

# La Base de données MNIST

CNN ou SVM

Chaoui Sokayna



Encadré par Monsieur Boubchir

2022-2023

# Table des matières

|   |           |
|---|-----------|
| <b>Sommaire exécutif</b>                              | <b>2</b>  |
| <b>Introduction</b>                                   | <b>3</b>  |
| <b>Partie théorique</b>                               | <b>4</b>  |
| <b>1 Présentation du jeu de données</b>               | <b>4</b>  |
| <b>2 Les réseaux de neurones convolutifs ou CNN</b>   | <b>6</b>  |
| 2.1 Le Deep Learning . . . . .                        | 6         |
| 2.2 L'algorithme . . . . .                            | 6         |
| <b>3 La machine à support de vecteurs ou SVM</b>      | <b>9</b>  |
| 3.1 Le Machine Learning . . . . .                     | 9         |
| 3.2 L'algorithme . . . . .                            | 9         |
| <b>4 Comparaison</b>                                  | <b>12</b> |
| 4.1 Deep Learning vs Machine Learning . . . . .       | 12        |
| 4.2 CNN ou SVM . . . . .                              | 12        |
| <b>Partie pratique</b>                                | <b>14</b> |
| <b>5 Préparation du jeu de données</b>                | <b>14</b> |
| 5.1 Importation des librairies et fonctions . . . . . | 14        |
| 5.2 Visualisation des images et label . . . . .       | 15        |
| 5.3 Preprocessing pour le CNN et le SVM . . . . .     | 16        |
| <b>6 Les réseaux de neurones convolutifs</b>          | <b>17</b> |
| <b>7 La machine à support de vecteurs</b>             | <b>19</b> |
| <b>8 Comparaison</b>                                  | <b>21</b> |
| <b>Conclusion</b>                                     | <b>22</b> |
| <b>Références</b>                                     | <b>23</b> |
| <b>Appendix</b>                                       | <b>24</b> |

## Sommaire exécutif

De nos jours, les données sont devenues une denrée plus qu'importante, elles nous permettent de trouver une explication à différents phénomènes ou comportements grâce à des algorithmes visant à prédire des résultats par régression ou classification. On utilise ces mêmes données afin de reconnaître une image ou encore afin d'assimiler une population à certaines catégories tel que "être atteint du diabète". Ces données servent de lignes directrices pour l'analyse faite ultérieurement.

Pour réaliser cette analyse, on retrouve deux grands modes d'apprentissage : le Machine Learning (ou encore l'apprentissage automatique) et le Deep Learning (aussi appelé l'apprentissage "profond"). Ces deux modes ont tout les deux le même objectif mais n'ont pas les mêmes caractéristiques ni le même mode d'emploi. Dans chacun de ces modes, on retrouve différents algorithmes tel que les forêts aléatoires, l'arbre de décision ou encore la machine à support de vecteurs en Machine Learning ; mais aussi les réseaux de neurones convolutifs et les réseaux neuronaux récurrents en Deep Learning.

Chacun de ces algorithmes s'adaptent à différentes situations. Certains sont plus adaptés pour la reconnaissance d'images, d'autres pour prédire une catégorie ou encore une valeur. On peut utiliser plusieurs algorithmes pour prédire la même chose mais il y en a toujours un qui se démarquera des autres grâce à ses performances ainsi que son temps d'exécution. Ce sont aussi des données que l'on peut analyser et ressortir afin de comparer nos différents algorithmes.

Grâce aux données trouvées précédemment, on peut maintenant trouver l'algorithme le plus pratique selon la situation dans laquelle nous nous trouvons sans même avoir à comparer cela avec d'autres algorithmes. Malgré tout, il reste intéressant de connaître les performances d'autres algorithmes dans cette même situation. Elles sont parfois meilleures qu'on ne le pense et elles nous permettent d'avoir un regard neuf face à l'algorithme qu'on aura utilisé.

# Introduction

Afin de réaliser cette étude, j'ai décidé de choisir la base de données MNIST (aussi appelée Modified ou Mixed National Institute of Standards and Technology). Il s'agit d'une base de données répertoriant des chiffres écrits à la main de différentes manières comme présenté ci-dessous :

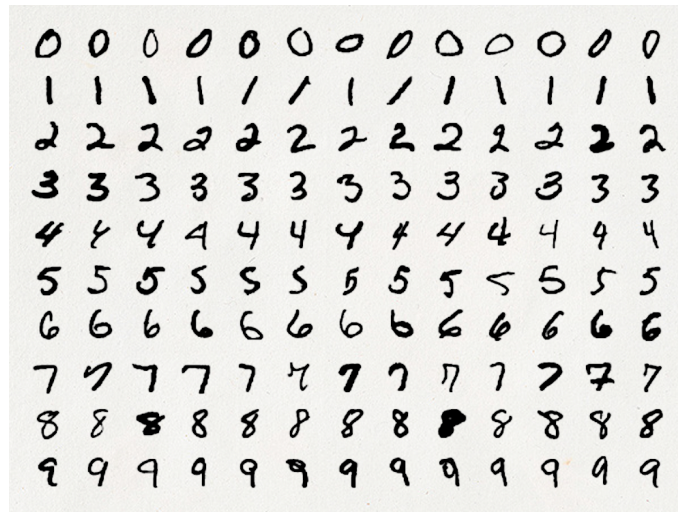


FIGURE 1 – Base de données de chiffres écrits à la main

Dans la base de données que j'ai utilisé, pour chaque ligne du fichier d'entraînement, on retrouve le chiffre correspondant à l'image ainsi que les 784 pixels représentés par celle-ci. Dans le fichier test, on retrouve la même chose mais sans la colonne donnant le chiffre correspondant.

