

ÉCOLE SUPÉRIEURE DE GÉNIE INFORMATIQUE



école supérieure de  
génie informatique

Quang Dat LE - Classe: 3IABD-1

Herilala MIADA - Classe: 3IABD-1

Nabil SARKER - Classe: 3IABD-1

## RAPPORT DE PROJET ANNUEL 3ÈME ANNÉE

Spécialité: Intelligence Artificielle & Big Data

Île-de-France, France – 2024

## REMERCIEMENTS

Nous tenons tout d'abord à exprimer notre profonde gratitude à Monsieur Nicolas VIDAL, dont les enseignements et conseils ont été essentiels à la réalisation de ce projet. Nous remercions également l'École Supérieure de Génie Informatique pour son soutien académique et les ressources mises à notre disposition tout au long de notre parcours.

Nous souhaitons également exprimer notre reconnaissance à nos coéquipiers, dont le dévouement et le travail acharné ont été déterminants dans le succès de ce projet. Leur collaboration et leur esprit d'équipe ont grandement contribué à l'aboutissement de cette thèse.

<b>Introduction</b>	<b>4</b>
<b>Chapitre 1 : Recherche sur les Modèles Linéaires (LM)</b>	<b>7</b>
1.1 Définition et concepts fondamentaux du modèle linéaire	7
1.1.1 Le problème général des fonctions à une variable se pose comme suit :	7
1.1.2 Application de la matrice 2D en apprentissage automatique:	7
1.1.2.1 Méthode de Rosenblatt (Perceptron)	8
1.1.2.2 Méthode de Descente de Gradient	8
1.2 Architecture d'un modèle linéaire	9
1.2.1. Perceptron	9
1.2 Modèle linéaire : régression linéaire	11
<b>Chapitre 2 : Recherche sur les Réseaux de Neurones Multicouches (MLP)</b>	<b>14</b>
2.1. Réseau de neurones artificiels et architecture de réseau MLP	14
2.2 Architecture des réseaux de neurones multicouches	15
2.3 Algorithmes d'apprentissage pour les MLP	15
2.4 Expérimentation avec l'ensemble de données MNIST.	18
<b>Chapitre 3 : Collecte et Traitement de l'Ensemble de Données</b>	<b>20</b>
3.1 Méthodologie de collecte des données	20
3.1.1 Plugin sur navigateur: WebScraper	20
3.2 Prétraitement des données	23
3.3 Conversion des images en matrices	23
3.4 Construction de l'ensemble de données final	25
3.4.1 Berline, SUV, Voiture de Sport	25
3.4.2 Avion, Moto, Voiture	26
<b>Chapitre 4 : Intégration et simulation de modèles LM et MLP</b>	<b>26</b>
4.1 Application du réseau neuronal multicouche	26
4.2 Simulation du système de reconnaissance d'objets	29
4.3 Analyse comparative des performances	30
<b>Chapitre 5 : Résumé et Orientations pour Développer le Sujet</b>	<b>30</b>
5.1 Résumé des contributions de la thèse	30
5.2 Études de cas pratiques:	31
5.3 Intégration de techniques d'apprentissage non supervisé:	31

# Introduction

Le domaine du Machine Learning a connu des progrès significatifs, permettant le développement de systèmes intelligents capables de résoudre des problèmes complexes et réels dans divers domaines. Le Modèles de machine learning peuvent être catégorisées en 2 grandes méthodes: l'apprentissage supervisé, où le modèle s'entraîne sur des données étiquetées, et l'apprentissage non supervisé, où le modèle s'entraîne sur des données non étiquetées. Parmi les techniques fondamentales de l'apprentissage supervisé figurent les modèles linéaires (LM) et les perceptrons multicouches (MLP). Ces modèles constituent la pierre angulaire de nombreux algorithmes avancés d'apprentissage automatique et ont des applications étendues allant de la classification aux tâches de régression.

Les modèles linéaires, y compris la régression logistique, constituent la forme la plus simple d'algorithmes d'apprentissage automatique utilisés à la fois pour les tâches de régression et de classification. Ils supposent une relation linéaire entre les variables d'entrée et la variable cible, ce qui les rend efficaces sur le plan informatique et faciles à interpréter. Cependant, leur simplicité peut devenir un défaut lorsqu'il s'agit de données non linéaires.

D'autre part, les perceptrons multicouches (MLP) sont une classe de réseaux de neurones artificiels à action directe constitués de plusieurs couches de nœuds. Chaque nœud, ou neurone, d'un MLP est connecté à chaque nœud de la couche suivante, permettant au réseau de capturer des modèles complexes et des relations non linéaires au sein des données. Les MLP sont devenus un élément fondamental de l'architecture des modèles d'apprentissage profond, contribuant à leur succès dans diverses applications telles que la reconnaissance d'images et de parole.

Récemment, les applications des modèles linéaires (LM) et des réseaux de neurones multicouches (MLP) dans le domaine de la reconnaissance d'objets sont devenues l'un des sujets de recherche importants et prometteurs. Les modèles linéaires, y compris la régression logistique, sont des algorithmes simples mais efficaces pour les problèmes de classification et de régression. Cependant, face à des données non linéaires, le modèle linéaire présente des limites évidentes. Pour surmonter ces limitations, des réseaux de neurones multicouches (MLP) ont été développés et sont devenus un outil puissant pour la reconnaissance d'objets complexes.

Dans ce contexte, notre rapport se concentrera sur la recherche et l'application de méthodes de formation de modèles linéaires (LM) et de réseaux neuronaux multicouches tel que le Perceptron Multicouches (MLP) pour résoudre des problèmes de reconnaissance d'objets tels que les voitures, les motos et les avions. Le rapport approfondira les méthodes d'interpolation, de débruitage et de classification d'objets à travers des modèles linéaires et des réseaux de neurones multicouches.

Le contenu du rapport abordera les problématiques suivantes :

- Recherche sur le Modèle Linéaire (LM).
- Recherche sur le Perceptron multicouches (MLP).
- Collecte et traitement de notre propre ensemble de données, y compris des images converties en matrices.
- Recherche sur la méthode de régression linéaire et la méthode de régression non linéaire.
- Création d'un logiciel pour simuler un système de reconnaissance d'objets avec des données basées sur une combinaison de modèles linéaires et de réseaux neuronaux multicouches.
- À travers la théorie et l'expérimentation, rechercher les caractéristiques et améliorer l'efficacité de cette méthode, en soulignant ses avantages par rapport à d'autres méthodes.

Afin de présenter logiquement le contenu de la recherche, le contenu de la thèse est divisé en 5 chapitres principaux :

- **Chapitre 1** : Recherche sur les modèles linéaires (LM)  
Ce chapitre couvre la définition et les concepts de base des modèles linéaires et des applications courantes. Méthode de régression linéaire.
- **Chapitre 2** : Recherche sur le Perceptron multicouches (MLP)  
Ce chapitre couvrira l'architecture du perceptron multicouches, des neurones, des couches et des fonctions d'activation. Décrire l'algorithme et la méthode de formation du réseau MLP à l'aide de l'algorithme SGD.
- **Chapitre 3** : Collecte et traitement des ensembles de données  
Ce chapitre couvrira les méthodes de collecte de données, les sources de données et les critères de sélection des données. Méthode de

prétraitement des données, comment convertir des images en matrices et créer l'ensemble de données final.

- **Chapitre 4 :** Intégration et simulation de modèles LM et MLP

Ce chapitre couvrira la régression linéaire appliquée à l'ensemble de données, la modélisation et l'évaluation des performances. Application du réseau neuronal multicouche. Conception, formation et évaluation. Implémentation du logiciel, tests et résultats pratiques. Analyse des performances et comparaison avec d'autres méthodes.

