Rapport de projet : Machine learning avec Pyhton

MAM 3 - S6

BENCHIHA Etan, COSTANTIN Perline, HAJLAOUI Khaoula

13 juin 2025

Table des matières

1 Introduction

1.1 Introduction générale

Dans ce rapport, nous allons vous présenter le travail que nous avons effectué au cours de cette semaine dans le cadre de ce projet "Machine Learning avec Python". Le machine learning permet à un système d'apprendre à partir de données connues afin d'effectuer des prédictions sur de nouvelles données. Nous allons travailler sur un exemple de machine learning et faire de l'apprentissage supervisé, qui consiste à entraîner notre modèle à partir d'exemples étiquetés.

L'objectif de notre projet est de récupérer un jeu de données, choisir un modèle, entraîner un modèle, prédire des résultats, analyser la donnée et réduire sa dimension en supprimant certaines informations tout en gardant un résultat fiable.
Nous allons entraîner un modèle pour reconnaître des chiffres sur des images manuscrites. Notre modèle devra apprendre à reconnaître des motifs visuels pour ensuite être capable de classer correctement de nouvelles images jamais vues. Cela est utilisé dans la vie quotidienne, notamment utilisée pour le tri du courrier postal, le traitement des chèques bancaires ou encore la saisie de données dans les formulaires Pour s'aider durant ce processus, nous utiliserons la bibliothèque sklearn (scikit-learn) qui regroupe une grande quantité d'outils en rapport avec le machine learning.

1.2 Base de données

Pour ce projet, nous allons utiliser le jeu de données « Handwritten digits » de la bibliothèque Sklearn. Chaque donnée est une image de 8 pixels par 8 pixels, en niveau de gris (0 pour noir et 16 pour blanc). Elle est stockée sous la forme d'un vecteur de dimension 64 comme une ligne de la matrice X et avec la valeur de la classe associée stockée dans un vecteur y à part. Nous avons 1797 lignes dans la matrice X, soit 1797 images dans notre base de données.
L'entrée de notre programme sera donc un chiffre manuscrit et la sortie le chiffre correspondant (0 à 9).

1.3 Introduction aux différents types de machine learning

Il existe trois grands types de machine learning :

— l'apprentissage supervisé, que nous allons utiliser ici, dans lequel le jeu de données est labellisé (on connaît la bonne réponse pour chaque donnée). Les labels sont utilisés lors de l'apprentissage du modèle

— l'apprentissage non supervisé, dans lequel il n'y a pas de réponses données à la machine

— l'apprentissage par renforcement, dans lequel la machine apprend par essais et erreurs en recevant des récompenses ou des pénalités

En apprentissage supervisé, plusieurs types de modèles existent :

— La classification / régression logistique si les étiquettes sont non numériques

— la régression si les valeurs sont continues

— la régression ordinale si les étiquettes sont des nombres ordonnés qui ne représentent pas des classes

Ici, on travaille sur les chiffres, les étiquettes sont donc les entiers entre 0 et 9, on pourrait donc penser qu'il s'agit d'une classification ordinale, mais c'est en fait de la régression car les nombres représentent des classes et ne sont donc pas ordonnés.

2 Etape 0 : Analyse et exploration de données

Nous chargeons tout d'abord les données depuis scikit learn. Pour commencer à travailler sur notre base de données, nous allons visualiser les données que nous avons récupérées. Chaque image de chiffre manuscrit est une matice 8 par 8 où chaque valeur représente le niveau de gris sur le pixel correspondant, par exemple :

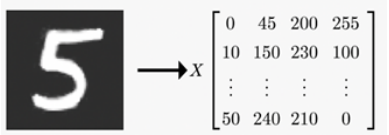


FIGURE 1 – exemple de représentation d'un nombre manuscrit dans la base de données

On va afficher les quatres premières photos de notre dataset :

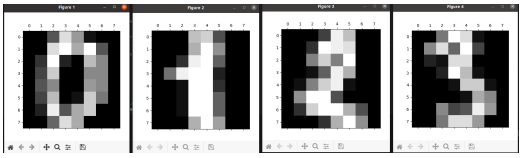


FIGURE 2 – Visualisation des 4 premières images du dataset

Voici une autre visualisation que nous avons faite en utilisant matplotlib pour afficher une image par classe :

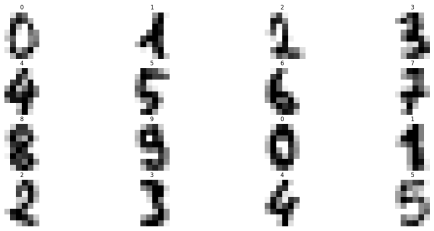


FIGURE 3 – Visualisation dataset

Pour vérifier si nos données sont exploitables, on trace la distribution de chaque classe dans le dataset :

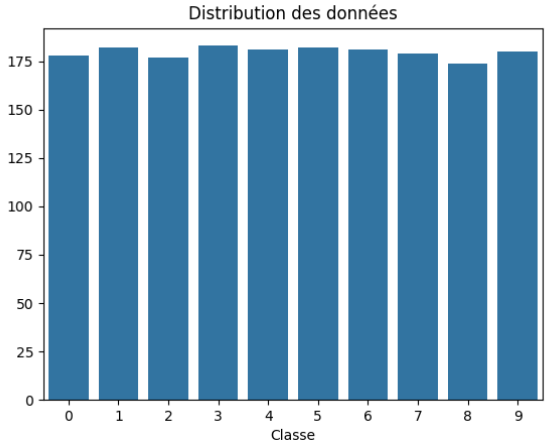


FIGURE 4 – Distribution des données

Les classes sont équilibrées, les données sont exploitables car on a suffisament de données dans chaque classe (175 images/classe). Il y a une entropie forte dans les données.