Le but de cette étude est de comparer deux algorithmes adaptés pour reconnaître le chiffre correspondant aux pixels de l'image. Ces deux algorithmes, choisis par moi-même, sont les réseaux de neurones convolutifs (appelés CNN) que l'on retrouve en Deep Learning ainsi que la machine à support de vecteurs (appelé SVM) que l'on retrouve en Machine Learning. Cette étude vise donc à comparer ces deux modes d'apprentissage grâce aux deux algorithmes choisis.

Afin de réaliser au mieux cette étude, je comparerai dans un premier temps les deux algorithmes grâce à leurs différentes caractéristiques puis je comparerai leurs performances lors de leur utilisation sur la base de données MNIST.

# Partie théorique

Afin de réaliser cette étude, il faut commencer par comprendre notre jeu de données ainsi que nos deux algorithmes. Suite à celà, la comparaison de ces deux modèles pourra être faite.

## 1 Présentation du jeu de données

Le jeu de données choisis pour cette étude est séparé en deux fichiers : celui d'entraînement qui répertorie le chiffre sur l'image ainsi que ses 784 pixels, ce qui nous donne 785 colonnes pour 42000 lignes ; et celui de test qui lui ne répertorie que les pixels de chaque image (soit 784) pour 28000 lignes. Pour exemple voici à quoi ressemble les premières lignes et colonnes du fichier d'entraînement :

| label | pixel0 | pixel1 | pixel2 | pixel3 | pixel4 | pixel5 | pixel6 | pixel7 |
|-------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1     | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      |
| 0     | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      |
| 1     | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      |
| 4     | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      |
| 0     | 0      | 0      | 0      | 0      | 0      | 0      | 0      | 0      |

FIGURE 2 – Fichier d'entraînement

Chaque image des fichiers a une hauteur et une longueur de 28 pixels, d'où les 784 pixels (28x28). Chaque pixel represente la luminosité de l'emplacement où il se trouve, plus le pixel est grand plus il est sombre, ce qui nous donne des images en nuance de gris.

La colonne label correspond au chiffre représenté sur l'image. Chaque colonne de pixels de l'ensemble d'entraînement a un nom comme pixelx, où x est un entier compris entre 0 et 783, inclus. Pour localiser ce pixel sur l'image, supposons que nous ayons décomposé x comme  $x = i * 28 + j$ , où i et j sont des entiers compris entre 0 et 27, inclus. Ensuite, pixelx est situé sur la ligne i et la colonne j d'une matrice 28 x 28, (indexation par zéro). Par exemple, pixel31 indique le pixel qui se trouve dans la quatrième colonne à partir de la gauche, et la deuxième ligne à partir du haut.

Visuellement, en omettant la partie "pixel" et en ne gardant que son numéro, la matrice a la forme suivante :

|     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|
| 000 | 001 | 002 | 003 | ... | 026 | 027 |
| 028 | 029 | 030 | 031 | ... | 054 | 055 |
| 056 | 057 | 058 | 059 | ... | 082 | 083 |
|     |     |     |     | ... |     |     |
| 728 | 729 | 730 | 731 | ... | 754 | 755 |
| 756 | 757 | 758 | 759 | ... | 782 | 783 |

FIGURE 3 – Matrice 28x28

Cette matrice représente l'image et est facilement visualisable avec le logiciel Python.

On retrouve les mêmes caractéristiques dans notre fichier test, la seule différence étant qu'il ne contient pas la colonne "label" donnant le chiffre représenté sur l'image.

Je trouve ce jeu de données particulièrement intéressant. Depuis sa sortie en 1999, il a servi de base aux algorithmes de classification comparative et sa fiabilité a permis à différents chercheurs de vérifier les nouveaux algorithmes qu'ils ont créés. Il sera tout autant utile à mon analyse des deux algorithmes ainsi que leur mode d'apprentissage, d'où le fait du choix de ce jeu de données. D'autant plus, les deux fichiers contiennent tout deux plusieurs milliers de données et cela fut donc un vrai défi pour moi de travailler sur un jeu de données si volumineux.

## 2 Les réseaux de neurones convolutifs ou CNN

### 2.1 Le Deep Learning

Afin de comprendre ce qu'est le CNN, il faut d'abord savoir ce qu'est le Deep Learning (ou apprentissage "profond"). Le Deep Learning est une technique de machine learning reposant sur le modèle des réseaux neurones : des dizaines voire des centaines de couches de neurones sont empilées pour apporter une plus grande complexité à l'établissement des règles.

Il est capable d'analyser des données non structurées tels que des images, du texte etc. Sa création s'inspire des neurones du cerveau humain qui assimile les données et sépare les différentes caractéristiques pour mieux les comprendre. Le Deep Learning est donc un ensemble d'algorithmes mimant les actions du cerveau humain grâce à un réseau de neurones artificiels. Ici, l'algorithme utilisé sera le CNN (Convolutional Neural Network) qui est l'un des algorithmes de Deep Learning le plus performant.

### 2.2 L'algorithme

Pour cet étude, le choix du CNN était plus qu'évident, celui-ci étant justement utilisé pour la reconnaissance d'images. Cet algorithme est aussi efficace pour l'analyse sémantique, la modélisation de phrase, la classification et la traduction.

Ce réseau convolutifs se séparent en plusieurs couches :

1. La convolution : elle vient détecter les motifs visuels, bords et autres caractéristiques d'une image en se déplaçant de la gauche vers la droite et en prenant de petites parties de l'image une à une. C'est grâce à un calcul convolutif que l'algorithme réussit à sortir des cartes d'activation qui lui serviront ensuite de repères pour l'analyse d'images.