- **Chapitre 5 :** Résumé et orientations pour développer le sujet

Dans ce chapitre, je résume ce que j'ai fait dans cette thèse et direction de développement du sujet.

# Chapitre 1 : Recherche sur les Modèles Linéaires (LM)

Ce chapitre couvre la définition et les concepts de base des modèles linéaires et des applications courantes, ainsi que la méthode de régression linéaire.

## 1.1 Définition et concepts fondamentaux du modèle linéaire

### 1.1.1 Le problème général des fonctions à une variable se pose comme suit :

Fonction:  $y = w^*x + b$ .

Où :

$y$  : résultat

$x$  : données d'entrée

$w$  : coefficient

$b$  : erreur

#### Trouver $w$ et $b$ :

Pour prédire  $y$  avec précision, nous devons trouver les meilleures valeurs pour  $w$  et  $b$ . Voici comment nous pouvons les calculer :

#### 1. Méthode des Moindres Carrés

Cette méthode cherche à minimiser l'erreur entre les valeurs prédictives et les valeurs réelles. Imaginez que vous avez plusieurs points sur un graphique, et que vous voulez tracer une droite qui est la plus proche possible de tous ces points.

La formule pour calculer  $w$  et  $b$  est un peu compliquée, mais en gros, elle ajuste les valeurs de  $w$  et  $b$  pour que la droite passe aussi près que possible des points.

### 1.1.2 Application de la matrice 2D en apprentissage automatique:

Lorsque nous avons de nombreuses variables d'entrée, le modèle devient une régression linéaire multiple. Supposons que nous ayons un ensemble de données et que nous souhaitons prédire une variable cible. Généraliser la formule pour plusieurs variables est :

Fonction:  $y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$

Où :

$y$  : résultat

$x$  : Matrice de donnée d'entrée ( $m,n$ ) où  $m$  est le nombre d'échantillons et  $n$  est le nombre de caractéristiques.

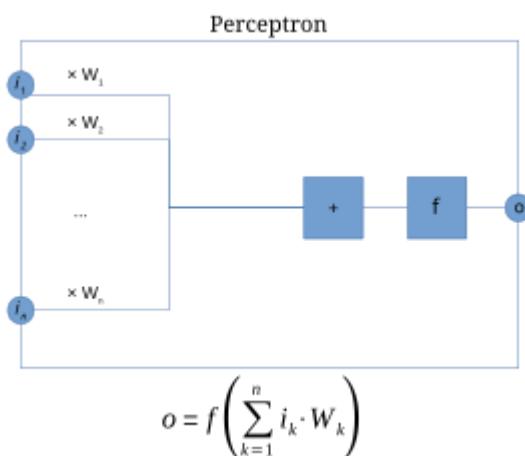
w : Vecteur colonne des coefficients.

b : erreur

### 1.1.2.1 Méthode de Rosenblatt (Perceptron)

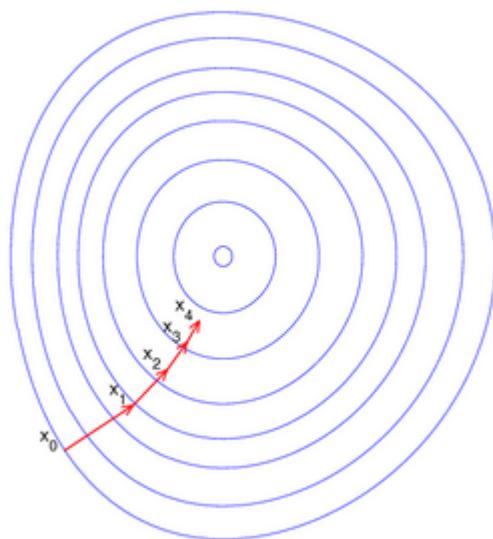
Initialiser les poids (w) et les biais (b) avec des valeurs aléatoires ou nulles.  
pour chaque itération :

- Multipliez  $X_i$  et  $W$  + la valeur du biais pour obtenir une prédiction.
- Calculez la valeur de mise à jour à l'aide de la règle de mise à jour du perceptron :  
$$\text{update\_value} = \text{learning\_rate} * (\text{y}_{\text{expected}} - \text{Signe}(\text{y}_{\text{pred}}))$$
- Met à jour les poids et les biais en fonction de la valeur de mise à jour calculée.  
$$\text{weights} += \text{update\_value} * x_i$$
  
$$\text{bias} += \text{update\_value}$$



### 1.1.2.2 Méthode de Descente de Gradient

La descente de gradient peut être définie comme une méthode d'optimisation pour trouver des paramètres réduisant la cost function.



Etapes:

Partir d'un point  $x_0$  au hasard dans les données

Répéter jusqu'à convergence:

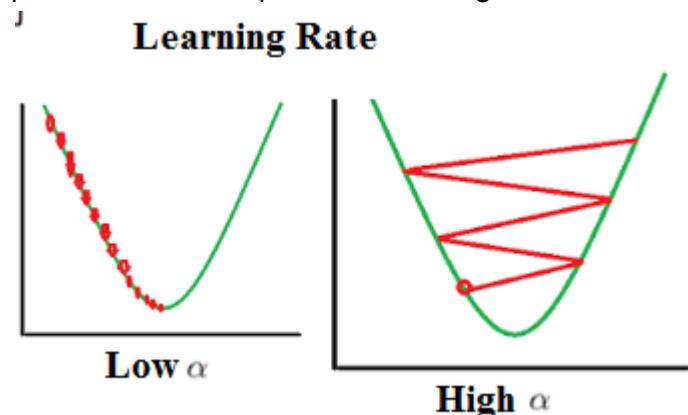
$$x_{t+1} = x_t - n \times \nabla f(x_t)$$

Où:

n: alpha, learning rate

$\nabla f(x_t)$  : gradient

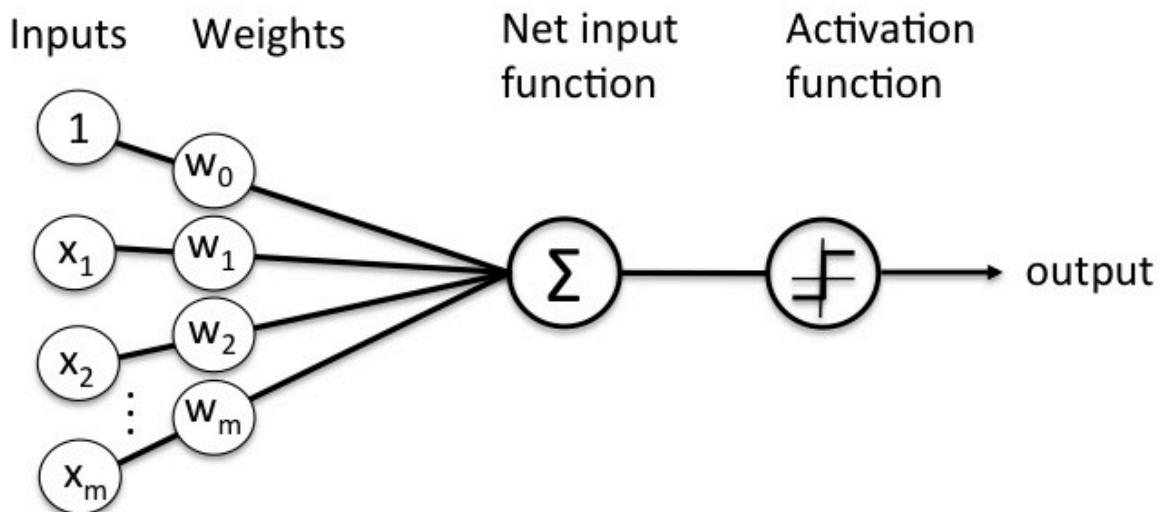
La descente du gradient a pour objectif d'atteindre un minimum. L'objectif peut cependant être inatteignable dépendant de l'hyper paramètre(learning rate choisi), un faible learning rate va amener à une convergence très lente est au contraire un learning rate élevé va potentiellement empêcher la convergence.