3 Etape 1 : Visualisation et préparation des données pour les modèle

Nous voulons approximer une fonction $f(x) \approx y$, où $f(x)$ est une fonction qui associe une entrée x à une sortie y. On note X la matrice de dimensions $N \times m$, avec N le nombre d'individus (images) et m le nombre de features (pixels). La variable y représente la variable cible, autrement dit la classe associée à chaque image.

X = `digits.data`
y = `digits.target`

3.1 Normalisation

On normalise les données contenues dans X pour simplifier le traitement et éviter les différences de dynamique dans les valeurs des pixels.

Chaque pixel peut prendre une intensité entre 0 et 16 (car il y a 16 niveaux de gris), donc on divise chaque valeur de la matrice par 16. On obtient alors une nouvelle matrice $F = \frac{X}{16}$ qui contient les valeurs normalisées (entre 0 et 1), avec chaque ligne qui représente une image.

Ainsi, chaque image est représentée sous forme d'un vecteur de features normalisés pour faciliter l'entraînement du modèle de machine learning fait dans la suite.

4 Etape 2 : Extraction des caractéristiques

4.1 Réduction de dimension avec l'Analyse en Composantes Principales

Pour continuer dans le prétraitement, on projette nos données sur un nouvel espace à n dimensions. Il faut alors trouver un compromis entre la taille de l'espace projeté (nombre de dimensions), et la variance conservée. Pour cela, nous allons utiliser la méthode de l'analyse en composantes principales (ACP) qui est une méthode de réduction de dimensionnalité.

Dans un premier temps, nous effectuons une visualisation en deux dimensions pour observer le comportement du modèle, et pour comparer l'image originale et l'image reconstruite après l'ACP.
Plutôt qu'une simple comparaison pixel à pixel, on utilise la Structural Similarity (SSIM) qui permet de mesurer la similarité de structure entre deux images, car l'œil humain est plus sensible aux changements dans la structure de l'image que dans les pixels. On a ainsi une comparaison qui transcrit mieux la perception de l'oeil humain.

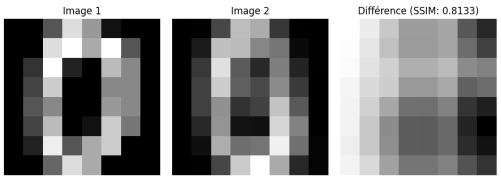


FIGURE 5 – Comparaison de l'image originale et l'image reconstruite après l'ACP avec le SSIM

Pour une dimension ACP de 2 (réduction à 2 composantes principales), on a donc une assez bonne reproduction de l'image de départ malgré le nombre de dimensions.

On se demande tout de même comment varie ce coefficient SSIM en fonction de la dimension mise dans la fonction PCA. Pour cela, on fait une boucle qui reconstruit des images en différentes dimensions avec le PCA et on observe les résultats pour le SSIM.

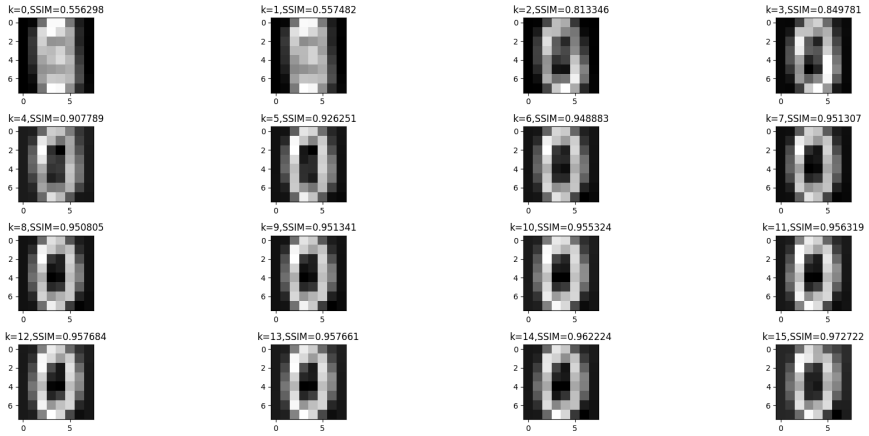


FIGURE 6 – SSIM en fonction du nombre de dimensions (k) de chaque ACP

On remarque qu'en augmentant le nombre de composantes principales, la qualité de reconstruction s'améliore : on aura un SSIM proche de 1. Cela est logique car si on projette une image, plus la dimension de l'espace de projection est grande et plus on aura une image reproduite proche de l'originale.

Remarque : on arrive bien à la même conclusion en faisant le processus en calculant l'erreur relative au lieu du SSIM (plus on ajoute des dimensions, plus l'erreur est proche de 0)

4.2 Clustering

Maintenant, nous pouvons également visualiser la répartition des chiffres en fonction de leurs deux premières composantes principales :