Ci-dessous, on peut retrouver le fonctionnement de la convolution :

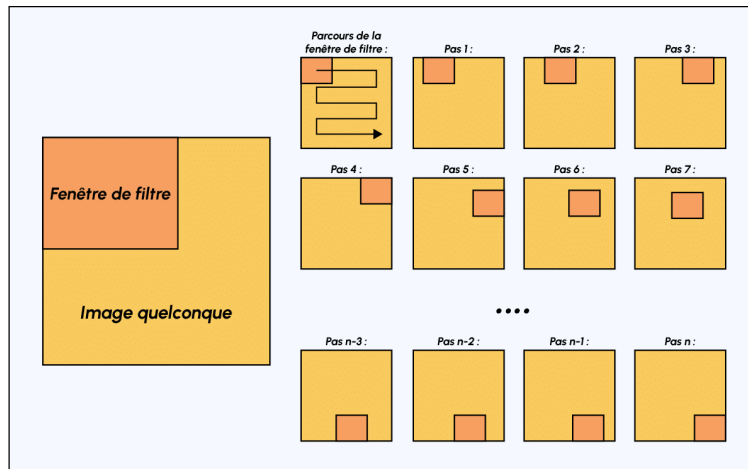


FIGURE 4 – Schéma représentatif de la convolution

2. Le Max-Pooling : il permet de réduire la dimension spatiale en préservant les caractéristiques importantes de l'image, autrement dit il réduit le nombre de paramètres pour ne garder que ceux qui lui seront utiles.
3. Les couches complètement connectées : cette partie vise à réaliser la classification finale grâce aux paramètres et caractéristiques trouvés à l'aide des étapes précédentes.
4. Les fonctions d'activation : Elles permettent de rendre le modèle non-linéaires et donc plus complexes. Celle qui est la plus utilisée remplace toutes les valeurs négatives reçues en entrée par des zéros.
5. L'entraînement et la rétropropagation : l'algorithme de rétropropagation utilisé par les CNN permet d'ajuster le poids du réseau et de minimiser l'erreur de prédiction.



Les CNN peuvent être représentés sous la forme d'un schéma comme ci dessous :

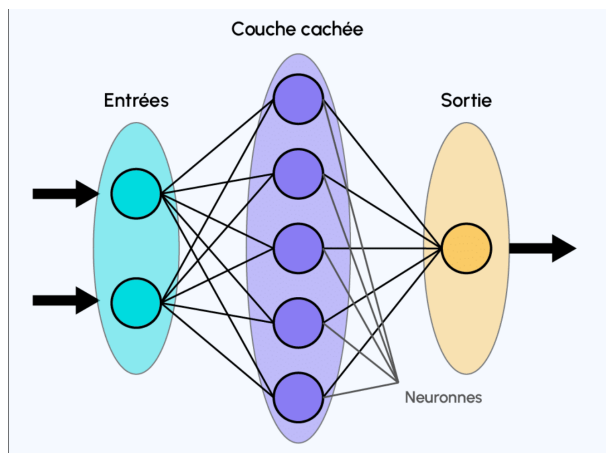


FIGURE 5 – Schéma représentatif du CNN

En entrée, on retrouve les images ainsi que leurs classes. Au niveau des couches cachées, on retrouve tout le réseau de neurones ainsi que ses couches. En sortie on trouve un algorithme totalement entraîné et prêt à reconnaître les images du même type que celle qui ont servi lors de l'entraînement.

## 3 La machine à support de vecteurs ou SVM

### 3.1 Le Machine Learning

Comme précédemment, afin de comprendre ce qu'est le SVM, il nous faut d'abord comprendre ce qu'est le Machine Learning (ou apprentissage automatique). Le Machine Learning est une branche de l'intelligence artificielle qui consiste à laisser des algorithmes découvrir des motifs récurrents dans des ensembles de données afin de permettre à ces mêmes algorithmes d'effectuer une tâche ou de réaliser des prédictions à partir des mêmes données. Il peut être utilisé pour la classification, le clustering ou encore la détection d'anomalie.

On retrouve deux grands types de Machine Learning : l'apprentissage supervisé et l'apprentissage non-supervisé. Pour ce qui est de l'apprentissage supervisé, les données sont étiquetées afin d'indiquer à la machine quelles motifs récurrents elle doit rechercher. Au contraire de l'apprentissage non-supervisé qui, lui, absorbe les données et met lui même en relief les motifs récurrents grâce à différents algorithmes.

Ici, on se concentrera sur un algorithme d'apprentissage supervisé qu'on appelle le SVM (Support Vector Machine) qui est l'algorithme de Machine Learning le plus adéquat pour notre étude car il permet de classer les données.

### 3.2 L'algorithme

La machine à support de vecteurs a d'abord été créée pour la classification binaire mais on peut maintenant l'utiliser pour la classification multi-classe. C'est un algorithme pour la classification mais aussi pour la régression ce qui le rend plutôt polyvalent.

Le SVM se caractérise par différentes étapes de construction :

1. L'hyperplan optimal : l'algorithme trouve un hyperplan optimal afin de séparer les données de différentes classes. Pour avoir le meilleur SVM, l'hyperplan choisis sera celui qui maximise la marge, ce qui nous amène à l'étape suivante.
2. Les marges maximales : La maximisation de la marge permettra la création de l'hyperplan optimal. Les deux étapes sont donc complémentaires.

On retrouve ici un exemple donnant deux hyperplans, l'un optimal, l'autre non :

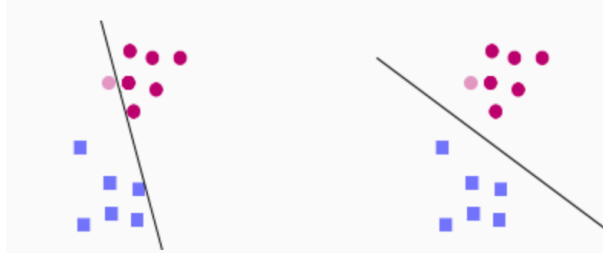


FIGURE 6 – Hyperplans pour une même situation

Pour cet exemple, l'algorithme doit reconnaître les carrés bleus et les ronds roses. Il doit pouvoir affecter le rond rose pale à une de ces classes, celle dont il s'en rapproche le plus. Ici, l'hyperplan à droite est optimal à l'instar de celui à gauche. On cherche ici l'unique hyperplan qui maximise la marge dont les paramètres  $(w, b)$  sont donnés par la formule :

$$\arg \max_{w, b} \min_k \frac{l_k(w^\top \cdot x_k + b)}{\|w\|}$$

FIGURE 7 – Formule de maximisation de la marge

3. Le noyau : si les données sont non linéairement séparables, il sera obligatoire d'utiliser l'astuce des noyaux. On utilise les noyaux pour transformer les données d'entrées en un espace de dimension supérieure. Autrement dit, on applique une fonction qui permettra de réunir nos données de même classe. On retrouve un certain nombre de noyau tel que :