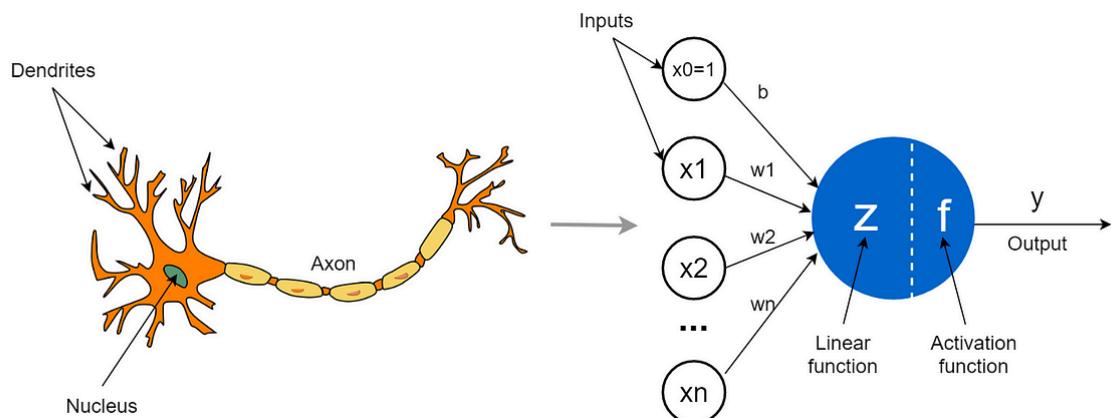
## 1.2 Architecture d'un modèle linéaire

### 1.2.1. Perceptron

Un perceptron est un algorithme, généralement de classification binaire. Crée par Frank Rosenblatt autour des années 1957, il est le modèle le plus simple d'un réseau de neurones, conçu pour effectuer des tâches de classification linéaire.



La structure du perceptron est très fortement inspirée du neurone humain, responsable de la transmission des informations dans notre cerveau



En résumé le perceptron prend les données d'entrées (inputs) + 1 biais et est associé à un poids (weights) le poids représente l'importance d'un input. Après l'association il effectue la somme pondérée et on passe cette somme dans une fonction d'activation, qui va nous donner une prédiction (pour la classification).

## 1.2 Modèle linéaire : régression linéaire

Le but de la régression linéaire est d'expliquer une variable Y à l'aide d'une variable X (resp. plusieurs variables X<sub>1</sub>, ..., X<sub>q</sub>). La variable Y est appelée variable dépendante, ou variable à expliquer.

La relation entre X et Y peut s'écrire sous la forme suivante :

$$y = \beta_0 + \beta_1 x$$

Où B<sub>1</sub> est la pente de la droite de régression et B<sub>0</sub> est l'intercepte (l'ordonnée à l'origine).

Ces coefficients sont définis dans le code comme ceci :

```
#[no_mangle]
pub extern "C" fn create_linear_model() -> *mut LinearModel {

    Box::into_raw(Box::new(x: LinearModel {
        slope: 0.0,
        intercept: 0.0,
    }))
}
```

Pour trouver ces coefficients, nous utilisons l'algorithme de pseudo inverse pour trouver en un coup. Voici la formule de pseudo inverse :

$$W = ((X^T X)^{-1} X^T) Y$$

Où X correspond à la matrice de jeu de donnée d'entraînement et Y est la valeur attendue.

Voici l'endroit où nous calculons ces coefficients :

```
// Calculer la pseudo-inverse ( formule : inverse(X_transposé * X)*X_transposé )

let x_transpose :OMatrix<...> = x_matrix.transpose();
let x_pseudo_inverse :OMatrix<...> = (x_transpose.clone() * &x_matrix)
    .try_inverse() :Option<OMatrix<...>>
    .unwrap() * x_transpose;

// Calculer les coefficients
let coefficients :OMatrix<...> = x_pseudo_inverse * y_vector; // Calcul des coefficients de la régression linéaire

unsafe {
    (*model).intercept = coefficients[0];
    (*model).slope = coefficients[1];
}
```

Suite à cette étape, nous obtenons les coefficients du modèle linéaire. Nous passons à l'étape de la prédiction qui consiste à appliquer formule ci-dessous :

$$y = \beta_0 + \beta_1 x$$

```
#[no_mangle]
pub extern "C" fn predict(model: *const LinearModel, x: c_double) -> c_double {
    // Fonction pour prédire une valeur `y` en fonction de `x` à l'aide du modèle
    unsafe { (*model).slope * x + (*model).intercept } // Calcul de la prédiction `y = pente * x + intercept`
}
```

Appliquons ce modèle à ce cas de test pour comprendre le raisonnement derrière :

Voici les données sur lesquelles nous allons entraîner le modèle :

```
X = np.array([1.0, 2.0], dtype=np.float64)
```

Sur ces données, nous attendons à ce que le modèle prédise ceci :

```
Y = np.array([2.0, 3.0], dtype=np.float64)
```

Après avoir calculé le pseudo-inverse, le modèle trouve les coefficients suivants pour ces données :

B0 : 1

B1 : 1

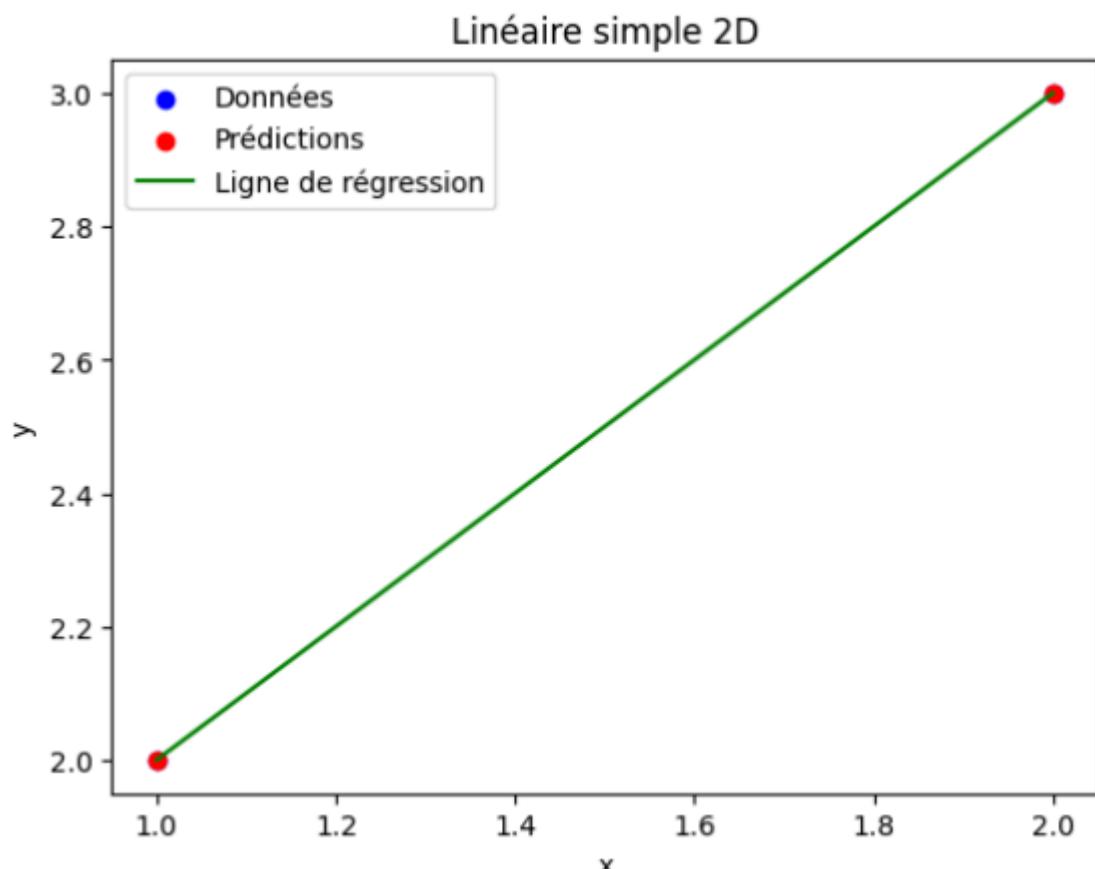
```
C:\Users\nabil\AppData\
Prédictions: [2. 3.]
Intercept: 1
Slope: 1
```

Désormais, passons à l'étape de prédiction. Nous souhaitons prédire sur les mêmes jeux de données ci-dessus.