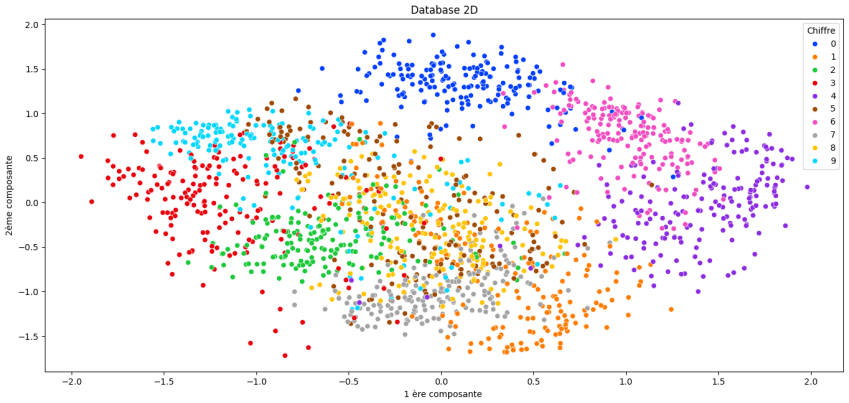


FIGURE 7 – Visualisation des clusters en 2D après la PCA

On observe que par exemple les 0 sont bien isolés et bien regroupés les uns des autres, ce qui signifie que pour les 2 composantes principales, la machine n'a pas de problème pour détecter ce chiffre, ce qui est aussi le cas des 4 et des 6. Cependant, les 7 ou les 8 par exemple se confondent avec d'autres chiffres et sont asez dispersés, donc on peut donc en déduire que nous aurons plus d'erreurs de prédiction sur ces chiffres là.

4.3 Nombre de dimensions optimal

On cherche maintenant à déterminer le meilleur compromis entre réduction de dimension et perte d'informations.
Pour cela, on trace le graphe de la variance cumulée expliquée en fonction du nombre de composantes dans le PCA :

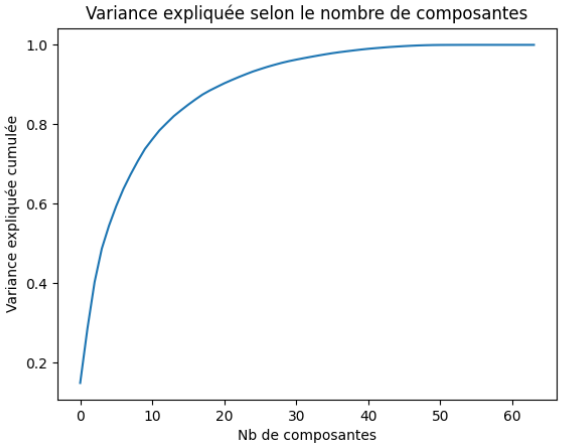
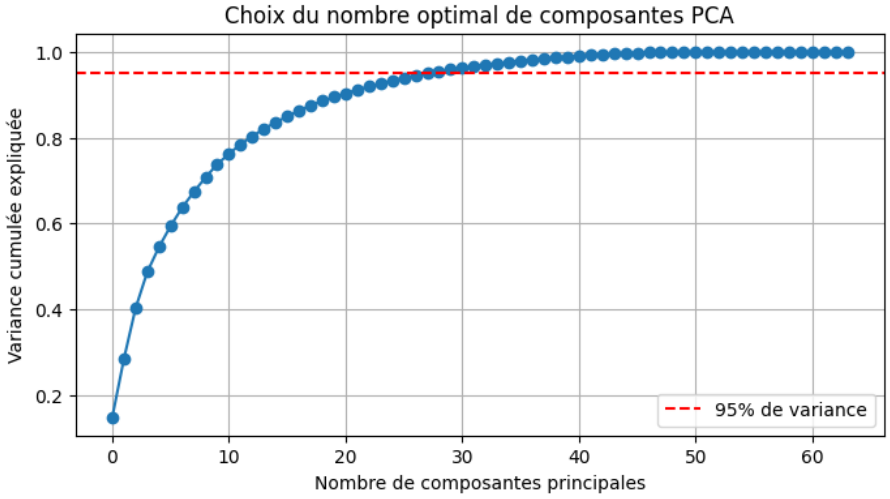


FIGURE 8 – Variance expliquée en fonction du nombre de composantes dans la PCA

La variance expliquée est la variance cumulée des données que l'on conserve en projetant sur certain nombre de composantes principales. Chaque point du graphique indique la quantité d'information (variance) capturée par les premières composantes jusqu'à ce nombre donné.

Avec ce graphique, nous fixons un seuil à 95% pour choisir le nombre minimal de composantes principales nécessaires pour conserver 95% de la variance afin d'être sûr que la majorité des caractéristiques importantes des données sont conservées.



On remarque donc que ce seuil est atteint pour un nombre de composantes égal à 28. On conservera 28 composantes pour le PCA dans la suite du projet.

4.4 Filtre de Sobel

Nous appliquons un filtre W, soit le filtre de Sobel, qui détecte les bords et les courbes. Il repose sur une opération avec des matrices de convolution appliquées dans les deux directions x et y.
Cela permet de calculer deux gradients G_x et G_y, et de la formule :

$$G = \sqrt{G_x^2 + G_y^2}.$$

Nous appliquons ce filtre à chaque chiffre manuscrit du jeu de données. Soit A la matrice qui représente l'image source.
On a :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

avec $*$ le produit de convolution. Par exemple, si on applique le filtre sur une image du chiffre 1, on aura :

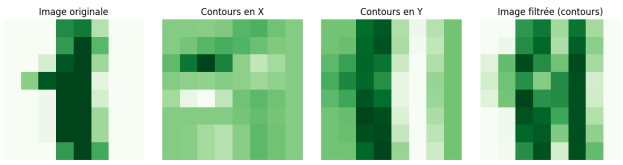


FIGURE 9 – Visualisation des contours avec le filtre

On applique d'abord le filtre Sobel sur l'axe x, (image 2) pour montrer les variations horizontales des intensités, puis on l'applique sur l'axe y (image 3) pour montrer les variations verticales. On observe bien que le filtre met en évidence les changements brusques, ce qui permet de repérer les bords et les formes caractéristiques de l'image du 1.
Ce filtre permet donc d'identifier les contours des chiffres pour distinguer les formes. Cela améliore la différenciation entre les chiffres qui se ressemblent, par exemple le 6 et le 8.