- Le noyau polynomial,  $K(x, x') = (\alpha x^\top \cdot x' + \lambda)^d$
- Le noyau gaussien,  $K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$
- Le noyau laplacien,  $K(x, x') = \exp\left(-\frac{\|x - x'\|}{\sigma}\right)$
- Le noyau rationnel,  $K(x, x') = 1 - \frac{\|x' - x\|^2}{\|x' - x\|^2 + \sigma}$

FIGURE 8 – Différents noyaux

4. La régularisation : celle-ci permet d'éviter le surapprentissage en contrôlant la complexité du modèle.

Ici, nous avons présenté le SVM pour une classification binaire mais celle-ci ne diffère pas tant que ça de la classification multi-classe. Il y a deux méthodes utilisées en amont pour cette classification :

1. La one-vs-one qui consiste à se concentrer sur deux classes et répéter l'opération pour que chaque classe ait été comparée par le SVM.

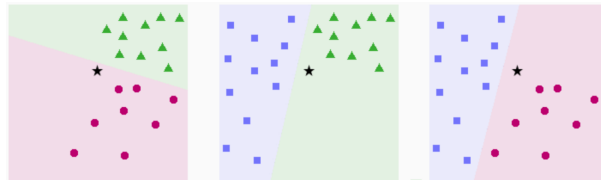


FIGURE 9 – One-vs-one

2. La one-vs-all qui consiste à analyser chaque classe une par une avec le SVM afin de la différencier des autres.

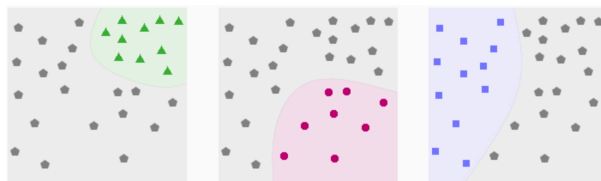


FIGURE 10 – One-vs-all

## 4 Comparaison

### 4.1 Deep Learning vs Machine Learning

Tout d'abord, il faut savoir que le Deep Learning est une sous catégorie du Machine Learning mais elle a bien plus de capacité que le Machine Learning supervisé. Grâce à ses réseaux de neurones, elle peut copier le cerveau humain et analyser des données sans l'aide d'indications préalable à l'instar du Machine Learning supervisé qui, lui, aura besoin d'être guidé par l'humain dans sa recherche. C'est l'humain qui devra d'abord étudier les différents paramètres et trouver ceux qui seront les plus utiles pour l'étude à réaliser. En Deep Learning, il suffit de donner les données et l'algorithme mettra en lumière seul les paramètres les plus intéressants pour l'étude.

Cependant, étant donné le travail colossal que doit réaliser l'algorithme de Deep Learning, il lui faut bien plus de données qu'un algorithme de Machine Learning. Dans cette même optique, l'algorithme de Machine Learning ne pourra pas étudier un jeu de données trop volumineux, ce qui reste un désavantage de nos jours avec la récolte de données qui devient de plus en plus fréquente.

Le temps d'entraînement d'un algorithme de Machine Learning est plus court cependant il n'aura pas de capacités aussi importantes qu'un algorithme en Deep Learning et sa précision en sera donc plus faible.

Enfin, le Machine Learning permet l'entraînement sur un CPU (unité de traitement centrale) tandis que le Deep Learning requiert un GPU (unité de traitement graphique). Ce puissant hardware est indispensable pour traiter les larges volumes de données et effectuer les calculs complexes des algorithmes.

### 4.2 CNN ou SVM

Comme vu précédemment, le procédé des deux algorithmes est totalement différent. Cependant, ils analysent tous deux des données chiffrées (ici des pixels) afin de répertorier ces données dans les différentes classes proposées. Le résultat final est le même, cependant il nous faudrait savoir si l'un des algorithmes est plus efficace que l'autre en mettant en relief les différences trouvées précédemment dans la comparaison entre l'algorithme de Machine Learning et Deep Learning.

D'un point de vue théorique, le CNN serait l'algorithme le plus pratique pour cette étude. D'abord, il est spécialisé dans la reconnaissance d'images et on veut ici reconnaître le chiffre écrit sur l'image correspondante. Ensuite, il est bien plus performant avec beaucoup de données et ici, on retrouve 42000 lignes pour 785 colonnes dans le fichier d'entraînement, ce qui représente tout de même beaucoup de données. Pour finir, les données ici ne peuvent pas être séparées, on ne pourra donc pas choisir des données plus ou moins utiles à notre étude. Or, en Machine Learning supervisé, les algorithmes ont l'habitude de ne prendre que les paramètres nécessaires donnés par l'humain contrairement au Deep Learning qui, lui, trie les données et utilise les paramètres les plus importants de lui-même sans aide préalable.

On peut donc, d'ores et déjà, affirmer que le CNN sera l'algorithme le plus utile pour cette étude. Malgré tout, une étude plus poussée nous permettra de différencier ces deux algorithmes grâce à leurs performances et donc émettre une comparaison finale.

## Partie pratique

Afin de réaliser au mieux cette étude, il faut étudier nos deux algorithmes grâce à un logiciel de programmation. Ici, le choix le plus judicieux était Python qui regorge de librairie utile pour le Machine Learning et le Deep Learning.

