurs par rapport à notre contrainte d'invariance à la transformations géométriques.
Pour minimiser la fonction de coût du classificateur, les optimiseurs SGD adoptent un modèle d'optimisation convexe [13]. De plus, pour augmenter la vitesse de convergence du classificateur, la descente de gradient stochastique, à chaque étape, tire un échantillon aléatoire de l'ensemble des paramètres de la fonction objectif [9].

Pour chaque algorithme, nous examinons l'efficacité de l'invariance à la transformation géométrique et comparons le score (accuracy, loss) pour différents cas.

0.7 LIMITATION

0.8 DIVISION DU TRAVAIL

En dehors de cette introduction, la partie conclusive et l'annexe, ce mémoire est organisé en trois chapitres comme suit.

> ***Chapitre 1*** est consacré à quelques rappels des matières sur lesquels je me base pour constituer l'ensemble de ce travail. Nous traitons des considérations de méthodes numériques et mathématiques impliquées dans la résolution de problèmes de minimisation des erreurs d'apprentissage. Certaines discussions sur les modèles de régression linéaire convexe et de classification dans d'apprentissage supervisé. Nous discutons également du réseau neuronal convolutif le plus adapté pour analyser l'imagerie visuelle.
>
> ***Chapitre 2*** explore une méthodologie parmi tant d'autres, pour entraîner les modèles d'apprentissage automatique de façon optimale, qui nous permettra par la suite de faire une prédiction d'images pour reconnaissance automatique de plaque d'immatriculation.
> Pour la minimisation de la fonction coût nous utilisons des algorithmes comme ASGD, ADAM, ADADELTA, NAG. Puis faire une étude comparative de leurs performances.

Chapitre 3, Ici nous construirons des modèles à partir d'une base de données annotée pour l'apprentissage et pour les tests de reconnaissance de plaque d'immatriculation sur l'image. Les résultats concluants de cette étude pourront conduire à un déploiement de notre système dans les domaines comme celui de la surveillance vidéo de voitures dans une entrée de parking. Des métriques connues pour mesurer les erreurs et en déduire le score du classificateur seront utilisées pour évaluer la qualité de la reconnaissance automatique des plaques d'immatriculation (ALPR) par notre approche.

1

CONCEPTS DE BASE ET ÉTAT DE CONNAISSANCES

1.1 LES ÉLÉMENTS D'OPTIMISATION NUMÉRIQUE

1.1.1 *Convexité*

La notion de convexité est important lorsque nous voulons minimiser une fonction. Autrement dit, une fonction convexe fait référence à une fonction dont le graphique a la forme d'une tasse $\bigcup$ ou une forme incurvée (Comme l'illustre la figure 1).

De manière équivalente, une fonction est convexe si et seulement si la région (ou ensemble des points) au-dessus de son graphe est un ensemble convexe C [10].
Une fonction f d'un intervalle réel $I \in C$ (ensemble convexe) est dite fonction convexe lorsque, $\forall(x, y)$ de I tel que $(x, y) \in C^2$ et tout $\alpha \in [0, 1]$, [23] on a :

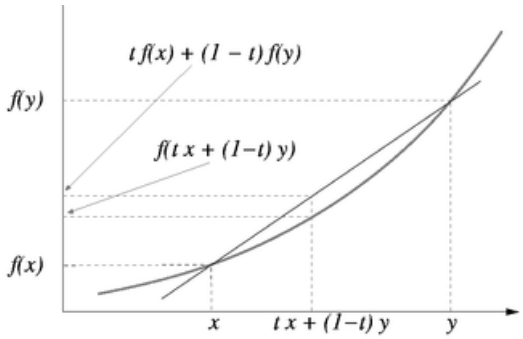


FIGURE 1 : Une fonction convexe, ici $\alpha = t$.

$$f(\alpha x + (1-\alpha)y) \leqslant \alpha f(x) + (1-\alpha)f(y)$$
$$\text{et si}$$
$$f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha)f(y)$$

on dit que la fonction est strictement convexe dans C, [23]. Une fonction deux fois différentiable d'une seule variable est convexe si et seulement si sa dérivée seconde est non négative sur tout son domaine [4].

EXTREMUM D'UNE FONCTION [10] : Soit $I \to \mathbb{R}$ une fonction et a un point de I ($a \in I$).

+ On dit que m est un **minimum local** de f, si pour tout $x \in I$, $f(x) \leqslant f(a)$ ou s'il existe $\alpha > 0$ tel que m soit le minimum de f restreinte à $I \cap]a - \alpha, a + \alpha[$.
+ On dit que M est un **maximum local** de f, si pour tout $x \in I$, $f(x) \geqslant f(a)$ ou s'il existe $\alpha > 0$ tel que M soit le maximum de f restreinte à $I \cap]a - \alpha, a + \alpha[$.

Donc nous pouvons dire qu'une fonction convexe à un unique point minimum. Les fonctions convexes sont, avec les ensembles convexes, jouent aussi un rôle singulier en optimisation, en supprimant la distinction entre minima locaux et globaux, tout minimum local d'une fonction convexe est un minimum global [10]. La fonction carré et la fonction exponentielle sont des exemples de fonctions strictement convexes sur l'ensemble réel $\mathbb{R}$.

Cette notion nous sera plus utile lorsque nous voudrions minimiser notre fonction coût.

1.1.2 *Gradient*

Minimiser une fonction consiste à trouver sa valeur minimum. La valeur minimum d'une fonction se trouve lorsque sa dérivée s'annule en un point de son ensemble de définition et change de signe passant de négatif à positif [10]. Pour les fonctions scalaires à variables vectorielles on utilise le gradient à la place du dérivé [23].
Le gradient est la généralisation à plusieurs variables de la dérivée d'une fonction d'une seule variable [4, 5].

Dans un système de coordonnées cartésiennes, le gradient d'une fonction $f(x_1, x_2, \ldots, x_n)$ est le vecteur de composantes $\partial f/\partial x_i$ $(i = 1, 2, \ldots, n)$, c'est-à-dire les dérivées partielles de f par rapport aux coordonnées [23].

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbb{R}^n \tag{1}$$

Une fonction multivariée, à variable vectorielle, $f(x) : \mathbb{R}^n \to \mathbb{R} : x \to f(x)$ définie sur ensemble un ouvert $O \in \mathbb{R}^n$ est dite dérivable [23] en x ssi il existe un vecteur noté $\nabla f(x) \in \mathbb{R}^n$ tel que

$$f(x + h) = f(x) + \nabla f(x)^T h + o(||h||) \tag{2}$$

avec $\nabla f(x) \in \mathbb{R}^n$ et où l'on a posé que le reste $o(||h||) = ||h||\epsilon(h) \in \mathbb{R}^n$, avec $h \in \mathbb{R}^n$

$$\epsilon(h) : \mathbb{R}^n \to \mathbb{R}, \qquad \lim_{||h|| \to 0} \epsilon(h) = 0$$

Le vecteur $\nabla f(x)$ est unique et nommé gradient de $f(x)$ en x. Le gradient s'adresse aux fonctions scalaires à variables vectorielles.

1.2 CONCEPTS DE LA MODÉLISATION ET CLASSIFICATION DES DONNÉES

1.2.1 *Concepts de la modélisation*

La modélisation est la conception et l'utilisation d'un *modèle*. Selon son objectif et les moyens utilisés, la modélisation est dite mathématique, géométrique, 3D, etc.
La modélisation permet de concevoir l'architecture globale d'un système d'information, ainsi que l'organisation des informations à l'aide de la modélisation des données [21].

Dans notre contexte nos modèle sont mathématique. Un **modèle mathématique** est une description d'un système utilisant des concepts et un langage mathématiques [12].
Ce modèle peut aider à expliquer un système et à étudier les effets de différents composants, et à faire des prédictions sur le comportement [18].

A *Modèles non paramétriques*

Un modèle non paramétrique est construit selon les informations provenant des données. Dans [3, 6] il est expliqué que : La régression non paramétrique exige des tailles d'échan-

tillons plus importantes que celles de la régression basée sur des modèles paramétriques parce que les données doivent fournir la structure du modèle ainsi que les estimations du modèle.

Un modèle paramétrique est, s'il est approximativement valide, plus puissant qu'un modèle non paramétrique, produisant des estimations d'une fonction de régression qui ont tendance à être plus précises que ce que nous donne l'approche non paramétrique [21]. Cela devrait également se traduire par une prédiction plus précise.

Dans [3], il est expliqué que nous pouvons construire un modèle d'apprentissage, ou l'espaces des hypothèses d'apprentissage, par :

- La classification
- La régression
- Les distributions de probabilités
- Les arbres de décisions
- Les réseaux bayésiens
- Etc.

B *Entraînement du modèle*

Tout modèle, où toutes les informations nécessaires ne sont pas disponibles, contient certains paramètres qui peuvent être utilisés pour adapter le modèle au système qu'il est censé décrire. Si la modélisation est effectuée par un réseau de neurones artificiels ou un autre apprentissage automatique, l'optimisation des paramètres est appelée **entraînement** (en anglais : **training**), tandis que l'optimisation des hyperparamètres du modèle est appelée **réglage** (en anglais : **tuning**) et utilise souvent la validation croisée [17]. Dans une modélisation plus conventionnelle à travers des fonctions mathématiques explicitement données, les paramètres sont souvent déterminés par ajustement de courbe (voir le chapitre **??**, section **??**).

Une partie cruciale du processus de modélisation consiste à évaluer si oui ou non un modèle mathématique donné décrit un système avec précision. Il peut être difficile de répondre à cette question car elle implique plusieurs types d'évaluation différents [17, 21].

1.2.2 *Les problèmes de régressions*

L'algorithme d'apprentissage automatique est défini comme un algorithme capable d'améliorer les performances d'un programme informatique à certaines tâches via l'expérience est quelque peu abstraite. Pour rendre cela plus concret, Une des méthode d'apprentissage automatique basique est *la régression linéaire* [17].

En statistique, la régression linéaire est une approche linéaire pour modéliser la relation entre une réponse scalaire (*cf.* section 1.2.1) et une ou plusieurs variables explicatives (également appelées variables dépendantes et indépendantes). Le cas d'une variable explicative est appelé régression linéaire simple ; pour plus d'un, le processus est appelé régression linéaire multiple [12].

Les problèmes de régressions ont de nombreuses utilisations pratiques. Si l'objectif est la prédiction, la prévision ou la réduction des erreurs, la régression peut être utilisée pour ajuster un modèle prédictif à un ensemble de données observées de valeurs de la réponse et de variables explicatives [12]. Après avoir développé un tel modèle, si des valeurs supplémentaires des variables explicatives sont collectées sans valeur de réponse d'accompagnement, le modèle ajusté peut être utilisé pour faire une prédiction de la réponse [18].