4.5 Partitionnement en zones

Pour mieux capter la forme des chiffres manuscrits, nous avons appliqué une méthode de découpage des images en trois zones horizontales :

— Zone du haut : lignes 1 à 3
— Zone au centre : lignes 4 et 5
— Zone du bas : lignes 6 à 8

Pour chaque zone, nous calculons la moyenne de l'intensité des pixels. On obtient trois nouvelles features (colonnes) avec les moyennes des intensités dans chacune des zones.

Cela permet de détecter les motifs particuliers dans chaque sous-partie de l'image, comme les boucles dans le haut ou le bas (6, 8, 9), les barres horizontales (5, 7), les lignes verticales sur plusieurs zones (1, 4).

Voici un exemple de ce que nous obtenons pour un chiffre 6 pour l'intensité moyenne de chaque zone :

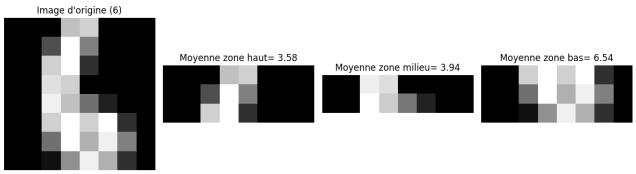


FIGURE 10 – Exemple du filtre sur un 6

5 Division des données

On va maintenant entraîner le modèle. Pour cela, on procède au découpage de nos données : on commence par prendre 80% du set pour faire un set d'apprentissage et 20% pour faire un set de test.
On réalise cela avec la fonction

```
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2,random_state=42)
```

Le 0.2 signifie qu'il y a 20% dans le set de test et le random state correspond à la graine (reproductibilité). Faire cela permet de limiter le sur-apprentissage. En séparant les données, on peut tester le modèle sur des exemples nouveaux, ce qui donne une estimation plus fiable des performances.

Distributions équilibrées

On vérifie maintenant que les distributions dans chaque classe sont équilibrées dans les ensembles d'apprentissage et de test :

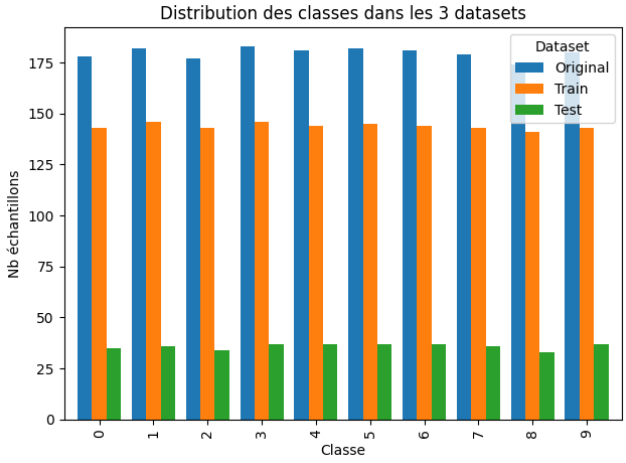


FIGURE 11 – Disribution dans les trois sets

Cependant, cette figure ne nous permet pas de bien voir si les données sont équilibrées puisque les datas ne contiennent pas le même nombre de données. Pour une meilleure visualisation, nous normalisons le nombre d'échantillons afin d'avoir plutôt des pourcentages que des chiffres. Voici ce que l'on obtient après normalisation :

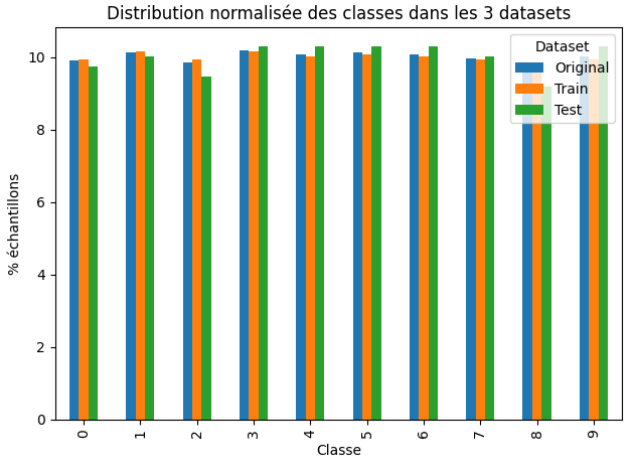


FIGURE 12 – Distribution dans les trois sets après normalisation

Sur ce graphique, on voit bien que la répartition est équilibrée dans chaque set.

6 Méthode de classification supervisée

6.1 SVM

Le SVM (support vector machine) est un algorithme de classification conçu pour séparer les classes de données en deux classes par une frontière appelée hyperplan. Il cherche à maximiser la distance entre l'hyperplan et les points les plus proches de chaque classe.

FIGURE 13 – schéma du SVM

6.1.1 SVM à Noyau linéaire

Un SVM à noyau linéaire ne transforme pas les données (i.e : n'applique pas une projection des données sur un espace de plus grande dimension). Il cherche à tracer une droite qui sépare les classes des données. Avec ce noyau, SVM est simple et rapide mais il ne peut travailler qu'avec des données linéairement séparables.

Construction de la pipeline

Dans cette partie, nous mettons en place un pipeline qui intègre plusieurs étapes essentielles au traitement de nos données avant la classification finale.