Donc, pour le premier élément ([1.0, 2.0]) 1(Slope)\*1 + 1(intercept) = 2.

Nous pouvons voir que notre modèle a bien prédit la valeur attendue.

Visualisons le résultat graphiquement :



Notre modèle va itérer ainsi de suite sur toute la matrice X.

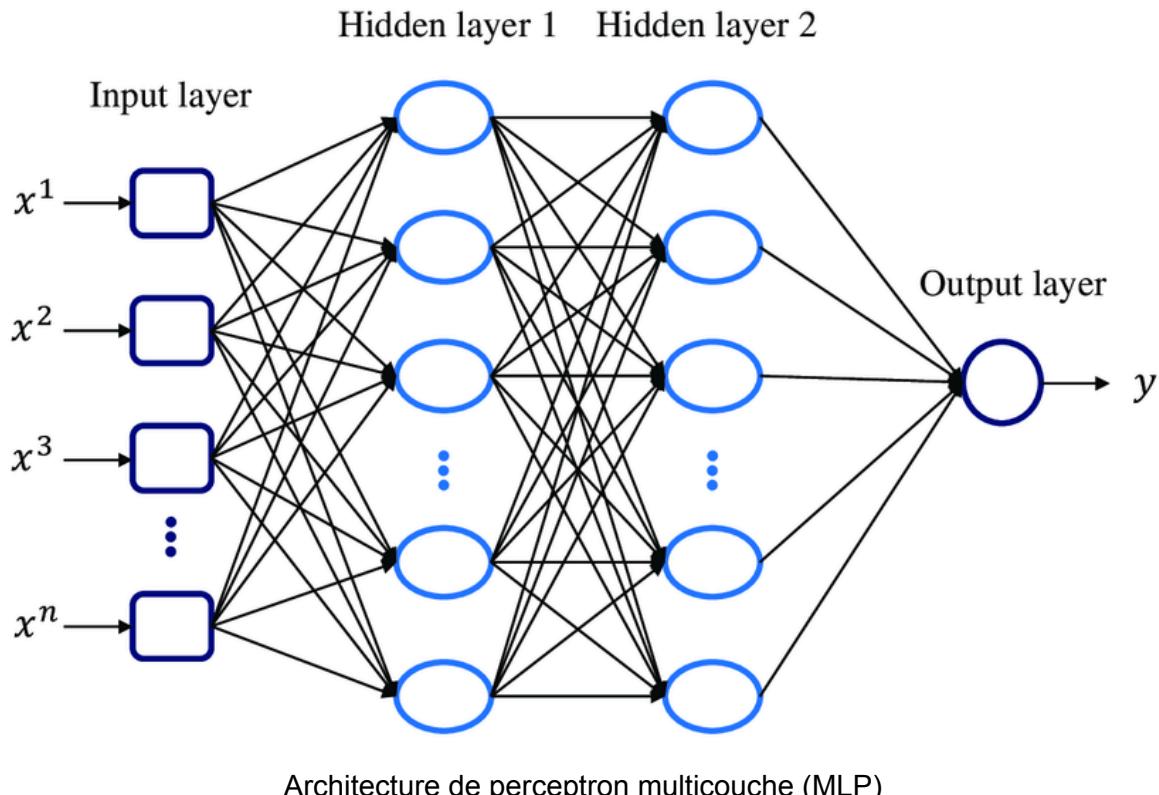
# Chapitre 2 : Recherche sur les Réseaux de Neurones Multicouches (MLP)

Ce chapitre examine plus en profondeur les réseaux de neurones multicouches (MLP), y compris leurs concepts, leurs structures et les algorithmes d'apprentissage associés.

## 2.1. Réseau de neurones artificiels et architecture de réseau MLP

Le réseau de neurones artificiels, appelé en abrégé réseau de neurones, est un modèle mathématique construit sur la base de réseaux de neurones biologiques constitués d'un grand nombre d'éléments (appelés neurones) connectés les uns aux autres via des connexions (appelées poids articulaires) fonctionnant comme un tout. un tout unifié pour résoudre des problèmes spécifiques tels que la reconnaissance de formes, la classification des données, etc. grâce à un processus d'apprentissage à partir d'un ensemble d'échantillons de formation.

Un réseau MLP général est un réseau à  $n$  couches ( $n \geq 2$ ) qui comprend une couche d'entrée, une couche de sortie et une ou plusieurs couches cachées.



## 2.2 Architecture des réseaux de neurones multicouches

### Neurones, couches, et fonctions d'activation:

Le réseau neuronal multicouche (MLP) est composé de plusieurs couches de neurones :

**Couche d'entrée:** les neurones reçoivent les signaux d'entrée et les traitent (calculer la somme pondérée, envoyer à la fonction de transfert)

**Couches cachées:** situées entre les couches d'entrée et de sortie, elles permettent au réseau de capturer des relations non linéaires complexes. Il peut y avoir une ou plusieurs couches cachées.

**Couches de sortie:** générez le résultat final du réseau. Le nombre de neurones dans cette couche dépend du type de tâche (classification, régression, etc.).

## 2.3 Algorithmes d'apprentissage pour les MLP

### Propagation avant et rétropropagation:

L'apprentissage dans un MLP se déroule généralement en deux phases : la propagation avant (Feedforward) et la rétropropagation de l'erreur (backpropagation).

#### Propagation avant :

- **Calcul d'entrée caché:** les données d'entrée sont multipliées par les poids de la première couche et le biais est ajouté.
- **Application de la fonction d'activation:** La fonction ReLU est appliquée à l'entrée cachée.
- **Calculez l'entrée finale:** la sortie de la couche cachée est multipliée par les poids de la deuxième couche et le biais est ajouté.
- **Application de la fonction d'activation de sortie:** la fonction Softmax est appliquée pour obtenir la probabilité de sortie.

```
fn forward(&mut self, x: &Array2<Float>) -> Array2<Float> {
    let hidden_input : ArrayBase<OwnedRepr<Float>, Ix2> = x.dot(&self.weights_input_hidden) + &self.bias_hidden;
    let hidden_output : Array2<Float> = MLP::relu(&hidden_input);
    let final_input : ArrayBase<OwnedRepr<Float>, Ix2> = hidden_output.dot(&self.weights_hidden_output) + &self.bias_output;
    MLP::softmax(&final_input)
}
```

#### Rétropropagation de l'erreur:

- **Calcul d'erreur:** la différence entre la prédiction du réseau et la valeur réelle (sortie souhaitée) est calculée.
- **Calcul de l'erreur et du biais dans les couches cachées:** pour chaque couche cachée, l'erreur de sortie sera propagée pour calculer l'erreur et la correction nécessaire.
- **Mise à jour des pondérations et des biais:** les pondérations et les biais sont mis à jour à l'aide de l'algorithme de descente de gradient stochastique (SGD).