A *Le cas de la régression générale*

La plupart des modèles de régression proposent que Y_i est une fonction de X_i et w, avec ϵ_i représentant un terme d'erreur additif ou bruit statistique aléatoire qui peut remplacer des déterminants non modélisés de Y_i :

$$Y_i = f(X_i, w) + \epsilon_i \tag{3}$$

L'objectif est d'estimer la fonction $f(X_i, w)$ qui correspond le mieux aux données.
Pour effectuer une analyse de régression, la forme de la fonction f doit être spécifié. Parfois, la forme de cette fonction est basée sur l'information de la relation entre Y_i et X_i. Si ces informations ne sont pas disponibles, un cas souple ou pratique pour f est choisi. Par exemple, une simple régression univariée peut proposer

$$f(X_i, w) = w_0 + w_1 X_i$$

ou

$$Y_i = w_0 + w_1 X_i + e_i$$

être une approximation raisonnable du processus statistique générant les données.

Différentes formes d'analyse de régression fournissent des outils pour estimer les paramètres. w. Par exemple, les moindres carrés trouvent la valeur de w qui minimise la somme des carrés des erreurs [13].

$$\sum_{i}^{n} (Y_i - f(X_i, w))^2$$

Étant donné un ensemble de données $\{y_i, x_{i1}, \ldots, x_{ip}\}_{i=1}^{n}$ de n unités statistiques, un modèle de régression linéaire suppose que la relation entre la variable dépendante y et le vecteur w des régresseurs x est linéaire. Cette relation est **modélisée** par un terme de perturbation ou une variable d'erreur ϵ : une variable aléatoire non observée qui ajoute du "bruit" à la relation linéaire entre la variable dépendante et les régresseurs [3, 12]. Ainsi le modèle prend la forme

$$y_i = w_0 + w_1 x_{i1} + \cdots + w_n x_{in} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol{w} + \varepsilon_i, \qquad \textit{avec} \quad i = 1, \ldots, n,$$

Souvent, ces n équations sont empilées et écrites en notation matricielle comme

$$\mathbf{y} = X\boldsymbol{w} + \boldsymbol{\varepsilon}, \tag{4}$$

où

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{pmatrix}, \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

$\mathbf{y}$ est un vecteur de valeurs observées y_i $(i = 1, \ldots, n)$ de la variable appelée variable mesurée ou variable dépendante.

X peut être vu comme une matrice de vecteurs-lignes $\mathbf{x}_i$ ou de vecteurs-colonnes à n dimensions X_j, appelées régresseurs, variables explicatives, variables d'entrée, variables prédictives ou variables indépendantes. La matrice X est parfois appelée la matrice de conception.

$\boldsymbol{w}$ est un vecteur de paramètre de dimension $(p+1)$, où w_0 est le terme d'interception, s'il n'est pas inclus dans le modèle $\boldsymbol{w}$ est de dimension p. Ses éléments sont appelés coefficients de régression [3]. En régression linéaire simple, $p = 1$, et le coefficient est appelé **pente** de régression.

L'estimation statistique et l'inférence dans la régression linéaire se concentrent sur w. Les éléments de ce vecteur de paramètres sont interprétés comme les dérivées partielles de la variable dépendante par rapport aux différentes variables indépendantes [12].

En définissant les vecteurs et matrice ci dessous, $\mathbf{X}$, $\boldsymbol{w}$ et $\mathbf{y}$ (avec $S_y = y$) [3] ; le critère de la somme des carrées des erreurs s'écrit alors :

$$\mathrm{SCE}(w|S) = \frac{1}{2}(\mathbf{S}_y - \mathbf{X}\boldsymbol{w})^T(\mathbf{S}_y - \mathbf{X}\boldsymbol{w}) \tag{5}$$

Il suffit de prendre la dérivée de la somme des carrés des erreurs (équation **??**) par rapport à w, qui est maintenant remplacer par w, pour obtenir les équations :

$$\frac{\partial \mathrm{SCE}}{\partial w} = -\mathbf{X}^T(\mathbf{S}_y - \mathbf{X}\boldsymbol{w})$$

$$\frac{\partial^2 \mathrm{SCE}}{\partial^2 w \partial^2 w^T} = -\mathbf{X}^T\mathbf{X}$$

En supposant que la matrice X est non singulière, et donc que X^TX est positive définie, et en posant que la dérivée première est nulle, on obtient :

$$X^TXw = X^TS_y \tag{6}$$

à partir de quoi on peut calculer l'unique solution par :

$$\hat{w} = (X^TX)^{-1}X^TS_y \tag{7}$$

La valeur $\hat{y}$ prédite pour une entrée x_n est donc :

$$\hat{y} = \hat{w} \cdot x_n = (X^TX)^{-1}X^TS_yx_n$$

RÉGRESSION LINÉAIRE MULTIPLE La régression linéaire multiple est une généralisation de la régression linéaire simple au cas de plus d'une variable indépendante, et un cas particulier des modèles linéaires généraux, limités à une variable dépendante.

1.2.3 *Les problèmes de classifications*

En apprentissage automatique, les classifieurs linéaires sont une famille d'algorithmes de classement statistique. Le rôle d'un classifieur est de classer dans des groupes (des classes) les échantillons qui ont des propriétés similaires, mesurées sur des observations. Un classifieur linéaire est un type particulier de classifieur, qui calcule la décision par combinaison linéaire des échantillons [3].

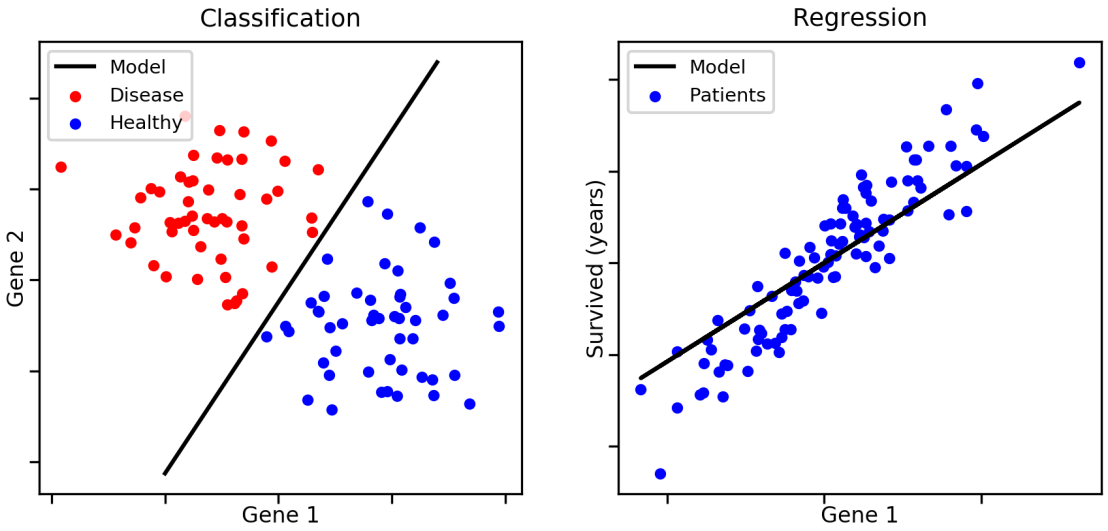


FIGURE 2 : Classification vs régression [27].

Nous nous plaçons dans le cadre où la variable dépendante ou à prédire prend ses valeurs dans un ensemble fini que l'on associe généralement à un ensemble de classes. A la différence de la régression linéaire où l'ensemble de valeurs à prédire est infini.

Lorsque l'on se place dans un espace de représentation euclidien, on peut librement faire des hypothèses sur la géométrie des classes ou sur celles de leurs surfaces séparatrices. La plus simple d'entre elles est de supposer que deux classes peuvent être séparées par une certaine surface, définie par une équation ; les paramètres qui régissent cette équation sont alors les variables à apprendre.

Le nombre de paramètres à calculer est minimal si l'on suppose cette surface linéaire ; aussi est-ce l'hypothèse qui prévaut souvent, en particulier lorsque l'échantillon de données est de taille réduite par rapport à la dimension de l'espace d'entrée, d'autant qu'elle permet de mener des calculs faciles et de visualiser précisément le résultat obtenu [25].

Dans $\mathbb{R}^n$, une surface linéaire est un hyperplan A, défini par l'équation :

$$a_0 + a^T x = 0$$

avec a vecteur de dimension n et a_0 scalaire. Si deux classes $\mathcal{C}_1$ et $\mathcal{C}_2$ sont *séparables* par A, tous les points de la première classe sont par exemple tels que :

$$x \in \mathcal{C}_1 \implies a_0 + a^T x > 0 \tag{8}$$

et ceux de la seconde vérifient alors :

$$x \in \mathcal{C}_2 \implies a_0 + a^T x \leqslant 0 \tag{9}$$

Dans un espace de dimension $d = 1$, une séparation linéaire se réduit à la comparaison à un seuil. Prenons ce cas particulier pour donner deux exemples où un problème de discrimination à deux classes ne peut pas en pratique être complètement résolu par une séparatrice linéaire.

SÉPARATRICE LINÉAIRE : On appelle hyperplan séparateur ou séparatrice linéaire un hyperplan qui sépare parfaitement deux classes, c'est-à-dire qui vérifie les équations 8 et 9 ; en particulier, il sépare parfaitement leurs points d'apprentissage. Un hyperplan discriminant est un classificateur linéaire pour deux classes qui ne sont pas linéairement séparables [3].