## 5 Préparation du jeu de données

Tout d’abord, il faut analyser les données afin de mieux les comprendre et les préparer pour qu’elles soient utilisées de manière optimal par les deux algorithmes.

### 5.1 Importation des librairies et fonctions

Sur le logiciel Python, afin d’utiliser certaines fonctions, il faut d’abord les importer ou importer toute la librairie les contenant. On retrouve d’abord la librairie numpy (appelée ici np) qui contient un bon nombre d’opérations qui nous seront bien utiles ici.

Ensuite, il y a les librairies pandas (appelée pd) et csv qui permettent de charger des jeux de données sur le programme et d’observer les différentes caractéristiques de ceux-ci.

Il y a aussi la fonction display de la librairie IPython et la librairie time qui fournisse des fonctions qui permettent de régler les temps d’affichage et de traitement.

On retrouve aussi les librairies matplotlib.pyplot (appelée plt) et plotly.express (appelée px) qui permettent d’afficher différents graphiques.

Pour créer l’algorithme CNN, il fallait importer la librairie tensorflow (appelée tf) et les fonctions layers et models de la librairie tensorflow.keras.

Pour créer l’algorithme SVM, il a fallu importer trois fonctions de la librairie sklearn : train\_test\_split, SVC et accuracy\_score.

## 5.2 Visualisation des images et label

Suite à l'importation des librairies, il faut maintenant importer les jeux de données. J'ai donc importé le jeu d'entraînement dans deux variables "mnist" et "data", l'une utilisée pour le CNN, l'autre pour le SVM. J'ai aussi importé le jeu de test dans la variable test\_data. J'ai vérifié les tailles de mes jeux de données et on trouve bien les tailles données dans la description du jeu de données.

```
(42000, 785)
(28000, 784)
```

FIGURE 11 – Taille des jeux de données (train et test)

Je peux maintenant visualiser les images de mon jeu de données d'entraînement et vérifier si le label donné coïncide bien avec l'image représentée.

Pour cela, il fallait d'abord convertir chaque ligne en une matrice 28x28, mettre le label correspondant en titre et afficher l'image. Pour exemple, on trouve une image comme ci-dessous :

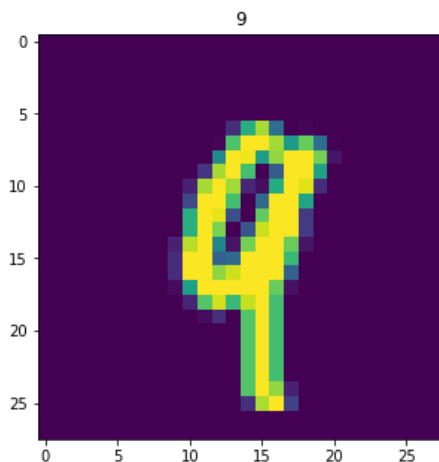


FIGURE 12 – Image représentée à la ligne 501 du jeu de données



Pour ce qui est des labels, j'ai visualisé leur répartition à l'aide d'un diagramme en camembert afin de voir si les classes étaient homogènement réparties.

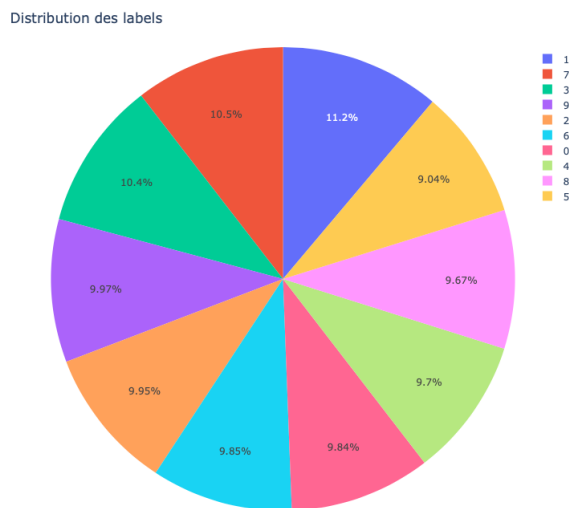


FIGURE 13 – Répartition en camembert des classes

Ici, on peut voir que les classes sont bien réparties.

### 5.3 Preprocessing pour le CNN et le SVM

Il faut maintenant préparer chaque jeu de données afin qu'ils soient utilisés au mieux lors de la création des algorithmes.

Pour ce qui est du CNN, pour le jeu de donnée d'entraînement, il a fallu délimiter dans deux variables la colonne "label" qui contient les classes et les colonnes contenant les pixels. J'ai appelé ces deux variables `train_data` et `train_label`. J'ai vérifié leurs tailles afin d'être sûr qu'il contenait le nombre de lignes et colonnes qu'il devait contenir. Pour finir, j'ai remodelé les tailles des variables `test_data` et `train_data` afin que les deux contiennent des matrices 28x28 en données et j'ai ensuite normalisé ces matrices pour que le CNN ait plus de facilités à analyser les données.

Pour ce qui est du SVM, j'ai affecté les valeurs des pixels à la variable `X` et les valeurs du label à la variable `y`. J'ai ensuite remodelé et normalisé ma variable `X` de sorte qu'elle contienne des matrices 28x28.

## 6 Les réseaux de neurones convolutifs

Grâce au travail préalablement fourni, je peux maintenant créer mon modèle CNN.

Dans un premier temps, il faut créer une variable contenant le modèle séquentiel. Suite à cela, je peux créer une première couche de convolution contenant 32 filtres de taille (3,3) ainsi que la fonction d'activation "relu" dont j'ai parlé précédemment lors de la présentation théorique du CNN. Cette même fonction qui renvoie zéro si le résultat est négatif. On trouve aussi dans cette première couche la forme d'entrée des données, ici (28,28,1), pour une matrice de 28x28 avec des pixels à nuance de gris.