- **Appliquer la régularisation L2:** pour éviter le surajustement, appliquez la régularisation L2. Cela punit les poids élevés en réduisant légèrement leur valeur.

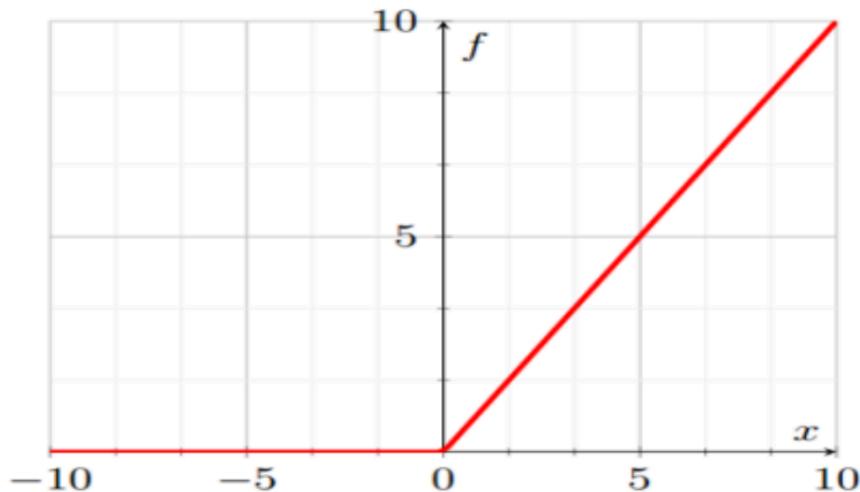
```
fn backward(&mut self, x: &Array2<Float>, y: &Array2<Float>, output: &Array2<Float>) {
    let output_error = output - y;
    let hidden_output = MLP::relu(&(x.dot(&self.weights_input_hidden) + &self.bias_hidden));
    let hidden_error = output_error.dot(&self.weights_hidden_output.t());
    let hidden_delta = hidden_error * MLP::relu_derivative(&hidden_output);

    self.weights_hidden_output -= &hidden_output.t().dot(&output_error) * self.learning_rate;
    self.bias_output -= output_error.sum_axis(Axis(0)) * self.learning_rate;

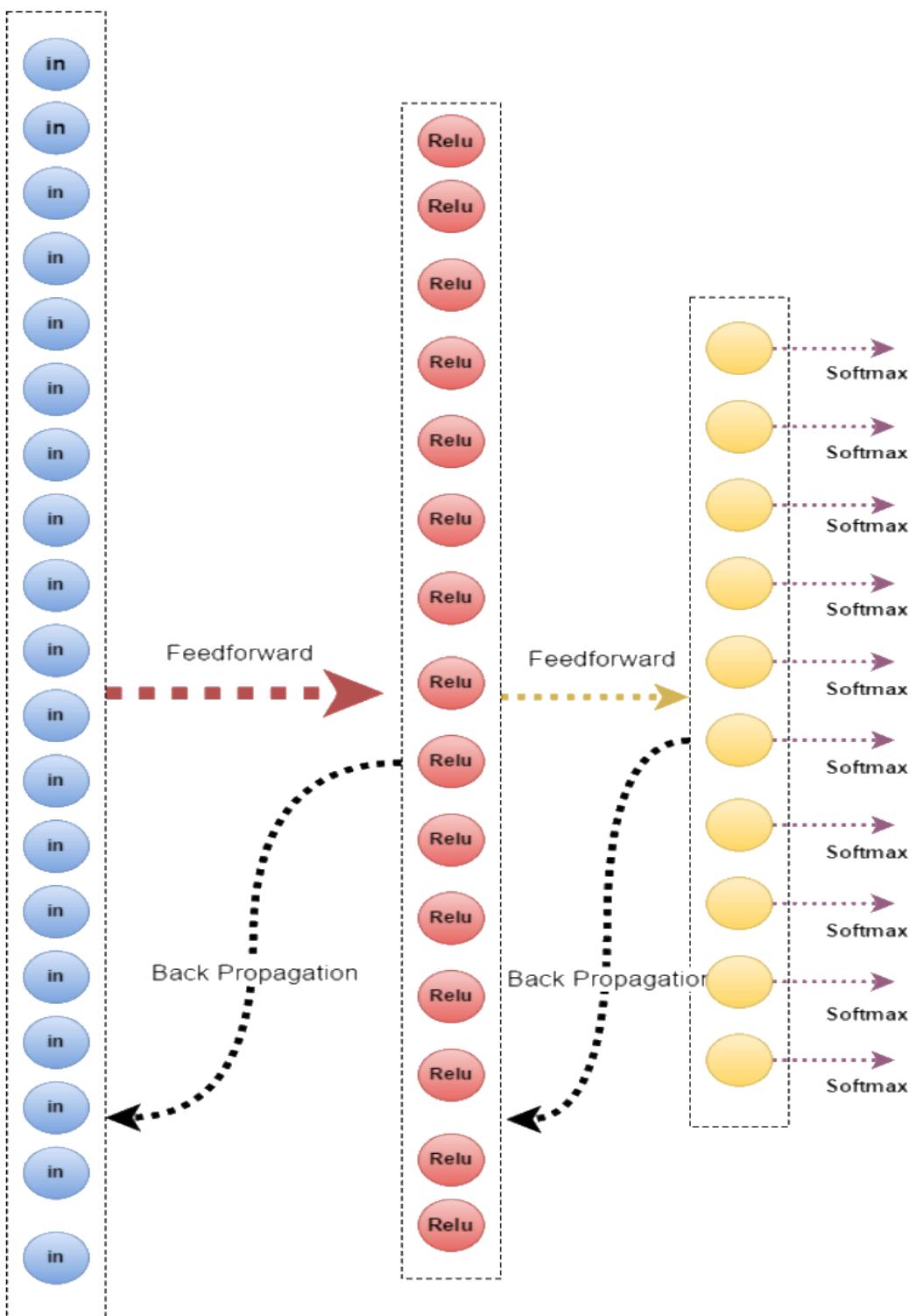
    self.weights_input_hidden -= &x.t().dot(&hidden_delta) * self.learning_rate;
    self.bias_hidden -= hidden_delta.sum_axis(Axis(0)) * self.learning_rate;

    let l2_reg :f64 = 0.01;
    self.weights_hidden_output -= &self.weights_hidden_output * l2_reg;
    self.weights_input_hidden -= &self.weights_input_hidden * l2_reg;
}
```

### Fonction ReLU:



$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$



Réseau PMC appliqué à la reconnaissance des numéros manuscrits (MNIST)

## 2.4 Expérimentation avec l'ensemble de données MNIST.

Pour garantir le fonctionnement efficace du modèle MLP, il est nécessaire de garantir que les données utilisées pour les tests constituent un ensemble de données stable et que la quantité de données est suffisante pour entraîner le modèle. Le MNIST est donc le choix pour déterminer si le modèle MLP fonctionne bien ou non.

Ici, l'ensemble de données utilisé est le MNIST, qui comprend une variété de données manuscrites populaires pour former un certain nombre de modèles d'apprentissage automatique. Il existe 10 images manuscrites avec 10 chiffres différents, le nombre de calques par défaut est donc de 10.

Les images de chiffres manuscrits sont représentées par des tableaux 2D et la taille initiale des données est de  $28 \times 28$  pour chaque image ( $28 \times 28$  pixels). Les images 2D sont ensuite aplatis et représentées à l'aide de vecteurs. Chaque image 2D est convertie en un vecteur 1D de dimensions  $[1, 28 \times 28] = [1, 784]$ .