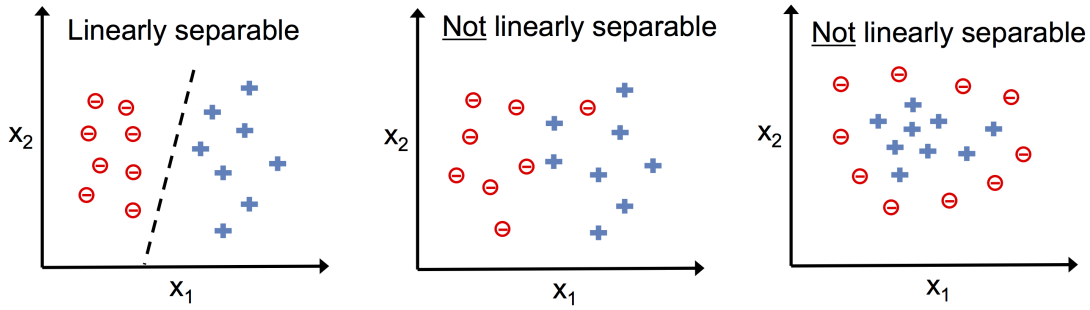


FIGURE 3 : Classes linéairement séparables [27]

A *Le modèle de la régression logistique*

Ce qu'il est convenu d'appeler *régression logistique* concerne en fait une méthode de classification binaire. A la différence du perceptron, cependant, nous allons chercher à apprendre une hypothèse h définie de $\mathbb{R}^n$ dans $[0,1]$, et non pas dans $0,1$, une motivation étant d'interpréter $h(x)$ comme étant la probabilité que l'entrée x appartienne à la classe d'intérêt que nous notons $\mathcal{C}_1$. [3]

La fonction de la droite séparatrice comme l'illustre la figure 4 s'écrit :

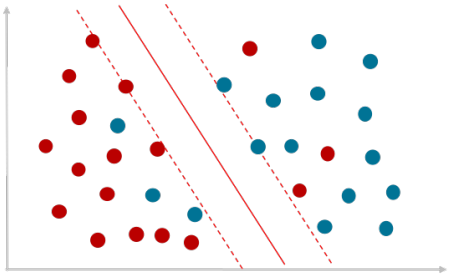


FIGURE 4 : Droite séparatrice.

$$z = w_1 x_1 + \cdots + w_n x_n + b \tag{10}$$

avec $i = 1, \ldots, n$, et w_i et b des paramètres de la droite.

$$\begin{cases} \hat{y} = 0 \ \ (y \in \mathcal{C}_1) & \text{si } z < 0 \\ \hat{y} = 1 \ \ (y \in \mathcal{C}_2) & \text{si } z \geqslant 0 \end{cases}$$

La fonction logistique est une fonction sigmoïde , qui prend n'importe quelle entrée réelle t, et renvoie une valeur comprise entre zéro et un [27]. La fonction logistique standard $\sigma : \mathbb{R} \to (0, 1)$ est défini comme suit :

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}} \tag{11}$$

Supposons que t est une fonction linéaire (comme la droite de la formule 10) $t = z$. Et la fonction logistique générale $p : \mathbb{R} \to (0, 1)$ peut maintenant l'écrire :

$$p(x) = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-(w_1 x_1 + \cdots + w_n x_n + b)}} \tag{12}$$

Dans le modèle logistique, $p(x)$ est interprété comme la probabilité de la variable dépendante Y équivalant à un succès/cas-oui plutôt qu'à un échec/non-cas. Il est clair que les variables de réponse Y_i ne sont pas identiquement répartis :$P(Y_i = 1 \mid X)$ diffère d'un point de données X_i à l'autre, bien qu'ils soient indépendants étant donné la matrice de conception X et paramètres partagés w [3].

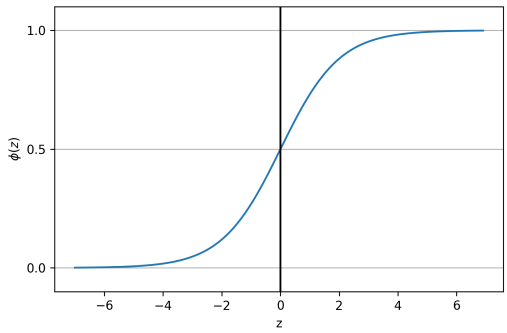


FIGURE 5 : Graphique représentant fonction sigmoïde logistique ajustée aux données (x_n, y_n) [27]

.

LA VRAISEMBLANCE : Indique la plausibilité du modèle vis-a-vis du vraies données. Soit l'échantillon $\mathcal{S} = (\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$, avec $y_i \in \{\mathcal{C}_1, \mathcal{C}_2\}, \forall_i \in (1, ..., m)$. Sa vrai semblance en fonction des paramètres à apprendre s'écrit :

$$L = \prod_{i=1}^{m} p_i{}^{y_i} (1 - p_i)^{1 - y_i} \tag{13}$$

où m est le nombre d'exemples d'apprentissage appartenant à la classe.

Dans [3], il est montré que ces paramètres peuvent être obtenus par maximisation de la vraisemblance des paramètres conditionnellement aux exemples. Maximiser une fonction $f(\cdot)$ consiste à minimiser $-f(\cdot)$ alors la log-vraisemblance s'écrira :

$$\mathcal{L} = -\frac{1}{m} \sum_{i=1}^{m} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \tag{14}$$

avec le terme $\frac{1}{m}$ pour augmenter la précision. Avec cette fonction, nous allons maximiser la vraisemblance L en minimisant $-\log(L)$.

La log-vraisemblance négative est égale à la perte logarithmique (Log-loss) sous une distribution de probabilité de Bernoulli [3, 12].

1.3 RÉSEAU DE NEURONES, APPRENTISSAGE EN PROFONDEUR

1.3.1 *Perceptron*

Le perceptron est un modèle simplifié d'un neurone biologique. Alors que la complexité des modèles de neurones biologiques est souvent nécessaire pour bien comprendre le comportement neuronal, la recherche suggère qu'un modèle linéaire de type perceptron peut produire certains comportements observés dans de vrais neurones.

Le perceptron, à l'instar du neurone artificiel classique, est conçu avec un algorithme d'apprentissage supervisé du même nom.
L'algorithme du perceptron proposé par Frank Rosenblatt [3], basé sur le modèle neuronal MP Neuron (McCulloch-Pitts Neuron), est un algorithme qui apprendrait automatiquement les coefficients de poids optimaux qui sont ensuite multipliés par les caractéristiques d'entrée afin de décider si un neurone se déclenche ou non. Dans le cadre de l'apprentissage supervisé et de la classification, un tel algorithme pourrait alors être utilisé pour prédire si un échantillon appartient à une classe ou à l'autre [27].

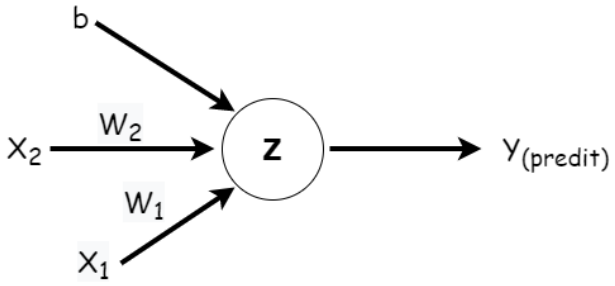


FIGURE 6 : Neurone logique avec deux entrées x_1 et x_2 et les paramètres w_1, w_2 (w_i dans un réseau de neurones il est nommé poids) et b aussi appelé biais. le fonction d'agrégation z (voir le point a, section 1.2.2 et la formule 10) sera : $z = w_1x_1 + w_2x_2 + b$.

Le perceptron a des entrées qui peuvent provenir de l'environnement ou peuvent être les sorties d'autres perceptrons. Associé à chaque entrée, $x_j \in \mathbb{R}$, avec $j = 1, 2, \ldots, n$, est un *poids de connexion, ou poids* synaptique $w_j \in \mathbb{R}$, et la sortie, $\hat{y}$. Dans le cas le plus simple $\hat{y}$ est une somme pondérée des entrées [2].

$$\hat{y} = \sum_{j=1}^{n} w_j x_j + w_0$$

w_0 est la valeur d'interception pour rendre le modèle plus général, il est généralement modélisé comme la pondération provenant d'une unité de biais supplémentaire, $b = w_0x_0$, avec x_0 qui est toujours égale $+1$. Nous pouvons écrire la sortie du perceptron sous la forme d'un produit scalaire.

$$\hat{y} = x^T w$$

Pendant le test, avec des poids donnés, w, pour l'entrée x, nous calculons la sortie $\hat{y}$. Pour implémenter une tâche donnée, nous avons besoin d'apprendre les poids w, les paramètres du système, de sorte que des sorties correctes soient générées compte tenu des entrées.

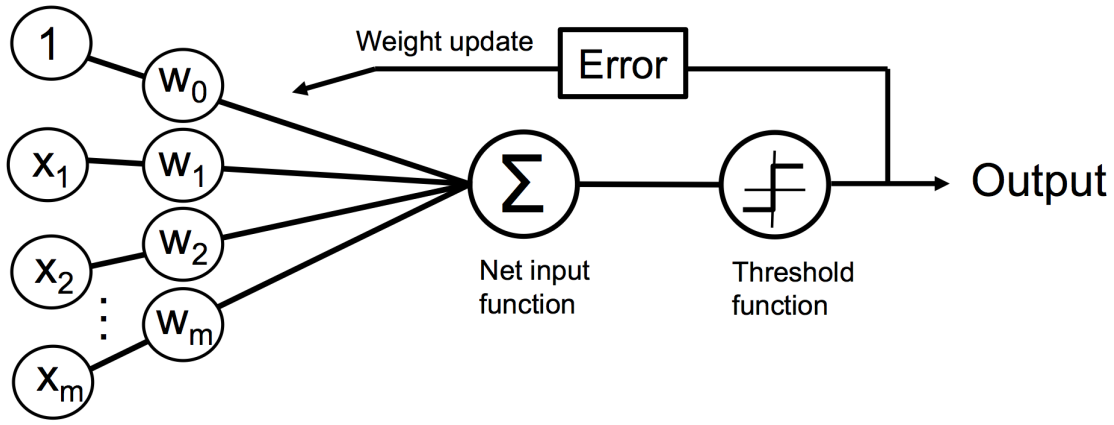


FIGURE 7 : Neurone artificiel modèle perceptron [27], cette figure est plus adapté pour représenter le modèle du perceptron.

A *Le réseau de neurones artificiels (Perceptron Multicouche)*

Les réseaux de neurones ont été introduits pour la première fois comme méthode d'apprentissage par Frank Rosenblatt, bien que le modèle d'apprentissage appelé perceptron soit différent des réseaux de neurones modernes, nous pouvons toujours considérer le perceptron comme le premier réseau de neurones artificiels [25].

Le perceptron multicouche (en anglais : multilayer perceptron MLP) est un type de réseau neuronal artificiel (ANN) organisé en plusieurs couches, où les informations ne circulent que de la couche d'entrée à la couche de sortie. Il s'agit donc d'un réseau à propagation directe (feed forward), autrement dit le réseau profond à action directe. Un perceptron multicouche est juste une fonction mathématique mappant un ensemble de valeurs d'entrée à des valeurs de sortie. La fonction est formée en composant de nombreuses fonctions plus simples. Nous pouvons considérer chaque application d'une fonction mathématique différente comme fournissant une nouvelle représentation de l'entrée [3, 17].

Les perceptrons à une seule couche ne sont capables d'apprendre que des motifs linéairement séparables. Pour une tâche de classification avec une **fonction d'activation** d'étape, un seul nœud aura une seule ligne divisant les points de données formant les motifs. Plus de nœuds peuvent créer plus de lignes de division, mais ces lignes doivent en quelque sorte être combinées pour former des classifications plus complexes. Une deuxième couche de perceptrons, voire de nœuds linéaires, suffit à résoudre de nombreux problèmes autrement non séparables [3].