Voici la ligne de code que nous avons utilisée pour créer la pipeline :

```
clf_rbf = Pipeline([('features', all_features),('scaler', StandardScaler()),
('svc',SVC(kernel='linear',random_state=42))])
```

— StandardScaler : Ce composant permet de normaliser les données. Il est particulièrement utile pour des algorithmes sensibles à l'échelle des variables comme la PCA.

— Features : Cette étape regroupe les différentes features que nous avons extraites précédemment (zonal features, filtre de Sobel, et PCA). Ces caractéristiques sont concaténées pour enrichir l'information pour la classification.

— MinMaxScaler : Une fois les features concaténées, elles sont de nouveau normalisées avec un MinMaxScaler. Cette opération est cruciale car les différentes features n'ont pas les mêmes échelles initiales, et un classifieur comme le SVM est sensible à la distribution des données.

— SVC(kernel='linear') : Enfin, nous utilisons un SVM(Support Vector Machine) avec noyau linéaire comme classifieur. Ce choix est motivé par la performance du SVM sur les données linéairement séparableset par sa rapidité d'exécution.

L'utilisation du pipeline a plusieurs avantages : cela permet d'automatisation du processus d'entraînement et de prédiction, de protéger les données contre les fuites d'information entre l'entraînement et le test et elle est facile à modifier pour tester d'autres transformations ou classifieurs.

6.1.2 SVM à Noyau RBF

En revanche le noyau RBF nous permet de projeter les données sur un espace de grande dimension, ce qui nous permet de tracer un hyperplan non linéaire.

6.1.3 Matrices de confusion pour les SVM

On regarde maintenant les performances de notre modèle pour savoir si les erreurs de prédiction sont fréquentes et quels chiffres sont confondus avec d'autres. Pour cela, on a fait des matrices de confusion sur les train et test sets.
Ces matrices montrent les résultats du test sur les colonnes et les valeurs attendues sur lignes. Les valeurs de la diagonales sont le nombre de chiffres qui sont bien reconnus. Les autres valeurs sont les faux positifs (pour la partie triangulaire inférieure) et les vrais négatifs (partie triangulaire supérieure).

Tout d'abord, pour le SVM à noyau linéaire, on obtient les matrices suivantes :

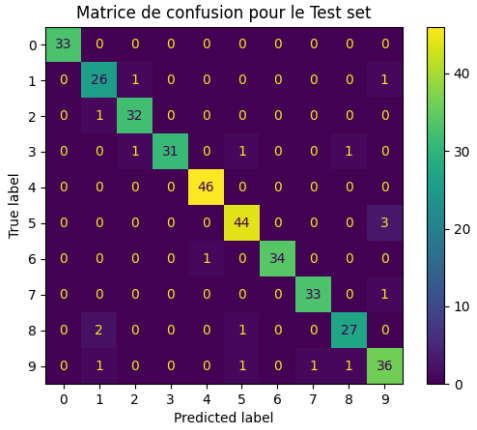


FIGURE 14 – Matrice de confusion sur le test set

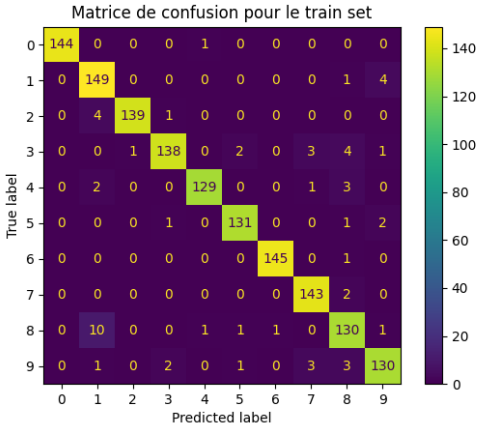


FIGURE 15 – Matrice de confusion sur le train set

Ces matrices nous permettent de dire que le modèle est fiable : en effet, il y a assez peu de faux positifs ou vrais négatifs. On peut également voir quels chiffres sont confondus avec quels autres : le 8 est confondu avec le 1, le 3 avec le 8. Pour le test set, sur 360 images, la machine se trompe 18 fois. On a un score de précision de 0.975 sur le test set et de 0.9874 sur le train set.

A présent, pour le SVM à noyau RBF, on obtient les matrices suivantes :

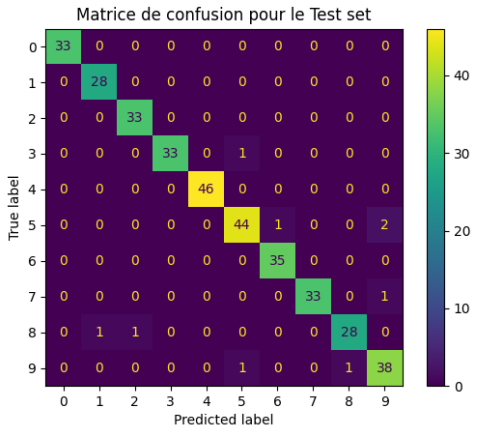


FIGURE 16 – Matrice de confusion sur le test set

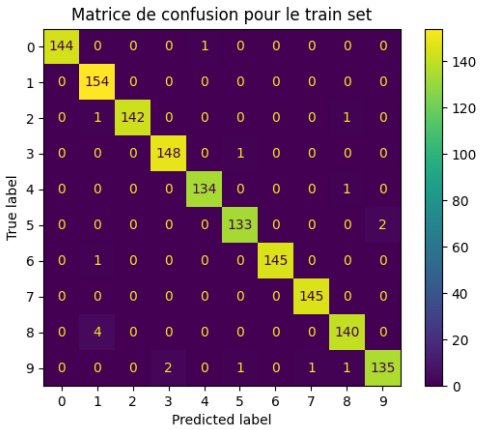


FIGURE 17 – Matrice de confusion sur le train set

Les observations sont similaires, mis à part que la précision a augmenté. En effet, pour le test set, sur 360 images, la machine se trompe 9 fois. On a un score de précision de 0.975 sur le test set et de 0.9882 sur le train set.