### Formation sur modèle:

Les données comprennent 60 000 images, divisées en environ 50 000 images pour la formation et 10 000 images pour les tests sur modèle.

Ensuite, l'image est renvoyée dans la plage  $[0,1]$  (c'est-à-dire  $I(x,y) = I(x,y)/255$  )

Après avoir divisé les deux ensembles de formation et testé le modèle, les données sont incluses dans la formation du modèle.

Cette étape réalise le processus de construction du modèle, de formation et d'exécution de la classification, grâce à l'utilisation d'une couche cachée de 512 neurones chacune. Définissez le nombre maximum d'itérations sur 100, le lot sur 32 et le taux d'apprentissage sur 0,1.

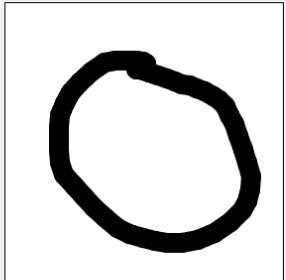
## Évaluation du modèle Multilayer Perceptron (MLP):

Évaluez le modèle en estimant la précision moyenne des données après chaque session de formation en comparant les étiquettes de formation et les résultats de l'ensemble de données à tester. Calculez ensuite la perte pendant l'entraînement.

Si les résultats de la formation et des tests sont bons, le modèle est mis en pratique.

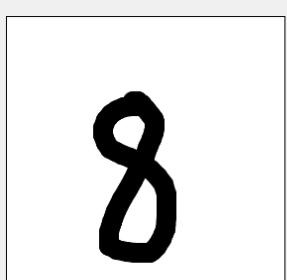
### Comparer les résultats obtenus :

**Draw a Digit**



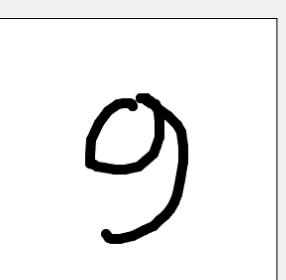
Prediction: 0

**Draw a Digit**



Prediction: 8

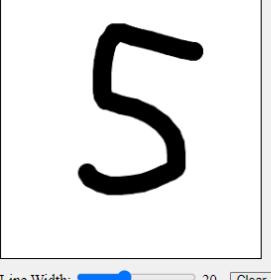
**Draw a Digit**



Prediction: 9

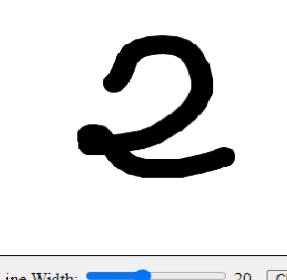
**Draw a Digit**



Line Width:  20

Prediction: 6

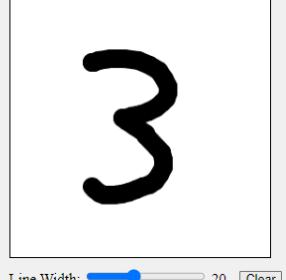
**Draw a Digit**



Line Width:  20

Prediction: 2

**Draw a Digit**



Line Width:  20

Prediction: 3

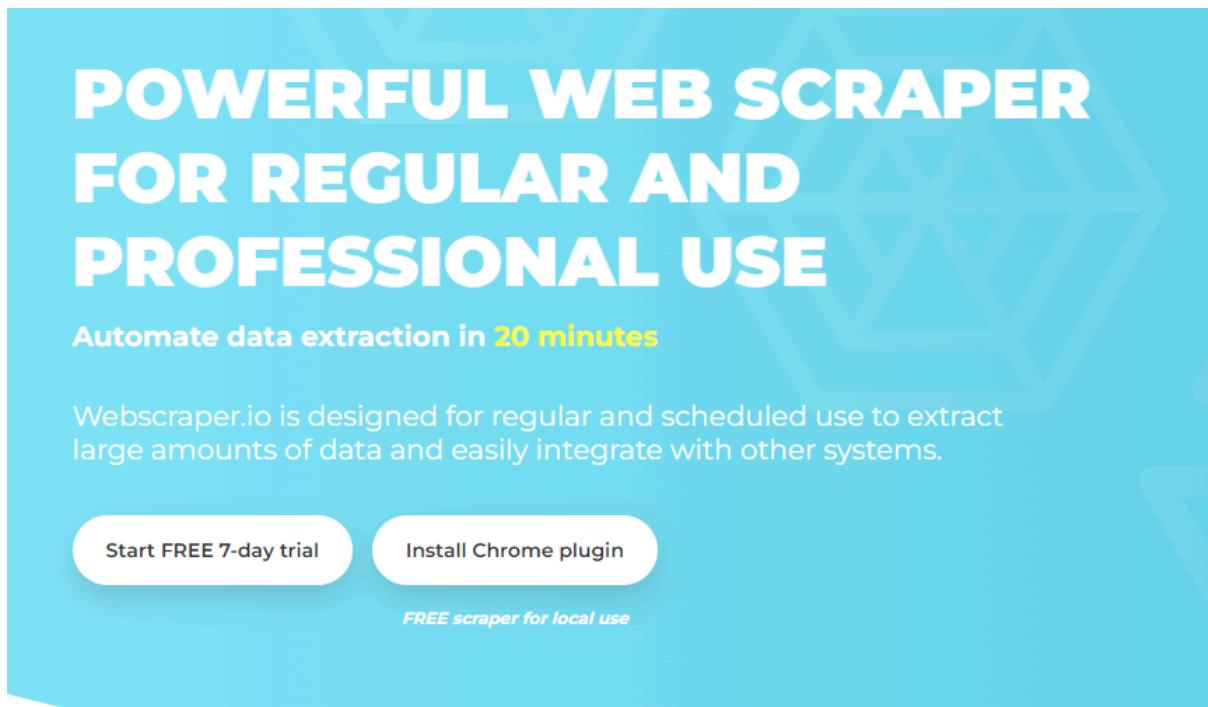
# Chapitre 3 : Collecte et Traitement de l'Ensemble de Données

Ce chapitre couvre les méthodes de collecte de données, les sources de données et les critères de sélection des données, ainsi que les méthodes de prétraitement des données, comment convertir des images en matrices et créer l'ensemble de données final.

## 3.1 Méthodologie de collecte des données

Afin de collecter assez d'échantillons pour notre problématique de classification ( initialement : Berline, Voiture de sport, SUV, actuellement: Avion, Moto, Voiture), nous avons utilisé divers outils.

### 3.1.1 Plugin sur navigateur: WebScraper



Webscraper est un outil de scraping permettant de récolter les liens d'images après certaines configurations:

La démonstration suivante a été réalisée sur un site autorisant le scraping et fait pour le scraping:

- En ouvrant les outils développeur après avoir installé le plugin, vous remarquerez un nouvel onglet qui s'est ajouté, qui est notre Web Scraper

ID	Domain	
airliners	airliners.net	<button>Delete</button>
mad4wheelz	mad4wheelz.com	<button>Delete</button>
pixels	pixels.com	<button>Delete</button>
planespotter	planespotters.net	<button>Delete</button>

- En se rendant sur l'onglet nous pourrons créer un "sitemap"(qui peut être considéré comme la carte du site que nous voulons scrap)
- Une fois les données du site entré(url, nom) nous pourrons selectionner les informations qui nous intéressent

The screenshot shows a web-based application for scraping quotes. It displays three cards, each containing a quote, its author, and a list of tags. Below the cards is a green button labeled 'Done selecting'.