Les réseaux de neurones artificiels fonctionnent vaguement sur le principe de l'apprentissage d'une distribution distribuée de données. Les paramètres du modèle ANN sont les poids de chaque connexion qui existent dans le réseau et parfois un paramètre de biais [25].

1.3.2 *Fonctions d'activation, poids et biais*

La fonction d'activation est responsable de la transformation de l'entrée pondérée sommée du nœud en activation du nœud ou de la sortie pour cette entrée. Pour un nœud donné, les entrées sont multipliées par les poids d'un nœud et additionnées. Cette valeur est appelée activation sommée du nœud. L'activation sommée est ensuite transformée via une fonction d'activation et définit la sortie spécifique ou « activation » du nœud [27]. La fonction sigmoïde (aussi appelé fonction logistique voir la section 1.2.3, le point a) est utilisée ici comme une fonction d'activation.

Les fonctions d'activation linéaires sont toujours utilisées dans la couche de sortie pour les réseaux qui prédisent une quantité (par exemple, les problèmes de régression, voir la section 1.2.2) [16, 20].

Les fonctions d'activation non linéaires sont préférées car elles permettent aux nœuds d'apprendre des structures plus complexes dans les données. Traditionnellement, deux fonctions d'activation non linéaires largement utilisées sont les fonctions d'activation tangente sigmoïde et hyperbolique [17].

A *Fonctions d'activation tangente (Sigmoïde et hyperbolique)*

La fonction d'**activation sigmoïde**, est traditionnellement une fonction d'activation très populaire pour les réseaux de neurones. L'entrée de la fonction est transformée en une valeur comprise entre 0,0 et 1,0. Les entrées qui sont beaucoup plus grandes que 1,0 sont transformées à la valeur 1,0, de même, les valeurs beaucoup plus petites que 0,0 sont alignées sur 0,0.

La forme de la fonction pour toutes les entrées possibles est une forme en S de zéro jusqu'à 0,5 à 1,0. Pendant longtemps, jusqu'au début des années 1990, c'était l'activation par défaut utilisée sur les réseaux de neurones [20].

Supposons que z est une fonction linéaire (comme la droite de la formule 10). Et la fonction sigmoïde (en partant de la fonction logistique générale, formule 12) est :

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(w_1x_1+\cdots+w_nx_n+b)}} \tag{15}$$

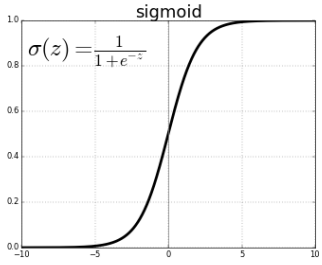


FIGURE 8 : Sigmoïde

avec $z = w_1x_1 + \cdots + w_nx_n + b$.

La fonction **tangente hyperbolique**, ou **tanh** en abrégé, est une fonction d'activation non linéaire de forme similaire qui génère des valeurs comprises entre -1,0 et 1,0. À la fin des années 1990 et au cours des années 2000, la fonction tanh a été préférée à la fonction d'activation sigmoïde car les modèles qui l'utilisaient étaient plus faciles à entraîner et avaient souvent de meilleures performances prédictives [17]. La fonction d'activation tangente hyperbolique fonctionne généralement mieux que la sigmoïde logistique.

B *Fonction d'activation ReLU*

Dans le domaine des réseaux de neurones artificiels, ReLU (Rectified Linear Unit) ou fonction d'activation d'unité linéaire rectifiée est une fonction d'activation définie comme la partie positive de son argument [17].

$$f(x) = x^+ = \max(0, x)$$

ReLU est une fonction linéaire par morceaux qui produira l'entrée directement si elle est positive, sinon, la sortie est nulle [30]. C'est devenu la fonction d'activation par défaut pour de nombreux types de réseaux de neurones, car un modèle qui l'utilise est plus facile à former et atteint souvent de meilleures performances [16].

La conception d'unités cachées est un domaine de recherche extrêmement actif et ne dispose pas encore de nombreux principes directeurs théoriques définitifs. Les fonctions d'activations ReLU sont un excellent choix par défaut d'unité cachée.
De nombreux autres types d'unités cachées sont disponibles. Il peut être difficile de déterminer quand utiliser quel type, bien que les unités linéaires rectifiées soient généralement un choix acceptable [17]. ReLU peut résoudre le problème des gradients de fuite, consultez le didacticiel [24].

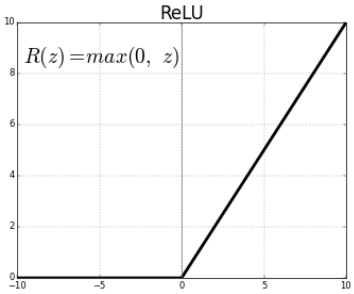


FIGURE 9 : ReLU

C *Autres fonctions d'activations*

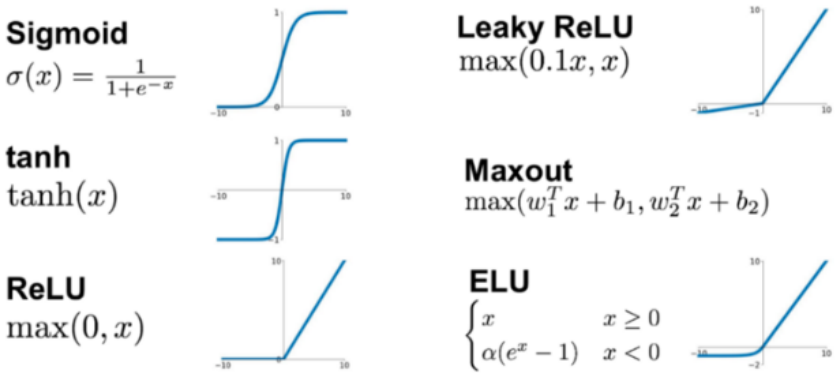


FIGURE 10 : Les différentes fonctions d'activation avec leurs graphes

1.3.3 *Réseau neuronal convolutif (CNN)*

Le réseau neuronal convolutif est un type de réseau neuronal artificiel (CNN, Convolutional Neural Network ou ConvNet) qui utilise plusieurs perceptrons qui analysent les entrées d'image et ont des poids et des bases apprenables sur plusieurs parties d'images et capables de se séparer les unes des autres [30].

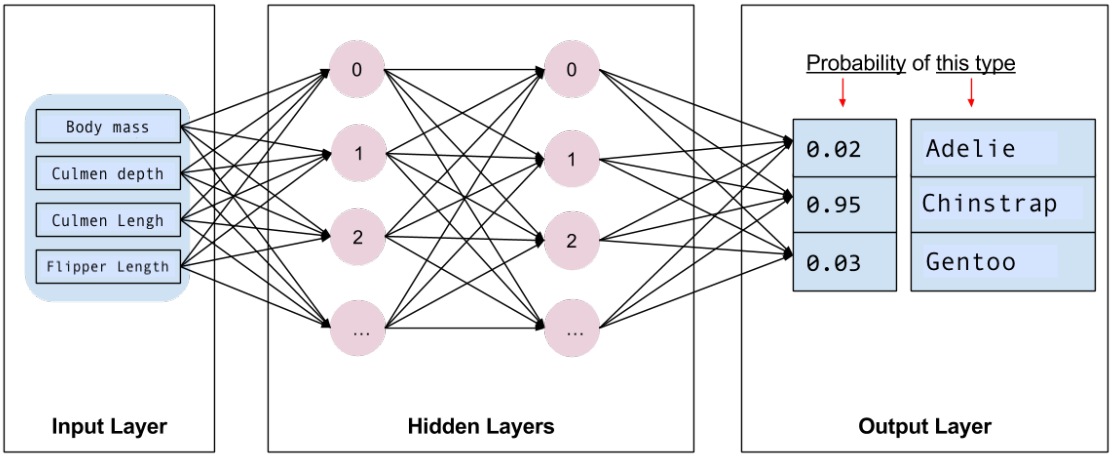


FIGURE 11 : Exemple de la classification avec un CNN à différentes couches : la couche d'entrée, les couches cachées et la couche de sortie (respectivement en anglais : Input Layer, Hidden Layers and Output Layer) [27]

.

Le CNN est un type de réseau de neurones acyclique à propagation avant, dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont arrangés de sorte à ce qu'ils correspondent à des régions (appelés champs réceptifs) qui se chevauchent lors du pavage du champ visuel. Ils sont de plus organisés de manière hiérarchique, en couches (aire visuelle primaire V1, secondaire V2, puis aires V3, V4, V5 et V6, gyrus temporal inférieur), chacune des couches étant spécialisée dans une tâche, de plus en plus abstraite [3]. En simplifiant à l'extrême, une fois que les signaux lumineux sont reçus par la rétine et convertis en potentiels d'action :

- L'aire primaire V1 s'intéresse principalement à la détection de contours, ces contours étant définis comme des zones de fort contraste de signaux visuels reçus.
- L'aire V2 reçoit les informations de V1 et extrait des informations telles que la fréquence spatiale, l'orientation, ou encore la couleur.
- L'aire V4, qui reçoit des informations de V2, mais aussi de V1 directement, détecte des caractéristiques plus complexes et abstraites liées par exemple à la forme.
- Le gyrus temporal inférieur est chargé de la partie sémantique (reconnaissance des objets), à partir des informations reçues des aires précédentes et d'une mémoire des informations stockées sur des objets.

L'architecture et le fonctionnement des réseaux convolutifs sont inspirés par ces processus biologiques. Ces réseaux consistent en un empilage multicouche de perceptrons [30], dont le but est de pré-traiter de petites quantités d'informations.
Un réseau convolutif se compose de deux types de neurones, agencés en couches traitant successivement l'information. Dans le cas du traitement de données de type images [3], on a ainsi :

- des neurones de traitement, qui traitent une portion limitée de l'image (le champ réceptif) au travers d'une fonction de convolution[3, 30] ;

- des neurones de mise en commun des sorties dits d'agrégation totale ou partielle (pooling) [3, 30].

Un traitement correctif non linéaire est appliqué entre chaque couche pour améliorer la pertinence du résultat. L'ensemble des sorties d'une couche de traitement permet de reconstituer une image intermédiaire, dite carte de caractéristiques (feature map), qui sert de base à la couche suivante. Les couches et leurs connexions apprennent des niveaux d'abstraction croissants et extraient des caractéristiques de plus en plus haut niveau des données d'entrée [3, 28].

L'un des avantages de l'utilisation du réseau de neurones convolutifs est qu'il exploite l'utilisation de la cohérence spatiale locale dans les images d'entrée, ce qui leur permet d'avoir moins de poids car certains paramètres sont partagés [30].