Après cela, il faut réduire la dimension spatiale grâce à une couche de max-pooling ayant une fenêtre de taille (2,2). Il faut ensuite créer une autre couche de convolution mais ici avec 64 filtres et sans précision des données d'entrées. Puis on réitère avec une couche de max-pooling puis une couche de convolution à 64 filtres.

Il faut maintenant ajouter les couches de classification. On y retrouve une couche d'aplatissement de 2D en 1D. Puis une couche dense avec 64 filtres et la fonction "relu" et enfin, une couche dense de sortie avec 10 neurones représentant les 10 classes de notre jeu de données.

La compilation du modèle est maintenant possible. Il faut ensuite l'entraîner à partir de nos données train\_data et train\_label avec 10 epochs d'entraînement, c'est-à-dire 10 itérations. Cela nous donne l'entraînement suivant :

```
Epoch 1/10
1313/1313 [=====] - 37s 27ms/step - loss:
0.1751 - accuracy: 0.9463
Epoch 2/10
1313/1313 [=====] - 39s 29ms/step - loss:
0.0528 - accuracy: 0.9838
Epoch 3/10
1313/1313 [=====] - 38s 29ms/step - loss:
0.0380 - accuracy: 0.9877
Epoch 4/10
0s 30ms/step - loss: 0.0288 - accuracy: 0.9910
Epoch 5/10
1313/1313 [=====] - 39s 30ms/step - loss:
0.0231 - accuracy: 0.9926
Epoch 6/10
1313/1313 [=====] - 38s 29ms/step - loss:
0.0194 - accuracy: 0.9936
Epoch 7/10
1313/1313 [=====] - 38s 29ms/step - loss:
0.0153 - accuracy: 0.9952
Epoch 8/10
1313/1313 [=====] - 41s 31ms/step - loss:
0.0135 - accuracy: 0.9955
Epoch 9/10
1313/1313 [=====] - 37s 29ms/step - loss:
0.0104 - accuracy: 0.9964
Epoch 10/10
1313/1313 [=====] - 38s 29ms/step - loss:
0.0089 - accuracy: 0.9970
875/875 [=====] - 5s 6ms/step
```

FIGURE 14 – Itérations d'entraînement

Pour la dernière itération d'entraînement, on retrouve une précision 0.9970 ainsi qu'une perte de données de seulement 0.0089. De cela, on peut tout de suite émettre la conclusion que le CNN est totalement adapté à la situation.

Afin de vérifier si le modèle est bien efficace, j'ai effectué des prédictions sur le modèle et j'ai ensuite vérifié si ces prédictions étaient bonnes en affichant aléatoirement des images du fichier ainsi que la prédiction en titre de celle-ci. Ci-dessous, on retrouve un exemple d'image dont la classe a été prédite :

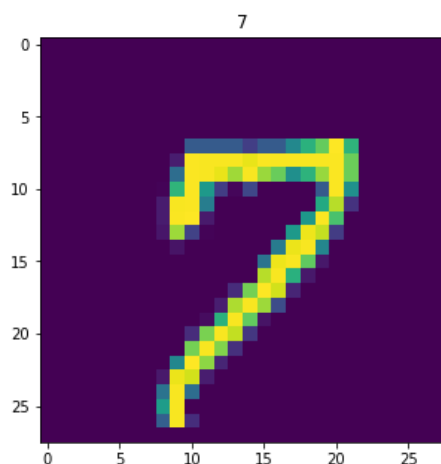


FIGURE 15 – Prédiction de l'image à la 21001ème ligne du fichier test

Le temps d'exécution du CNN n'a pas été très long, si on additionne le temps d'entraînement de chaque itération, on trouve un temps global de quelque minutes, ce qui n'est pas énorme.

## 7 La machine à support de vecteurs

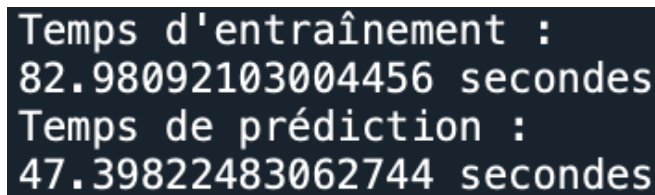
Comme précédemment, je peux tout de suite créer mon modèle SVM grâce au travail fourni au préalable.

Dans un premier temps, j'ai coupé mes jeux de donnée X et y en deux en créant une variable pour l'entraînement contenant 80% des données et l'autre contenant 20 % pour le test.

Pour mes deux variables X d'entraînement et de test, j'ai appliqué une transformation de déroulement pour qu'elles puissent être utilisées dans mon modèle SVM.

Il m'a ensuite suffi d'entraîner mon modèle, tout en vérifiant sont temps d'entraînement, grâce à mes variables d'entraînement. Puis, j'ai réalisé les prédictions de mon modèle, tout en vérifiant le temps de prédiction, grâce a ma variable X\_test.

On retrouve les temps d'entrainement et de prédiction suivant :



```
Temps d'entraînement :  
82.98092103004456 secondes  
Temps de prédiction :  
47.39822483062744 secondes
```

FIGURE 16 – Temps d'entraînement et de prédiction

En additionnant les deux temps, on trouve à peu près deux minutes d'exécution pour le SVM, ce qui est très raisonnable étant donné le volume du jeu de données.

J'ai ensuite calculé la précision du jeu de données grâce à mes variables répertoriant les classes et trouvé environ 0.97. Ce résultat est très bien, cela nous donne tout de même une bonne idée des performances du modèle.