ID	Selector	type	Multiple	Parent selectors	Actions
Add new selector					

**Card 1:**  
"The person, be it gentleman or lady, who has not pleasure in a good novel, must be intolerably stupid."  
by Jane Austen (about)  
Tags: [illiteracy] [books] [classic] [humor]

**Card 2:**  
"Imperfection is beauty, madness is genius and it's better to be absolutely ridiculous than absolutely boring."  
by Marilyn Monroe (about)  
Tags: [be-yourself] [inspirational]

**Card 3:**  
"Try not to become a man of success. Rather become a man of value."  
by Albert Einstein (about)  
Tags: [schema.org/CreativeWork] > [success] [value]

C S Done selecting

me's language | Switch DevTools to French | Don't show again

L'ajout des éléments similaires se feront automatiquement.

- Lorsque tout est bon nous n'aurons qu'à cliquer sur scrape this website et tout est bon

"The person, be it gentleman or lady, who has not pleasure in a good novel, must be intolerably stupid."
"Imperfection is beauty, madness is genius and it's better to be absolutely ridiculous than absolutely boring."
"Try not to become a man of success. Rather become a man of value."
"It is better to be hated for what you are than to be loved for what you are not."

Les données sont exportables en csv ou xlsx

Etant donné que les données sont en tableau, nous avons créés un script python pour faciliter le téléchargement des images

```
import os
import pandas as pd
import requests
from PIL import Image
from io import BytesIO

df = pd.read_csv('planes.csv')

saving_folder = './planes'

os.makedirs(saving_folder, exist_ok=True)

for index, row in df.iterrows():

    image_url = row[0]

    try:

        response = requests.get(f'{image_url}')

        if response.status_code == 200:

            img = Image.open(BytesIO(response.content))

            img_name = f'{index}.jpg'
            img_path = os.path.join(saving_folder, img_name)
            img.save(img_path)
            print(f'Image {index} downloaded successfully.')

        else:

            print(f'Failed to download image {index}. Status code: {response.status_code}')

    except Exception as e:
```

```
print(f"Error downloading image {index}: {e}")
print("All images downloaded.")
```

## 3.2 Prétraitement des données

Avant la conversion des images en données, nous avons trié manuellement les images qui étaient jugées comme mauvais



*Image retirée car elle n'est pas homogène avec le reste des images*

## 3.3 Conversion des images en matrices

Chaque image a été ensuite converti en ndarray(via numpy, initialement 50x50) et aplati(flattened) pour obtenir une ligne de 2500 colonnes + 1 colonne pour le target, allant de 0 à 2(0 => berline, 1 => sportscar , 2=>suv)

```
import os
import numpy as np
import pandas as pd
from PIL import Image

main_folder_path = './images'

target_size = (28, 28)

flattened_images = []
labels = []

label_mapping = {'bike': 0, 'car': 1, 'planes': 2}

for folder_name in ['bike', 'car', 'planes']:
```

```

folder_path = os.path.join(main_folder_path, folder_name)

for image_name in os.listdir(folder_path):
    image_path = os.path.join(folder_path, image_name)

    # Check if the file is an image
    if not image_path.lower().endswith('.png', '.jpg',
'.jpeg', '.bmp', '.gif'):
        print(f"Skipping non-image file: {image_path}")
        continue

    try:

        with Image.open(image_path) as img:

            gray_img = img.convert('L')

            resized_img = gray_img.resize(target_size)

            flattened_img =
np.array(resized_img).flatten()

            flattened_images.append(flattened_img)

            labels.append(label_mapping[folder_name])

    except Exception as e:
        print(f"Error processing image {image_path}: {e}")

flattened_images = np.array(flattened_images)
labels = np.array(labels)

df_images = pd.DataFrame(flattened_images)
df_labels = pd.DataFrame(labels, columns=['target'])

df = pd.concat([df_images, df_labels], axis=1)

df.to_csv('twentyeight_dataset.csv', index=False)

print("Dataset created and saved to CSV successfully.")

```

## 3.4 Construction de l'ensemble de données final

### 3.4.1 Berline, SUV, Voiture de Sport



Ce dataset étiqueté de 0 à 2(0 => berline, 1 => sportscar , 2=>suv) contient 600 images de train par classe, et 200 images de test. Chaque image est redimensionné en 50x50 (2500 + 1 target quand aplati), pour donner au final un dataset train de (1800,2500) et test (600, 2500). Les chaque valeur d'une colonne est entre la range [0; 255] étant donné que l'image est transformée en niveau de gris. Lors de l'entraînement et des prédictions le dataset est normalisé pour que les valeurs soient entre 0 et 1

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

### 3.4.2 Avion, Moto, Voiture



Ce dataset étiqueté de 0 à 2(0 => avion, 1 => voiture , 2=>moto) contient 2000 images par classe au total. Chaque image est redimensionné en 28x28 (784 + 1 target quand aplati), pour donner au final un dataset de (6000,784)

## Chapitre 4 : Intégration et simulation de modèles LM et MLP

Ce chapitre couvre la régression linéaire appliquée aux ensembles de données collectées (voitures, motos, avions), la modélisation et l'évaluation des performances, ainsi que l'application, la conception, la formation et l'évaluation des réseaux neuronaux multicouches. Il comprend également la mise en œuvre du modèle, les tests et les résultats réels, ainsi que l'analyse des performances et la comparaison avec d'autres méthodes. Le niveau de gris est appliqué sur ce dataset, ainsi que la normalisation(min - max) lors de l'entraînement.

### 4.1 Application du réseau neuronal multicouche

Ensuite le processus de formation du modèle MLP, en gardant inchangé le traitement des données d'image d'entrée. L'image est redimensionnée à 28x28 pixels, convertie en noir et blanc (0-255) puis ramenée à la plage [0,1] en divisant les pixels par 255.

Commencez par initialiser le modèle, le modèle comprend 3 couches : Couche d'entrée, couche cachée et couche de sortie. Puisque les données d'entrée ont 28x28 pixels, la couche d'entrée aura 784 neurones. Ensuite, pour garantir une grande précision des données d'entraînement, la couche cachée a été testée avec respectivement 128, 256 et 514 neurones. Et enfin, comme les données sont divisées en 3 types (avion, voiture, moto), la dernière couche possède 3 neurones, et utilise softmax pour augmenter le déterminisme lors du renvoi de la valeur finale.

Module	Time consume (s)	Loss last epoch	Accuracy last epoch (%)	Accuracy on test dataset (%)
MLP LR = 0.001 Epochs = 20 Batch = 32	15.0	0.0282	99.75	88.58
MLP LR = 0.001 Epochs = 40 Batch = 32	31.0	0.0057	100.00	89.66
MLP LR = 0.001 Epochs = 20 Batch = 60	12.5	0.0316	99.64	87.91
MLP LR = 0.001 Epochs = 40 Batch = 60	24.1	0.0063	99.98	90.17
MLP LR = 0.001 Epochs = 30 Batch = 128	15.3	0.0292	99.67	89.67
MLP LR = 0.001 Epochs = 40 Batch = 256	19.9	0.1731	93.71	89.67

```
Epoch 21, Loss: 0.1364496923553202, Accuracy: 95.708%
Epoch 22, Loss: 0.13244466103904132, Accuracy: 95.896%
Epoch 23, Loss: 0.10968415527214134, Accuracy: 96.625%
Epoch 24, Loss: 0.14651329999115184, Accuracy: 95.125%
Epoch 25, Loss: 0.07977582166612521, Accuracy: 97.854%
Epoch 26, Loss: 0.7836760668249697, Accuracy: 88.292%
Epoch 27, Loss: 0.4091384483809053, Accuracy: 91.021%
Epoch 28, Loss: 0.09738152121189392, Accuracy: 97.417%
Epoch 29, Loss: 0.07780035142099025, Accuracy: 98.146%
```