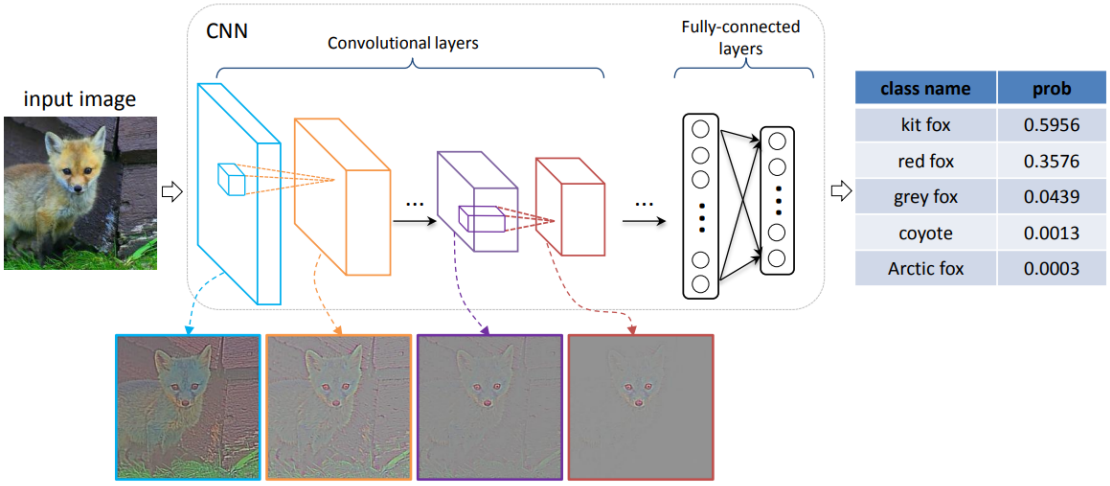


FIGURE 12 : L'illustration du comportement externe et interne d'un CNN. Le comportement externe correspond aux catégories de prédiction de sortie pour les images d'entrée. Le comportement interne est à sonder en visualisant les espaces de représentation construits par chaque couche et les informations visuelles conservées dans chaque couche. [32]

A • *Couche de convolution*

La couche de convolution est la pierre angulaire du CNN. Il porte la majeure partie de la charge de calcul du réseau.

Cette couche effectue un produit scalaire entre deux matrices, où une matrice est l'ensemble de paramètres apprenables autrement connu sous le nom de noyau (en anglais kernel) K ou encore filtre de convolution, et l'autre matrice est la partie restreinte du champ récepteur I. Le noyau est spatialement plus petit qu'une image mais il est plus en profondeur. Cela signifie que, si l'image est composée de trois canaux (RVB), la hauteur et la largeur du noyau seront spatialement petites, mais la profondeur s'étend jusqu'aux trois canaux [17].

DÉFINITION : *Soient* $h_1,\ h_2 \in \mathbb{N}, K \in \mathbb{R}^{(2h_1+1)\times(2h_2+1)}$. *La convolution de* I *par* K *est donnée par :*

$$(I * K)_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \tag{16}$$

où K est donnés par :

$$K = \begin{bmatrix} K_{-h_1,-h_2} & \cdots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \cdots & K_{h_1,h_2} \end{bmatrix}$$

La taille du filtre $(2h_1+1)\times(2h_2+1)$ précises le champ visuel capturé et traité par K. Lorsque K parcourt I, le déplacement du filtre est réglé par deux paramètres de *stride* (horizontal et vertical). Un stride de 1 horizontal (respectivement vertical) signifie que K se déplace d'une position horizontale (resp. verticale) à chaque application de la formule 16. Les valeurs de stride peuvent également être supérieures et ainsi sous-échantillonner I [3, 17].
Le comportement du filtre sur les bords de I doit également être précisé, par l'intermédiaire d'un paramètre de *padding*. Si l'image convoluée $(I * K)$ doit posséder la même taille que I, alors $2h_1$ lignes de 0 (h_1 à gauche et $2h_1$ à droite) et $2h_2$ colonnes de 0 (h_2 en haut et h_2 en bas) doivent être ajoutées. Dans le cas où la convolution est réalisée sans padding, l'image convoluée est de taille $n_1 - 2h_1 \times n_2 - 2h_2$ [3].

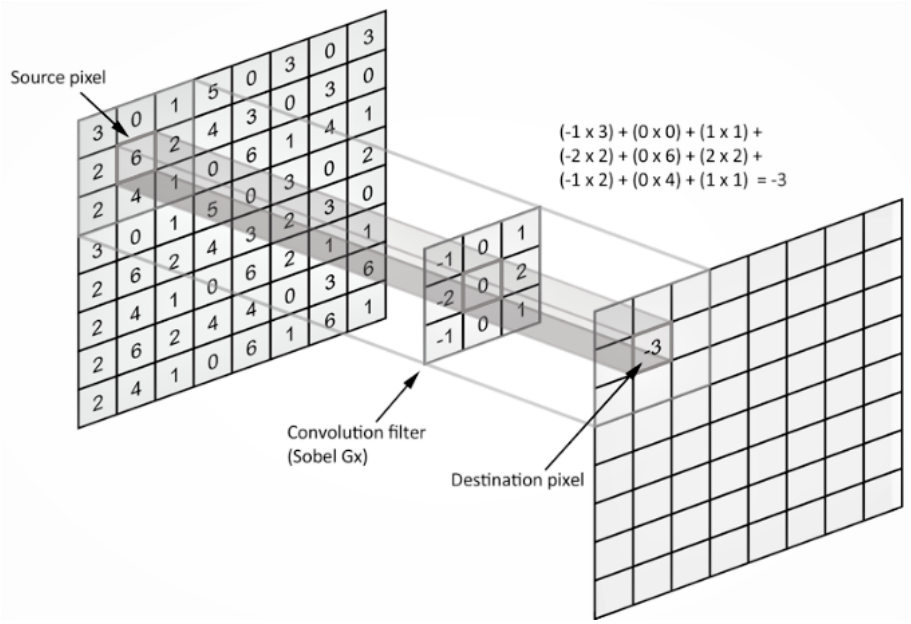


FIGURE 13 : Illustration des calculs effectués dans une opération de convolution [3].

Par conséquent, la couche de convolution prend plusieurs images en entrée et utilise chaque filtre pour calculer la convolution de chaque image. Le filtre correspond exactement à la caractéristique que nous voulons trouver dans l'image [28]. Pour chaque paire (image, filtre), obtenez une carte d'activation ou une carte d'entités montrant où se trouvent les entités dans l'image. Plus la valeur est élevée, plus les points correspondants dans l'image sont similaires à l'entité [17].

B *L'architecture VGGNet*

VGG signifie Visual Geometry Group, il s'agit d'une architecture standard de réseau de neurones à convolution profonde (CNN) à plusieurs couches [29].

Les réseaux VGG ont été les premiers à utiliser de petits filtres de convolution (3×3) et à les combiner pour décrire des séquences de convolution, l'idée étant d'émuler l'effet de larges champs réceptifs par cette séquence. Cette technique amène malheureusement à un nombre exponentiel de paramètres (le modèle entraîné qui peut être téléchargé a une taille de plus de 500 Mo). VGG a concouru à ILSVRC 2014, a obtenu un taux de bonne classification de 92.3% mais n'a pas remporté le concours [20]. Aujourd'hui, VGG est une famille de réseaux profonds (de A à E) qui varient par leur architecture (figure 15).

L'architecture VGG est la base d'un modèle innovant de reconnaissance d'objets. Développé en tant que réseau neuronal profond, VGGNet va au-delà d'ImageNet et dépasse la ligne de base de nombreuses tâches et ensembles de données. De plus, c'est toujours l'une des architectures de reconnaissance d'images les plus populaires [3, 30].

Les VGGNet sont basés sur les caractéristiques les plus essentielles des réseaux de neurones convolutifs (CNN). Le graphique suivant montre le concept de base du fonctionnement d'un CNN.

Un bref coup d'œil à l'architecture de VGG :

- **Entrée** : Le VGGNet prend une taille d'entrée d'image de 224×224. Pour le concours ImageNet, les créateurs du modèle ont recadré le patch central 224 × 224 dans chaque image pour conserver la cohérence de la taille d'entrée de l'image [29].
- **Couches convolutives** : Les couches convolutives de VGG tirent parti d'un champ de réception minimal, c'est-à-dire 3 × 3, la plus petite taille possible qui capture toujours haut/bas et gauche/droite. De plus, il existe également des filtres de convolution 1 × 1 agissant comme une transformation linéaire de l'entrée. Vient ensuite une unité ReLU, qui est une énorme innovation d'AlexNet qui réduit le temps de formation. La foulée de convolution est fixée à 1 pixel pour conserver la résolution spatiale préservée après la convolution (la foulée est le nombre de décalages de pixels sur la matrice d'entrée) [20, 30].
- **Couches cachées** : Toutes les couches cachées du réseau VGG utilisent ReLU. VGG n'utilise généralement pas la normalisation de la réponse locale (LRN) car elle augmente la consommation de mémoire et le temps de formation. De plus, il n'apporte aucune amélioration à la précision globale [30].
- **Couches entièrement connectées** : Le VGGNet a trois couches entièrement connectées. Sur les trois couches, les deux premières ont 4096 canaux chacune et la troisième a 1000 canaux, 1 pour chaque classe [30].

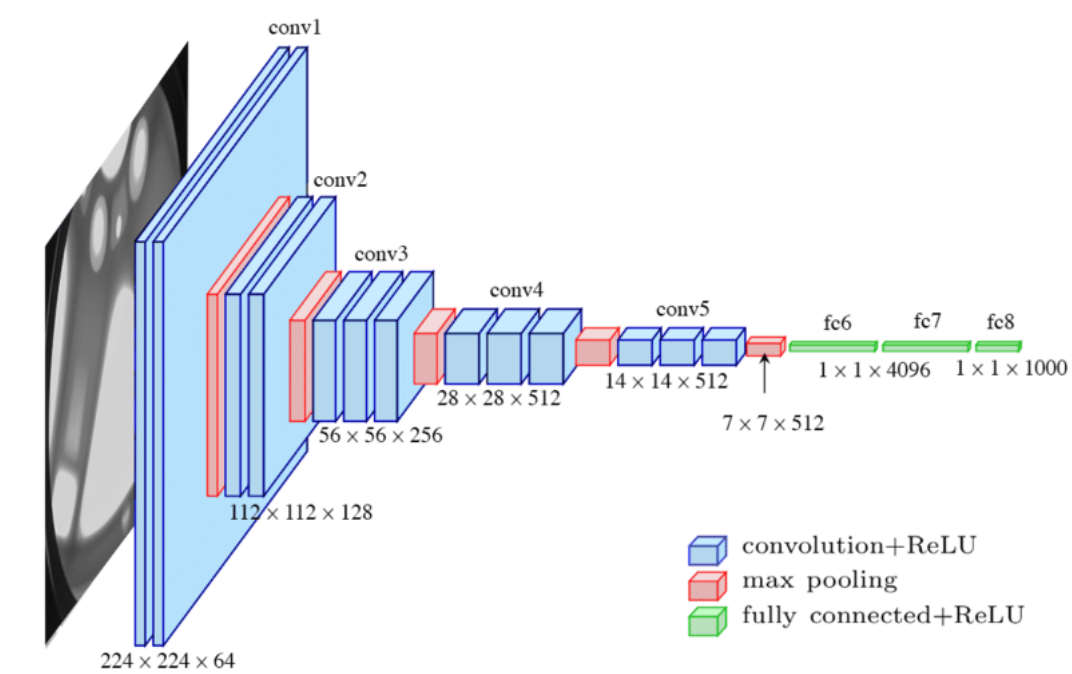


FIGURE 14 : CNN : architecture VGG [27]

.