Comme précédemment, j'ai aussi vérifié si mes prédictions étaient exact en utilisant les variables `X_test` et `y_pred` :

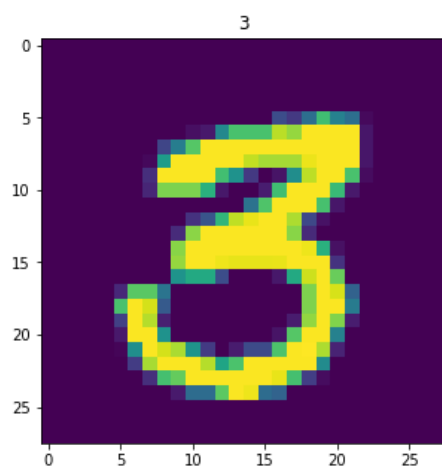


FIGURE 17 – Prédiction de l'image à la 2807ème ligne du fichier test

## 8 Comparaison

Grâce à la création des deux algorithmes, je peux maintenant effectuer une comparaison plus pratique et donner mon avis sur ces deux modèles.

D'abord, la complexité des algorithmes est différente. Pour ce qui est du CNN, il y a tout une création de l'algorithme à effectuer au préalable tandis que pour le SVM, seul les données doivent être préparées, le modèle étant déjà créé dans la librairie où la fonction est récupérée. Sur ce point là, on peut d'ores et déjà dire que le SVM est plus facile à créer en Python que le CNN.

Ensuite, pour ce qui est des temps d'exécution, il n'y a pas énormément de choses à dire si ce n'est que les deux algorithmes se rejoignent avec des temps d'exécution de quelques minutes. Cependant, si j'avais fait le choix de réaliser plus ou moins d'epochs pour le CNN, son temps d'exécution serait aussi plus ou moins long, cependant cela aurait affecté la précision du modèle.

Pour finir, pour ce qui est de la précision du modèle, le CNN l'emporte grâce à ses 99.7%, cependant le SVM ne se trouve pas si loin avec une précision de 97.3%.

## Conclusion

Pour conclure, cette étude m'aura permis d'abord de découvrir la base de données MNIST et de mieux la comprendre. Ensuite, d'en savoir plus sur le Machine Learning et le Deep Learning et de mieux comprendre les algorithmes de CNN et SVM ce qui fut très enrichissant et intéressant.

Cette étude m'aura aussi permis de mettre en relief l'idée selon laquelle le CNN était plus adapté pour la classification du jeu de données MNIST. Ce qui fut vérifiable même si le modèle SVM l'est aussi, le CNN reste plus performant.

D'autres paramètres peuvent encore être comparés entre les deux algorithmes et peut-être que ces nouveaux paramètres mettront en lumière une comparaison différente de celle que j'ai faite.

Afin de compléter cette étude, il est possible de la réaliser sur d'autres algorithmes en essayant de choisir des algorithmes aussi différents que le CNN et le SVM. Cela donnerait un regard neuf sur le jeu de données ainsi que les différents algorithmes de Machine Learning et Deep Learning.

## Références

<https://www.kaggle.com/competitions/digit-recognizer/overview>

<https://datascientest.com/quelle-difference-entre-le-machine-learning-et-deep-learning#:~:text=Le%20Deep%20Learning%20requiert%20de,apprendre%20et%20corriger%20ses%20erreurs.>

<https://www.talend.com/fr/resources/what-is-machine-learning/#:~:text=Les%20principaux%20algorithmes%20du%20machine,logistique%20et%20boosting%20de%20gradient.>

[https://fr.wikipedia.org/wiki/Base\\_de\\_donnes\\_MNIST#:~:text=La%20base%20de%20donnes%20MNIST,trs%20utilis%20en%20apprentissage%20automatique.](https://fr.wikipedia.org/wiki/Base_de_donnes_MNIST#:~:text=La%20base%20de%20donnes%20MNIST,trs%20utilis%20en%20apprentissage%20automatique.)

<https://www.jedha.co/formation-ia/vraie-difference-machine-learning-deeplearning#:~:text=Il%20est%20souvent%20expliqu%20que,son%2C%20le%20texte%2C%20l%27>

<https://datascientest.com/convolutional-neural-network>

<https://datascientest.com/svm-machine-learning>

<https://datascientest.com/machine-learning-tout-savoir>

<https://datascientest.com/quelle-difference-entre-le-machine-learning-et-deep-learning#:~:text=Le%20Deep%20Learning%20requiert%20de,apprendre%20et%20corriger%20ses%20erreurs.>

<https://www.talend.com/fr/resources/what-is-machine-learning/#:~:text=Les%20principaux%20algorithmes%20du%20machine,logistique%20et%20boosting%20de%20gradient.>

<https://zestedesavoir.com/tutoriels/1760/un-peu-de-machine-learning-avec-les-svm/>



## Appendix

Vous trouverez ci-après mon programme python que j'ai téléchargé en tant que pdf sur le logiciel Latex que j'ai utilisé pour écrire mon rapport.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

Created on Mon Jun 12 18:56:15 2023

```
@author: sokaynachaoui
"""
```

```
#Importation des librairies
```

```
##Partie 1
```

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import csv
```

```
from IPython import display
```

```
import time
```

```
import plotly.express as px
```

```
##Partie 2
```

```
import tensorflow as tf
```

```
from tensorflow.keras import layers, models
```

```
##Partie 3
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.svm import SVC
```

```
from sklearn.metrics import accuracy_score
```

```
#### Partie 1 : Préparation des jeu de données
```

```
#Affichage de toutes les colonnes
```

```
pd.set_option("display.max_columns", None)
```

```
#Chargement des csv d'entrainement et de test
```

```
mnist = pd.read_csv("/Users/sokaynachaoui/Documents/projet  
d'été/train.csv",delimiter=",")
```

```
test_data = pd.read_csv("/Users/sokaynachaoui/Documents/projet  
d'été/test.csv",delimiter=",").values
```

```
data = pd.read_csv("/Users/sokaynachaoui/Documents/projet d'été/train.csv")
```