Surapprentissage pour le réglage : époques = 40, batch\_size = 256

## 4.2 Simulation du système de reconnaissance d'objets

### Upload an Image

car\_background\_1.jpg



Prediction: plane

### Upload an Image

bike\_background\_2.jpg



Prediction: plane

### Upload an Image

Mahindra Scorpio\_399.jpg



Prediction: bike

Select Model:

### Upload an Image

Mahindra Scorpio\_395.jpg



Prediction: car

Select Model:

### Upload an Image

bike\_16.jpg



Prediction: bike

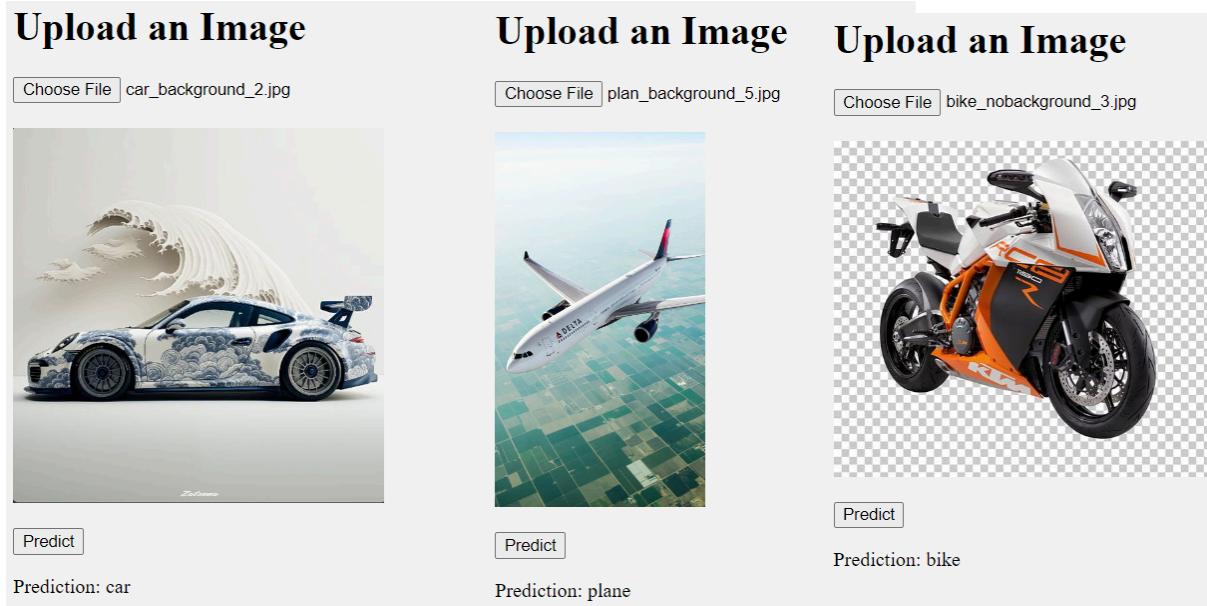
### Upload an Image

Audi\_221.jpg



Prediction: car

Select Model:



## 4.3 Analyse comparative des performances

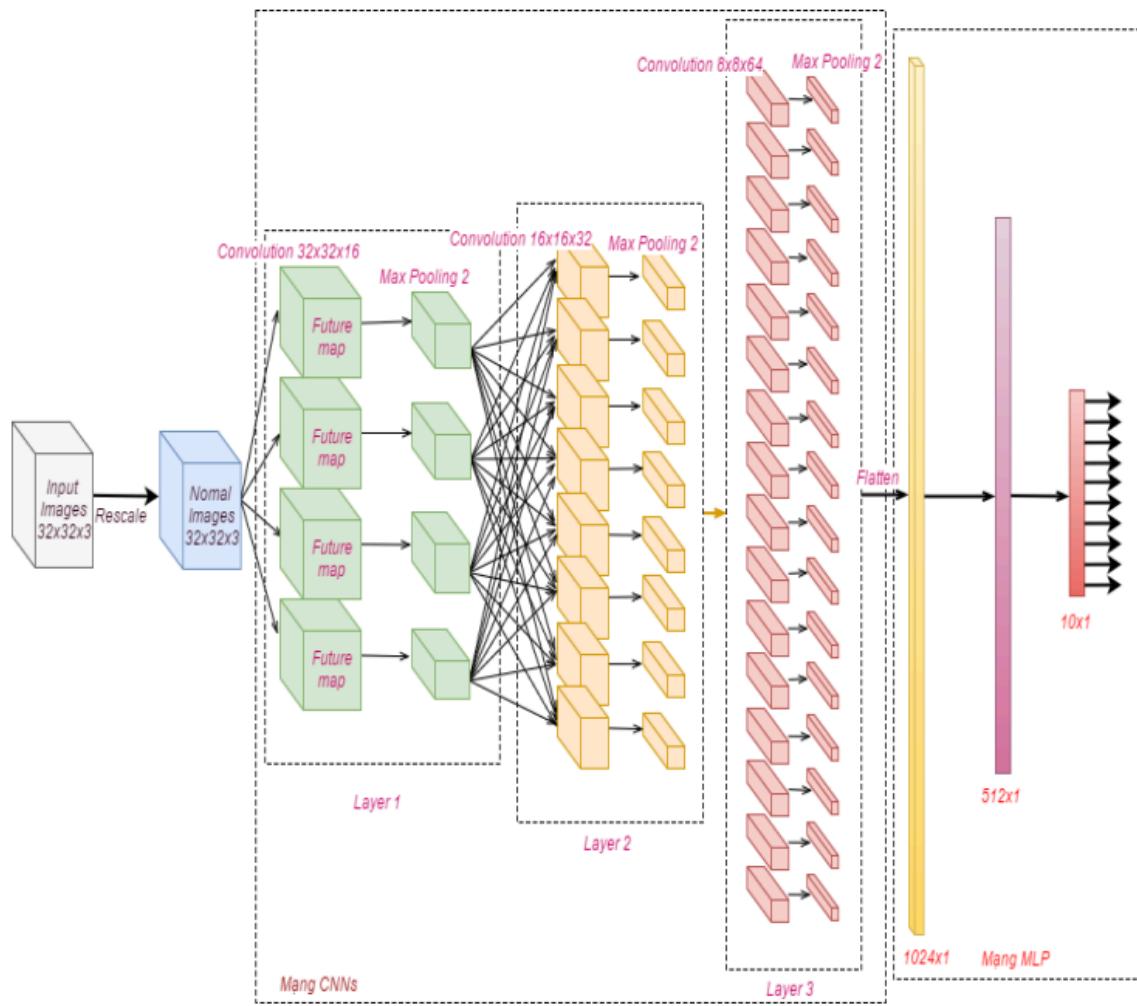
Model	Train Accuracy	Test Accuracy	Learning Rate	Epochs	L2 Regularization	Training time
LM	NaN	81.17%	0.01	1000	NaN	28s
MLP	100%	89.59%	0.001	40	0.01	40s

## Chapitre 5 : Résumé et Orientations pour Développer le Sujet

Ce chapitre résume les travaux effectués dans cette thèse et présente les directions de développement du sujet.

### 5.1 Résumé des contributions de la thèse

Étudier et expérimenter avec des architectures de réseaux plus avancées, telles que les réseaux neuronaux convolutifs (CNN) pour améliorer encore la précision de la reconnaissance d'objets.



## 5.2 Études de cas pratiques:

Appliquer les modèles développés à des études de cas réels pour évaluer leur performance dans des environnements pratiques et identifier des domaines d'amélioration.

## 5.3 Intégration de techniques d'apprentissage non supervisé:

Explorer l'intégration de techniques d'apprentissage non supervisé, telles que le clustering et l'analyse en composantes principales, pour prétraiter les données et améliorer la performance des modèles supervisés.