LE MODÈLE VGG16 ET VGG19 : Visual Geometry Group et se compose de blocs, où chaque bloc est composé de couches 2D Convolution et Max Pooling. Il se décline en deux modèles - VGG16 et VGG19 - avec 16 et 19 couches [32].

VGG16 (également VGGNet-16) est modèle VGGNet, qui prend en charge 16 couches convolutives. Le nombre de filtres que nous pouvons utiliser double à chaque étape ou à travers chaque pile de la couche de convolution. C'est un principe majeur utilisé pour concevoir l'architecture du réseau VGG16. L'un des principaux inconvénients du réseau VGG16 est qu'il s'agit d'un réseau énorme, ce qui signifie qu'il faut plus de temps pour former ses paramètres [32].
En raison de sa profondeur et du nombre de couches entièrement connectées, le modèle VGG16 fait plus de 533 Mo. Cela rend la mise en œuvre d'un réseau VGG une tâche fastidieuse.
Le modèle VGG16 est utilisé dans plusieurs problèmes de classification d'images d'apprentissage en profondeur, mais des architectures de réseau plus petites telles que GoogLeNet et SqueezeNet sont souvent préférables. Dans tous les cas, le VGGNet est un excellent élément de base à des fins d'apprentissage car il est simple à mettre en œuvre.

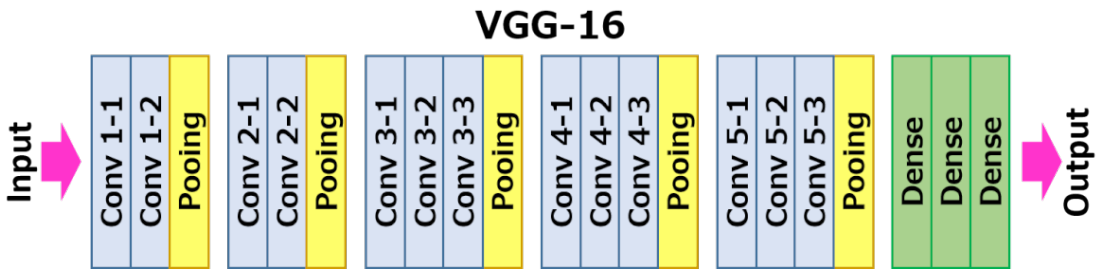


FIGURE 15 : Le modèle VGG-16 [30]

Le concept du modèle **VGG19** (également VGGNet-19) est le même que celui du VGG16, sauf qu'il prend en charge 19 couches. Le « 16 » et le « 19 » représentent le nombre de couches de poids dans le modèle (couches convolutives). Cela signifie que VGG19 a trois couches convolutionnelles de plus que VGG16. Nous aborderons plus en détail les caractéristiques des réseaux VGG16 et VGG19 dans la dernière partie de cet article [32].

2

RÉSULTATS ET DISCUSSION

2.1 EXPÉRIMENTATION

2.1.1 *Outils, Choix des technologies*

A *Language & technologie*

Le langage de programmation et technologie utilisée

Python : est le language majeur utilisé dans ce projet, c'est un langage de programmation interprété de haut niveau. Sa philosophie de conception met l'accent sur la lisibilité du code avec l'utilisation d'une indentation significative. Python est typé dynamiquement et ramassé.

TensorFlow : c'est une bibliothèque de logiciels gratuite et open source pour l'apprentissage automatique et l'intelligence artificielle. TensorFlow fournit un ensemble de workflows pour développer et former des modèles à l'aide de Python ou JavaScript. Il peut être utilisé dans une gamme de tâches, mais se concentre particulièrement sur la formation et l'inférence des réseaux de neurones profonds.

OpenCV : est une bibliothèque de fonctions de programmation principalement destinées à la vision par ordinateur en temps réel. OpenCV fournit une bibliothèque, des outils et du matériel de vision par ordinateur optimisés en temps réel. Il prend également en charge l'exécution de modèles pour Machine Learning.

LabelImg : est un outil gratuit et open source pour l'étiquetage graphique, un outil graphique d'annotation d'images.
Il est écrit en Python et utilise Qt pour son interface graphique. Les annotations sont enregistrées sous forme de fichiers XML au format PASCAL VOC, le format utilisé par ImageNet. Il est écrit en Python et utilise QT pour son interface graphique. C'est un moyen facile et gratuit d'étiqueter quelques centaines d'images pour notre projet de détection d'objets. En outre, il prend également en charge les formats YOLO et CreateML.

B *choix du modèle de CNN*

Le VGGNet (Visual Geometry Group), Le modèle VGG16 du VGGNet atteint près de 92,7 % de précision dans le top 5 des tests dans ImageNet. le VGGNet-16 prend en charge 16 couches (tout comme le VGG19 prend en charge 19 couches.) et peut classer les images en 1000 catégories d'objets,de plusieure catégorie de plus, le modèle a une taille d'entrée d'image de 224 par 224 (*cf.* le chapitre 1, section 1.3.3, point b).

VGG16 surpasse largement les versions précédentes des modèles des compétitions ILSVRC-2012 et ILSVRC-2013. De plus, le résultat VGG16 est en compétition pour le

vainqueur de la tâche de classification (GoogLeNet avec une erreur de 6,7%) et surpasse considérablement la soumission gagnante ILSVRC-2013 Clarifai. Il a obtenu 11,2% avec des données de formation externes et environ 11,7% sans elles. En termes de performances à réseau unique, le modèle VGGNet-16 obtient le meilleur résultat avec environ 7,0% d'erreur de test, dépassant ainsi un seul GoogLeNet d'environ 0,9% [29]

	Training accuracy	**Validation accuracy**
Réseau neuronal convolutif de base	`98.20%`	`72.40%`
Réglage fin de CNN avec augmentation d'image	`81.30%`	`79.20%`
Réglage fin de CNN avec modèle VGG-16 pré-entraîné et augmentation d'image	`86.50%`	`95.40%`

Le tableau ci-dessus montre la précision de la formation et de la validation (training accuracy & validation accuracy) pour différents modèles de réseaux neuronaux, résultats comparatifs de [30].

Nous avons aussi fait un test comparatif entre VGG-19 et VGG-16 pour savoir quel modèle est plus adapté à notre cas de reconnaissance des plaques d'immatriculation. Le résultats comparatifs dans le tableau ci-dessous, qui nous montre le nombre total de paramètres, le nombre de paramètres entrainables, le nombre de paramètres non-entrainables, la perte, la précision (respectivement : total params, trainable params, non-trainable params, loss, accuracy).

Message de sortie du Modèle VGG19	
Total params : 24,784,644	loss : 4.0037e-04
Trainable params : 4,760,260	accuracy : 0.9636
Non-trainable params : 20,024,384	validation loss : 0.0109
	validation accuracy : 0.7727
Message de sortie du Modèle VGG16	
Total params : 19,474,948	loss : 3.4066e-04
Trainable params : 4,760,260	accuracy : 0.9570
Non-trainable params : 14,714,688	validation loss : 0.0102
	validation accuracy : 0.8182

Peut importe le nombre de couches, 19 ou 16, le deux modèle génère le même nombre de paramètres entrainable (donc minimisable) du coup il y a rien d'intéressant à prendre le VGG-19 vu que sa précision de validation est faible comparé au modèle VGG-16 et qu'il n'y a pas un grand écart entre leurs précision et d'entraînement.

2.1.2 *Élaboration du dataset*

C'est une tâche, fastidieuse dans le l'apprentissage automatique, qui consiste à montrer à la machine où se trouve la plaque d'immatriculation (la zone précise) sur l'image. Ce

processus aussi nommé étiquetage est très important dans l'étape de l'apprentissage supervisé. Et il faut faire ça pour toute 430 images qui constituent notre Dataset.

Dataset est construit avec l'outil **LabelImg** est un outil graphique d'annotation d'images. Le Dataset est un répertoire composé des sous répertoire qui contient :

- Les annotations sont des fichiers xml, qui contient les coordonnées des régions intérêt sur les images (les étiquettes).
- Les differentes images des plaques d'immatriculations.

Nous avons 433 images des plaques d'immatriculations pour 433 fichiers xml d'annotations. Le dataset est fractionné en trois sous ensemble proportionnellement : 70% pour l'entrainement, 10% pour la validation, 20% pour le teste.

FIGURE 16 : Exemple des quelques images de notre dataset sans annotations

FIGURE 17 : Exemple des images (figure 16) de notre dataset sans annotations

2.1.3 *Entraînement du CNN (VGG)*

A *Élaboration du modèle*

```
import keras
from keras.models import Sequential, Model
from keras.layers import Dense, Flatten, Dropout
from keras.applications.vgg19 import VGG19
from keras.applications.vgg16 import VGG16

IMAGE_SIZE = 200

model = Sequential()

model.add(VGG16(
        weights="imagenet",
        include_top=False,
        input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3))
)
model.add(Flatten())
model.add(Dropout(0.4))
model.add(Dense(256, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(64,  activation="relu"))
model.add(Dense(4,   activation="sigmoid"))
model.layers[-7].trainable = False

model.summary()
```

Ci-dessus se trouve le code pour construire un modèle pré-entraîné VGG-16. Nous devons inclure $weights = \text{"}imagenet\text{"}$ pour récupérer le modèle VGG-16 qui est formé sur l'ensemble de données imagenet. Il est important de définir $include\_top = False$ pour éviter de télécharger les couches entièrement connectées du modèle pré-entraîné [30]. Nous devons ajouter notre propre classificateur car le classificateur de modèle pré-entraîné a plus de 1 classe alors que notre objectif est de classer l'image en 1 classe (image de plaques). Une fois que les couches convolutionnelles du modèle pré-entraîné ont extrait les caractéristiques d'image de bas niveau telles que les bords, les lignes et les blobs, la couche entièrement connectée les classe ensuite en 1 catégories.