On en déduit qu'avec un noyau RBF, la mthode SVM permet une meilleure fiabilité

6.2 Optimisation des hyperparamètres

Pour avoir les meilleurs résultats possibles on a utilisé la fonction GridsearchCV. Cette fonction effectue une recherche automatique de la meilleure combinaison de paramètres possible.
Les paramètres testés sont :

- Nombre de composantes PCA : Ce paramètre correspond au nombre de dimensions conservé après la réduction de dimension avec la PCA. Il permet de réduire la complexité tout en conservant un maximum de variance de l'information de départ.
- Paramètre C du SVM : Ce paramètre contrôle le compromis entre la maximisation de la marge et la minimisation des erreurs de classification sur les données d'entraînement. Une valeur faible favorise une marge plus large (mais avec plus d'erreurs), tandis qu'une valeur élevée tente de minimiser les erreurs mais avec une marge plus petite.
- Paramètre γ : Ce paramètre est propre aux SVM avec noyau non linéaire (par exemple RBF). Il détermine l'influence d'un point d'entraînement sur la frontière de décision

Voici la meilleure combinaison qu'on a obtenue :

```
Nb features computed:  32
Accuracy of the SVC on the test set:  0.95
Accuracy of the SVC on the train set:  0.9582463465553236
Fitting 5 folds for each of 64 candidates, totalling 320 fits
Best parameters: {'features__pca__n_components': 20, 'svc__C': 10, 'svc__gamma': 0.01}
Best cross-validation score: 0.9888695315524585
Accuracy on test set: 0.9861111111111112
```

Le meilleur score de précision que nous avons obtenu est donc 0.986.
Cela confirme que le modèle SVM avec noyau RBF et les paramètres bien réglés permettent une excellente capacité de généralisation et de prédiction.

Taille des sets de train et test

Depuis le début du projet, nous avons pris 80% des données pour faire l'échantillon d'entraînement et 20% pour les tests. On se demande si la taille du set de test influence les performances. Pour cela, on trace le coefficient de précision en fonction de la taille de l'échantillon de test.

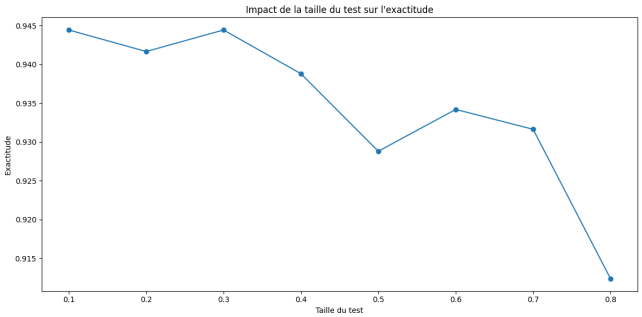


FIGURE 18 – Impact de la taille de l'échantillon de test sur les perfomances du modèle

Le graphique montre que les meilleures performances sont obtenues pour des valeurs entre 20% et 30%, ceci maximise la précision tout en garantissant une évaluation représentative. On conserve donc notre test avec 20% des données car il permet d'avoir une bonne précision.

6.3 Validation croisée (k-cross validation)

Enfin, afin d'évaluer la robustesse du modèle SVM avec noyau RBF et d'optimiser ses hyperparamètres (C, γ, n components de la PCA), nous avons utilisé la méthode de validation croisée K-Fold à l'aide de GridSearchCV. La validation croisée consiste à diviser le jeu d'entraînement en K sous-ensembles. Le modèle est entraîné sur K-1 de ces sous-ensembles et validé sur le dernier, de manière tournante. Les valeurs testées sont : $K = 3$, $K = 5$, et $K = 10$.

```
Running GridSearchCV with K=3-fold cross-validation:
- Best CV score with K=3: 0.9861
- Best parameters with K=3: {'features__pca__n_components': 30, 'svc__C': 10, 'svc__gamma': 0.01}

Running GridSearchCV with K=5-fold cross-validation:
- Best CV score with K=5: 0.9889
- Best parameters with K=5: {'features__pca__n_components': 20, 'svc__C': 10, 'svc__gamma': 0.01}

Running GridSearchCV with K=10-fold cross-validation:
- Best CV score with K=10: 0.9882
- Best parameters with K=10: {'features__pca__n_components': 30, 'svc__C': 10, 'svc__gamma': 0.01}
```

En observant ces résultats, on remarque que les valeurs de C et γ restent les mêmes pour les 3 valeurs de K, ce qui montre la stabilité du modèle. Le meilleur score est obtenu avec $K = 5$. Mais l'écart est faible, ce qui suggère que le modèle est peu sensible au découpage.

6.4 One-vs-Ones (OvO) et One-vs-Rest (OvR)

Afin de gérer les différentes classes (labels) de notre jeu de données, nous pouvons mettre en place deux stratégies : OvO et OvR.

La stratégie OvR, One vs Rest, a pour but d'entraîner une classe face à toutes les autres, par exemple : est-ce que c'est un 8 ou non ? Cette stratégie est une approche simple, rapide et efficace, notamment si notre jeu de données est grand (du point de vue du nombre de classes). On notera que des classes déséquilibrées peuvent impacter les résultats avec cette stratégie.