```
#Affichage du nombre de ligne et colonnes
```

```
print(mnist.shape)
```

```
print(test_data.shape)
```

```
# Affichage des images du jeu d'entrainement
```

```
with open("/Users/sokaynachaoui/Documents/projet d'été/train.csv", 'r') as f:
```

```
    reader = csv.reader(f, delimiter = ",")
```

```
    next(reader) # Passe la première ligne
```

```
    ligne = f.readlines() # Extrait les lignes du csv
```

```
    for l in ligne[1:][500:503]: # Extraction des données à partir de la deuxième colonne et de  
la ligne 501 à la ligne 504 (choisis au hasard pour vérification)
```

```

split_lines = [int(x) for x in l.split(",")] # Retire les ' pour reutiliser la ligne comme une
matrice
newline = split_lines[1:] # Suppression du premier élément qui ne correspond pas a un
pixel de l'image mais au chiffre représenté sur l'image
#Verification
print(newline)
#Création d'une matrice 28x28 avec chaque ligne
matrice = np.reshape(newline,(28,28))
print(matrice) #Verification
#Affichage de l'image
#Mise en place du graphe
plt.figure(figsize=(5,5))
plt.imshow(matrice)
plt.title(split_lines[0]) # Chiffre correspondant en titre
display.clear_output(wait=True)
pause_time = 0.2
time.sleep(pause_time)

```

```

#Préparation des données pour CNN
# Creation de deux nouvelles data frame repertoriant chaque images et chaque labels
train_data, train_label = mnist.iloc[:, 1:].values, mnist['label'].values
print(train_data.shape)
print(train_label.shape)

```

```

train_data=train_data.reshape(-1,28,28,1)

```

```

test_data=test_data.reshape(-1,28,28,1)

```

```

# Normaliser les valeurs des pixels entre 0 et 1
train_data, test_data = train_data / 255.0, test_data / 255.0

```

```

#Préparation des données pour SVM

```

```

# X matrice 28x28 et y label
X = data.drop('label', axis=1)
y = data['label']

```

```

X=X.values.reshape(-1,28,28,1)
X = X / 255.0

```

```

## Visualisation (Camembert de répartition des labels)

```

```

# Convertir train_label en une série Pandas
train_label_series = pd.Series(train_label)

```

```

# Obtenir les comptages des valeurs
label_dis = train_label_series.value_counts()

```

```
# Extrait le nombre d'années de chaque experience de experience_dis
index = label_dis.index
```

```
# Creation du diagramme en camembert
fig = px.pie(values=label_dis, names=index, color=index)
```

```
# Création du titre
fig.update_layout(title_text='Distribution des labels' )
```

```
# Affichage du diagramme sur un navigateur web
fig.show(renderer='browser')
```

```
#### Partie 2 : Mise en place du CNN
```

```
## Création du modèle CNN
# Création du modèle séquentiel
model = models.Sequential()
# Création d'une première couche de convolution avec 32 filtres de taille (3, 3), une fonction
d'activation ReLU, et une forme d'entrée de (28, 28, 1)
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
# Reduction de la dimension spatiale avec couche de max-pooling avec fenêtre (2,2)
model.add(layers.MaxPooling2D((2, 2)))
# Autre couche de convolution mais 64 filtres
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
# Autre Max-pooling
model.add(layers.MaxPooling2D((2, 2)))
# couche de convolution 64 filtres
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
## Ajout des couches de classification
#Couche d'aplatissement de 2D en, 1D (Couche complètement connecté)
model.add(layers.Flatten())
#Couche dense avec 64 filtres
model.add(layers.Dense(64, activation='relu'))
#Couche dense de sortie avec 10 neurones pour les dix classes
model.add(layers.Dense(10)) # 10 classes de chiffres
```

```
## Compiler le modèle
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
## Entraîner le modèle
model.fit(train_data, train_label, epochs=10) #10 epochs d'entrainement
```

```
## Faire des prédictions sur les données de test
```

```

predictions = model.predict(test_data)

## Retrouver la classe prédite

softmax_predictions = np.exp(predictions - np.max(predictions, axis=1, keepdims=True))

softmax_predictions /= np.sum(softmax_predictions, axis=1, keepdims=True)

predicted_classes = np.argmax(softmax_predictions, axis=1)

print(predicted_classes)

# Verification des prédictions d'image test (21000 n'est qu'un exemple)
plt.figure(figsize=(5,5))
plt.imshow(test_data[21000])
plt.title(predicted_classes[21000]) # Chiffre correspondant en titre
display.clear_output(wait=True)
pause_time = 0.2
time.sleep(pause_time)

```

### ### Partie 3 : Mise en place du SVM

```

##Couper le fichier en 80% et 20% pour l'entrainement et le test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

```

# Appliquez une transformation de déroulement (flatten) pour chaque image
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

```

```

svm_model = SVC(kernel='rbf', gamma='scale')

```

```

##Entrainement du modèle
start_time = time.time()
svm_model.fit(X_train, y_train)
training_time = time.time() - start_time #Temps d'entrainement du modèle

```

```

##Prédiction du modèle
start_time = time.time()
y_pred = svm_model.predict(X_test)
prediction_time = time.time() - start_time #Temps de prédiction du modèle

```

```

##Calcul de la précision du modèle
accuracy = accuracy_score(y_test, y_pred)

```

```
print(f'Exactitude du modèle SVM : {accuracy}')
```

# Affichez les temps de traitement

```
print(f'Temps d\'entraînement : {training_time} secondes')
print(f'Temps de prédiction : {prediction_time} secondes')
```

# Verification des prédictions d'image test

# Utilisation de reshape pour redonner sa forme d'origine à X\_test

```
X_test = X_test.reshape(X_test.shape[0], 28, 28)
plt.figure(figsize=(5,5))
plt.imshow(X_test[2806]) #Pris au hasard pour vérification
plt.title(y_pred[2806]) # Chiffre correspondant en titre
display.clear_output(wait=True)
pause_time = 0.2
time.sleep(pause_time)
```