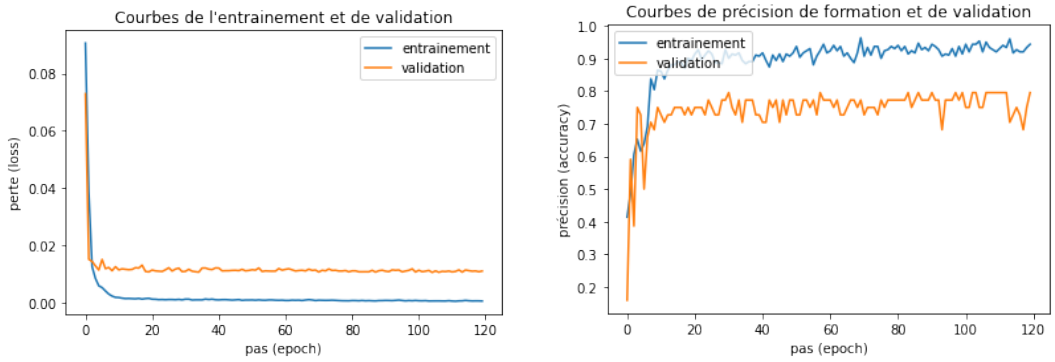
Nous avons crée un modèle **"Séquentiel"** et ajoutons **VGG16** comme première couche. Le modèle prendra en entrée des tableaux de taille $(200 \times 200 \times 3)$ qui correspondent à la taille d'une image du dataset. La sortie du VGG16 passe dans **Flatten** une couche qui a aplatis le tableau $(200 \times 200 \times 3)$ en un tableau de taille (18432×1). Enfin nous ajoutons 4 couches **Dense** qui feront office d'un réseau complètement connecté avec comme sortie un tableau de taille (4×1) qui correspond au 4 points qui nous permettront d'obtenir ue rectangle contenant la plaque d'immatriculation sur l'image.

Ci-dessous le message de sortie, les informations sur notre reseaux de neurones construit.

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 6, 6, 512)	14714688
flatten (Flatten)	(None, 18432)	0
dropout (Dropout)	(None, 18432)	0
dense (Dense)	(None, 256)	4718848
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 4)	260

Total params: 19,474,948
Trainable params: 4,760,260
Non-trainable params: 14,714,688

Il est difficile de trouver un critère général pour arrêter cet algorithme. Le problème est que l'erreur tend à diminuer lentement et à ne jamais se stabiliser complètement, ce qui mène à un surapprentissage. La meilleure manière d'éviter ce phénomène est d'utiliser un ensemble de validation.



(a) loss : perte (b) accuracy : précision

FIGURE 18 : Graphe de précision (accuracy) et perte (loss) du modèle VGG

Le script ci-dessous est une partie du programme que nous avons élaboré pour l'entraînement du CNN. Ici nous entraînons avec l'optimiseur SGD et un taux d'apprentissage de $1e-4 = 1.10^{-4}$. Dans la section 2.2.1 nous montrons les résultats avec différents optimiseurs.

```
import keras.optimizers as optimizers
import keras.losses as losses
optimizer = optimizers.SGD(learning_rate=1e-4)
loss = losses.MeanSquaredError(reduction="auto", name="mean_squared_error")
model.compile(loss=loss, optimizer=optimizer, metrics=['accuracy'])
model.fit(X_train, y_train,
        validation_data=(X_val, y_val),
        epochs=200,
        batch_size=32,
        verbose=1
)
```

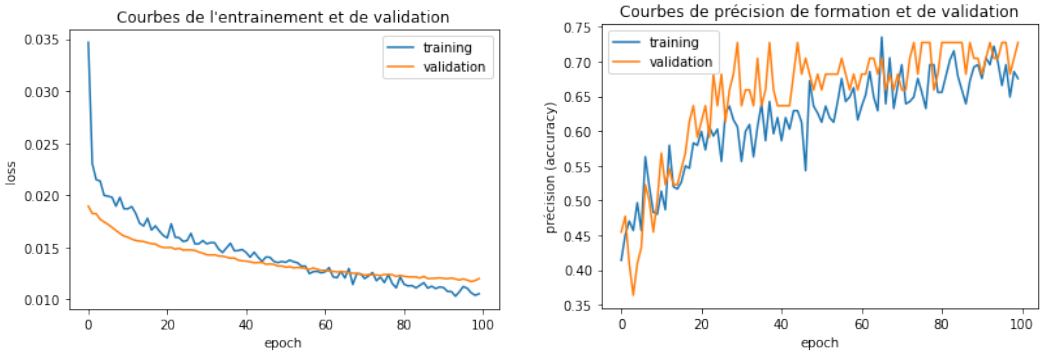
2.2 ÉVALUATION DU RÉSULTATS D'EXPÉRIMENTATION

2.2.1 *Résultat comparatif des différent optimiseurs*

Pour le même modèle élaboré pour la reconnaissance de plaque d'immatriculation. Nous allons tester son efficacité après entraînement en fonction des différents optimiseurs. Nous évaluons les quelques optimiseurs candidats pour cette études (*cf.* chapitre **??**, section **??**). Voici une liste non exhaustive des optimiseurs étudiés.

- *SGD*

Training	**Validation**	**Test**
`loss: 0.0124`	`val loss: 0.0156`	`test loss: 0.0092`
`accuracy: 0.6291`	`val accuracy: 0.6591`	`test accuracy: 0.5946`



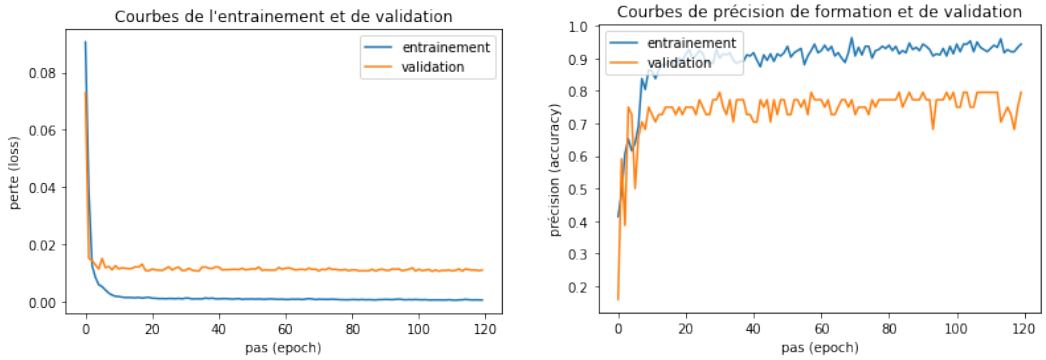
(a) loss : perte

(b) accuracy : précision

FIGURE 19 : Graphe de précision (accuracy) et perte (loss) pour RMSprop

• *RMSprop*

Training	Validation	Test
`loss: 7.0379e-04`	`val loss: 0.0109`	`test loss : 0.005158`
`accuracy: 0.9470`	`val accuracy: 0.7727`	`test accuracy : 0.86206`

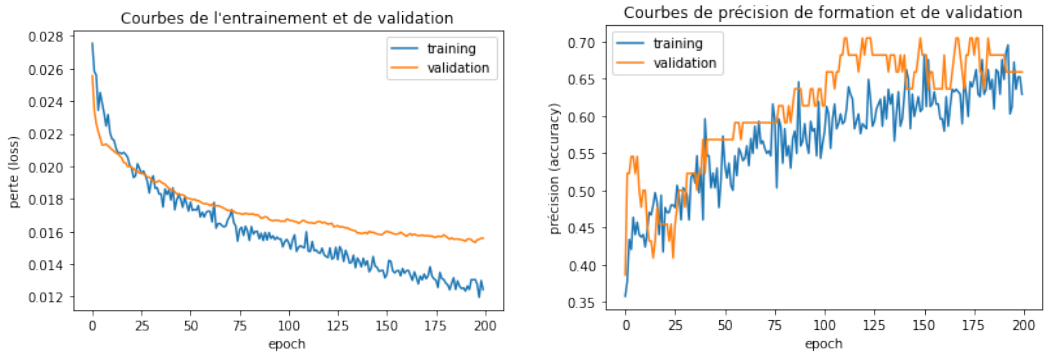


(a) loss : perte (b) accuracy : précision

FIGURE 20 : Graphe de précision (accuracy) et perte (loss) pour RMSprop

• *Adagrad*

Training	Validation	Test
`loss: 0.0124`	`val loss: 0.0156`	`test loss: 0.0092`
`accuracy: 0.6291`	`val accuracy: 0.6591`	`test accuracy: 0.5946`

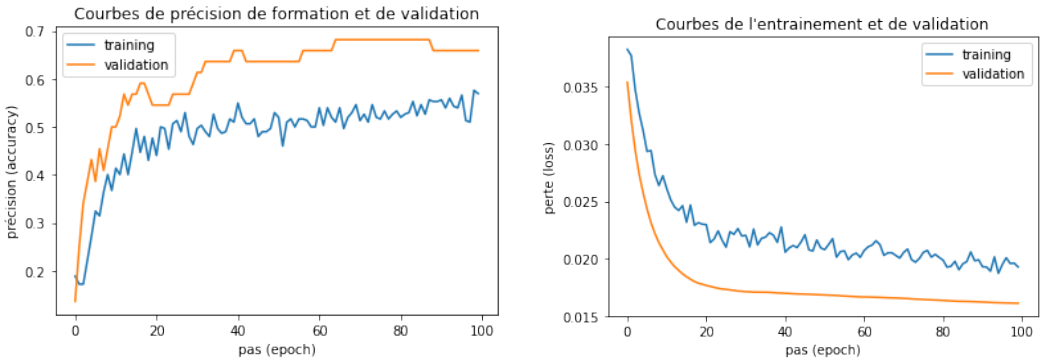


(a) loss : perte (b) accuracy : précision

FIGURE 21 : Graphe de précision (accuracy) et perte (loss) pour Adagrad

• *Adadelta*

Training	Validation	Test
`loss: 0.0130`	`val loss: 0.0156`	`test loss : 0.01374378`
`accuracy: 0.6523`	`val accuracy: 0.6591`	`test accuracy : 0.54022`