La stratégie OvO, One vs One, consiste à entraîner un classifieur pour chaque paire de classes possibles. Par exemple : est-ce un 3 ou un 7 ? Ainsi, si nous avons 10 classes, cela donnera lieu à 45 modèles différents. Chaque modèle est spécialisé dans la distinction entre deux classes. Cette méthode peut donner de très bons résultats, notamment lorsque les classes sont proches les unes des autres, car elle permet des décisions plus fines. Cependant, elle demande plus de ressources en calcul et en mémoire, ce qui peut poser problème si le nombre de classes est très élevé.

Dans notre code, nous faisons appel à SVC, qui utilise de base un OVO, mais on peut modifier "l'enrobage" pour en faire un OVO ou un OVR, puis comparer leurs performances.

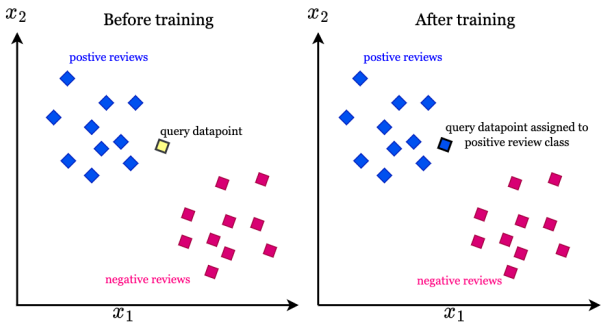
```
OVR :
  Accuracy train : 0.9408
  Accuracy test  : 0.9361
  Temps d'entraînement : 0.2087 s

OVO :
  Accuracy train : 0.9569
  Accuracy test  : 0.9500
  Temps d'entraînement : 0.1806 s
```

FIGURE 19 – OvR vs OvO

Les résultats sont les mêmes que ceux attendus : l'OVR est moins précis que l'OVO, mais il est plus rapide. On notera que l'OVO a la même précision que le SVC seul, ce qui est logique, car le SVC de scikit-learn fait par défaut appel à une stratégie OVO.

6.5 Méthode de classification des k-NN

La méthode K-NN est une méthode qui consiste à classer l'entrée dans la catégorie à laquelle appartiennent les k plus proches voisins. Cette méthode ne nécessite pas d'entraînement, ce qui la rend simple à implémenter mais elle peut être coûteuse en calcul.



Nous obtenons un bon score de précision :

```
Précision (accuracy) sur le test set avec K-NN : 0.9722
```

Le K-NN nous a donc donné de bons résultats, proches de celles de SVM avec noyau RBF. Son principal avantage est sa simplicité, néanmoins le coût de calcul reste relativement élevé car chaque test nécessite la comparaison aux données d'entraînement.

7 Réseau de neurones

Lors de cette dernière partie, nous allons étudier les réseaux de neurones dans le cadre du machine learning. Un réseau de neurones se compose de plusieurs couches de nœuds, comme montré sur le schéma ci-dessous :

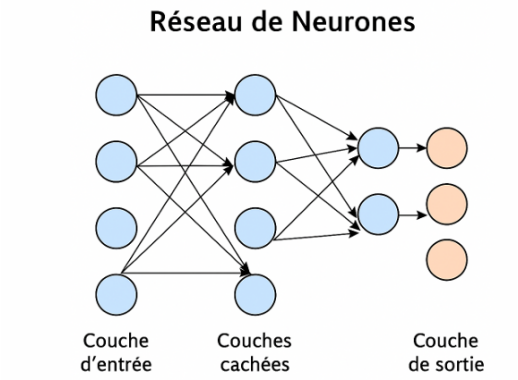


FIGURE 20 – Schéma d'un réseau de neurones

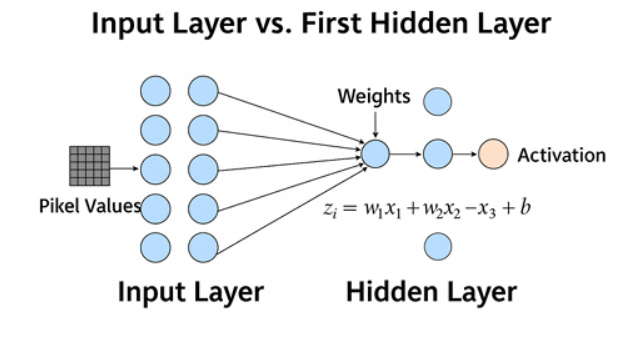
La première couche prend en entrée un type d'information que l'on doit choisir (nous discuterons les différents choix à prendre en entrée). Elle diffuse et transforme ensuite l'information brute d'étape en étape, de couche en couche. Lorsqu'elle arrive dans la dernière couche de neurones (le nombre de neurones de cette dernière couche doit correspondre au nombre de labels de notre base de données), le réseau compare sa réponse à la réponse attendue, puis il rétro-propage l'information afin de réajuster les coefficients de calcul pour obtenir une réponse plus précise. Un cycle d'apprentissage de cette forme est appelé une "epoch".

Dans un premier temps, nous avons utilisé la bibliothèque TensorFlow avec Keras pour construire et entraîner notre réseau de neurones. Ce réseau est composé de :

— une couche d'entrée de 64 neurones, correspondant aux 64 pixels de chaque image 8×8 ;

— une couche cachée de 32 neurones, avec la fonction d'activation ReLU ;

— une couche de sortie composée de 10 neurones, avec la fonction d'activation softmax, correspondant aux 10 chiffres (0 à 9).