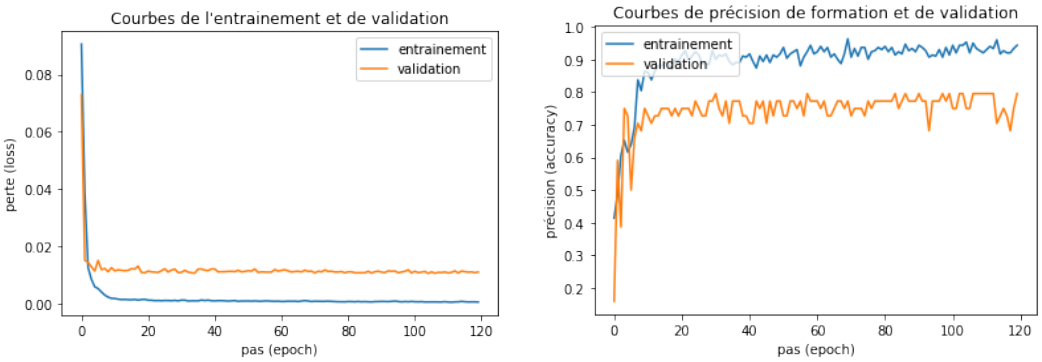


(a) loss : perte (b) accuracy : précision

FIGURE 22 : Graphe de précision (accuracy) et perte (loss) pour Adadelta

• *Adam*

Training	Validation	Test
`loss: 4.0037e-04`	`val loss: 0.0109`	`test loss : 0.00407`
`accuracy: 0.9636`	`val accuracy: 0.7727`	`test accuracy : 0.88505`

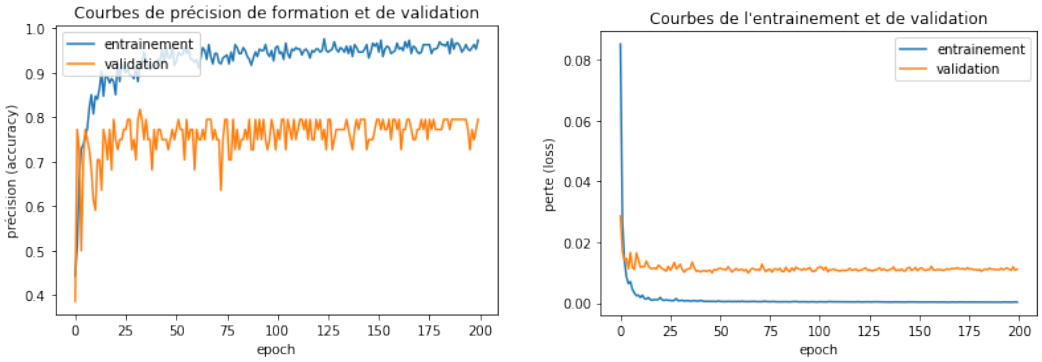


(a) loss : perte (b) accuracy : précision

FIGURE 23 : Graphe de précision (accuracy) et perte (loss) pour Adam

- *Nadam*

Training	Validation	Test
loss: 3.0496e-04	val loss: 0.0111	test loss : 0.004073061
accuracy: 0.9536	val accuracy: 0.7955	test accuracy : 0.87356



(a) loss : perte (b) accuracy : précision

FIGURE 24 : Graphe de précision (accuracy) et perte (loss) pour Nadam

Nous remarquons RMSprop minimise le mieux les erreurs par rapport aux autres optimiseurs mais il n'est pas précis par rapport à Nadam ou Adam. Ci-dessous une liste des optimiseurs qui minimise le mieux dans le cas de notre problème reconnaissance de plaque d'immatriculation.

N°	Training	Validation	Test
1. RMSprop	7.0379e-04	0.0109	0.005158
2. Nadam	3.0496e-04	0.0111	0.004073
3. Adam	4.0037e-04	0.0109	0.00407
4. Adagrad	0.0124	0.0156	0.0092
5. Adadelta	0.0130	0.0156	0.013743

2.2.2 *Invariance des transformation géométrique*

Nous avons dit en amont que le modèle entraîné doit reconnaître les objet même après transformation géométrique dans l'image. Nous allons examiner l'efficacité par rapport au 3 meilleur optimiseurs c.a.d les 3 modèle qui ont on eu un bon score d'entraînement, de validation et de test.

	Training	Validation	Test
Nadam	95.36%	79.55%	87.35%
Adam	96.36%	77.27%	88.50%
RMSprop	94.70%	77.27%	86.20%

??? (Suite)

2.3 SOMMAIRE DU CHAPITRE

C'est là toute la force des réseaux de neurones convolutifs : ceux-ci sont capables de déterminer tout seul les éléments discriminants d'une image, en s'adaptant au problème posé. Par exemple, si la question est de distinguer les chats des chiens, les features automatiquement définies peuvent décrire la forme des oreilles ou des pattes.

BIBLIOGRAPHIE

[1] Yaovi Ahadjitse. "Reconnaissance d'objets en mouvement dans la vidéo par description géométrique et apprentissage supervisé". Thèse de doct. Université du Québec en Outaouais, 2013.

[2] E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2010. isbn : 9780262012430.

[3] Vincent Barra Antoine Cornuéjols Laurent Michet. *Apprentissage artificiel : Deep leaning, concepts et algorithmes*. 3rd. Eyrolles, 2018, p. 239-263.

[4] P. Benner, M. Bollhöfer, D. Kressner, C. Mehl et T. Stykel. *Numerical Algebra, Matrix Theory, Differential-Algebraic Equations and Control Theory : Festschrift in Honor of Volker Mehrmann*. Springer International Publishing, 2015. isbn : 9783319152608. url : https://books.google.cd/books?id=MilACQAAQBAJ.

[5] M. Bierlaire. *Introduction à l'optimisation différentiable*. Enseignement des mathématiques. Presses polytechniques et universitaires romandes, 2006. isbn : 9782880746698. url : https://books.google.cd/books?id=HK55T5x2bygC.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. First. Springer-Verlag New York, 2006, p. 179-195.

[7] Léon Bottou. "Large-scale machine learning with stochastic gradient descent". In : *Proceedings of COMPSTAT'2010*. Springer, 2010, p. 177-186.

[8] Léon Bottou. "Stochastic gradient descent tricks". In : *Neural networks : Tricks of the trade*. Springer, 2012, p. 421-436.

[9] Léon Bottou, Frank E Curtis et Jorge Nocedal. "Optimization methods for large-scale machine learning". In : *Siam Review* 60.2 (2018), p. 223-311.

[10] F. Coulombeau, G. Debeaumarché, B. David, F. Dorra, S. Dupont et M. Hochart. *Mathématiques MPSI-PCSI : Programme 2013 avec algorithmique en Scilab*. Cap Prépa. Pearson, 2013. isbn : 9782744076527.

[11] Pádraig Cunningham, Matthieu Cord et Sarah Jane Delany. "Supervised learning". In : *Machine learning techniques for multimedia*. Springer, 2008, p. 21-49.

[12] R.B. Darlington et A.F. Hayes. *Regression Analysis and Linear Models : Concepts, Applications, and Implementation*. Methodology in the Social Sciences. Guilford Publications, 2016. isbn : 9781462521135.

[13] Natarajan Deepa, B Prabadevi, Praveen Kumar Maddikunta, Thippa Reddy Gadekallu, Thar Baker, M Ajmal Khan et Usman Tariq. "An AI-based intelligent system for healthcare analysis using Ridge-Adaline Stochastic Gradient Descent Classifier". In : *The Journal of Supercomputing* 77 (2021), p. 1998-2017.

[14] Kary Främling. "Scaled Gradient Descent Learning Rate". In : *Reinforcement Learning With Light-Seeking Robot, Proceedings of ICINCO* (2004), p. 1-8.

[15] Yoav Freund et Robert E Schapire. "Large margin classification using the perceptron algorithm". In : *Machine learning* 37.3 (1999), p. 277-296.

[16] A. Géron. *Hands-on Machine Learning with Scikit-Learn and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017. ISBN : 9781491962299.

[17] I. Goodfellow, Y. Bengio et A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN : 9780262035613.

[18] F.E. Harrell et F.E.H. jrl. *Regression Modeling Strategies : With Applications to Linear Models, Logistic Regression, and Survival Analysis*. Graduate Texts in Mathematics. Springer, 2001. ISBN : 9780387952321. URL : https://books.google.cd/books?id=kfHrF-bVcvQC.

[19] Daniel Kirsch Judith Hurwitz. *Machine Learning For Dummies*. IBM Limited Edition. John Wiley et Sons, Inc., 2018.

[20] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In : *Advances in neural information processing systems* 25 (2012).

[21] N. Matloff. *Statistical Regression and Classification : From Linear Models to Machine Learning*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, 2017. ISBN : 9781351645898.

[22] Praneeth Netrapalli. "Stochastic gradient descent and its variants in machine learning". In : *Journal of the Indian Institute of Science* 99.2 (2019), p. 201-213.

[23] Jorge Nocedal et Stephen J Wright. *Numerical optimization*. T. 35. 1999.

[24] Arnu Pretorius, Elan Van Biljon, Steve Kroon et Herman Kamper. "Critical initialisation for deep signal propagation in noisy rectifier neural networks". In : *Advances in Neural Information Processing Systems* 31 (2018).

[25] D. Sarkar, R. Bali et T. Sharma. *Practical Machine Learning with Python : A Problem-Solver's Guide to Building Real-World Intelligent Systems*. Apress, 2017. ISBN : 9781484232071.

[26] Carl-Erik Särndal, Bengt Swensson et Jan Wretman. *Model assisted survey sampling*. Springer Science & Business Media, 2003.

[27] Vahid Mirjalili Sebastien Raschka. *Python Machine Learning and Deep Learning, with sckit-learn and Tensorflow*. 2nd. Packt, 2017, p. 17-139.

[28] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura et Ronald M Summers. "Deep convolutional neural networks for computer-aided detection : CNN architectures, dataset characteristics and transfer learning". In : *IEEE transactions on medical imaging* 35.5 (2016), p. 1285-1298.

[29] Karen Simonyan et Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In : *arXiv preprint arXiv :1409.1556* (2014).

[30] Srikanth Tammina. "Transfer learning using VGG-16 with deep convolutional neural network for classifying images". In : *International Journal of Scientific and Research Publications (IJSRP)* 9.10 (2019), p. 143-150.

[31] Rob GJ Wijnhoven et PHN de With. "Fast training of object detection using stochastic gradient descent". In : *2010 20th International Conference on Pattern Recognition*. IEEE. 2010, p. 424-427.

[32] Wei Yu, Kuiyuan Yang, Yalong Bai, Tianjun Xiao, Hongxun Yao et Yong Rui. "Visualizing and comparing AlexNet and VGG using deconvolutional layers". In : *Proceedings of the 33 rd International Conference on Machine Learning*. 2016.