Comparons maintenant les différents résultats en fonction des paramètres ou des données passées en entrée.

Si l'on passe en entrée les 64 pixels des images (soit l'image comme elle est avant toute transformation dans la base), avec une première couche de 32 neurones, et une couche de sortie de 10 neurones (correspondant aux chiffres 0 à 9), pour 10 epochs, on obtient les résultats suivants :

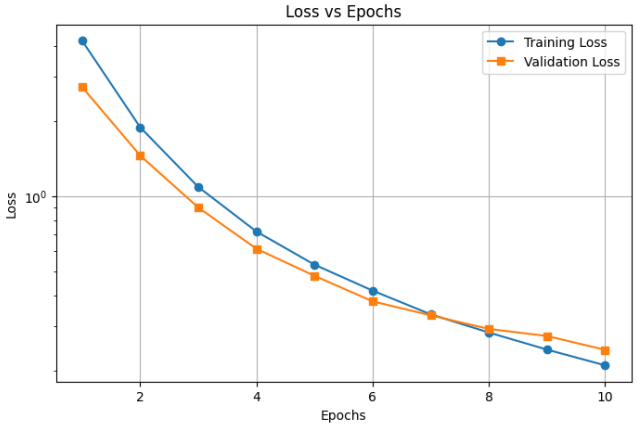


FIGURE 21 – Loss vs Epoch (10 epochs)

Si on met maintenant 100 epochs, on obtient ceci :

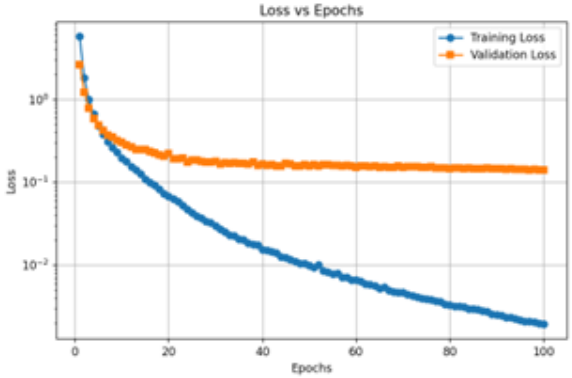


FIGURE 22 – Loss vs Epoch (100 epochs)

On peut voir que le loss training tombe assez nettement aux alentours de 10, tandis que le loss validation stagne assez vite vers 10.

On tente ensuite avec 4 couches intermédiaires de 32, 25, 20, 15 neurones :

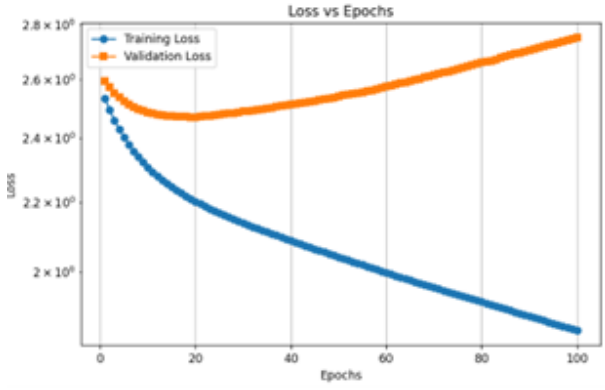


FIGURE 25 – Loss vs Epoch avec les données prétraitées (RBF)

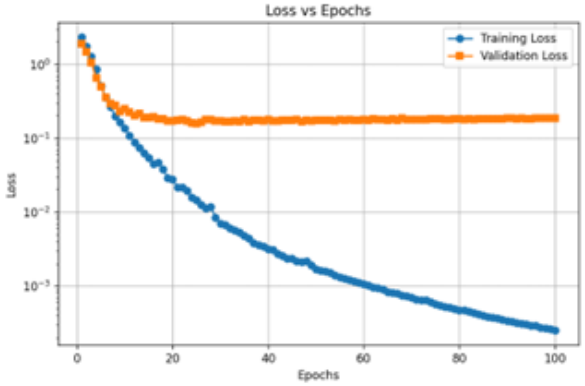


FIGURE 23 – Loss vs Epoch avec 4 couches intermédiaires de neurones (32, 25, 20, 15)

Les résultats semblent assez similaires, à part qu'à nombre égal d'epochs, le training loss est plus faible : on peut parler de surapprentissage.

On essaye ensuite avec des données prétraitées : tout d'abord, les données nettoyées avec les features demandées dans les premières séances (zonal, edges et PCA).

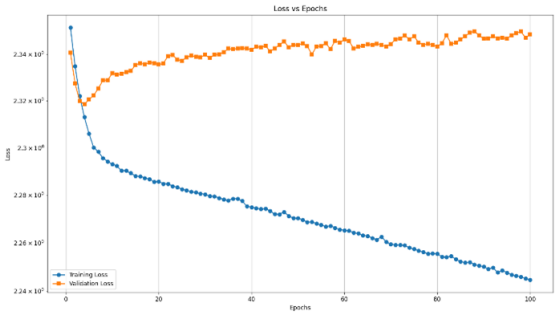


FIGURE 24 – Loss vs Epoch sur les données prétraitées

Les résultats sont moins bons qu'avec les images brutes. On observe encore une fois un phénomène de surapprentissage (avec cette fois une augmentation du validation loss). On a le même type de résultats pour nos données prétraitées avec le RBF.

8 Conclusion

Ce projet nous a permis de découvrir et de comprendre plusieurs approches de classification supervisée, appliquées à la reconnaissance de chiffres à partir d'images faites à la main.
Après une phase de prétraitement, d'analyse et de construction de features pour décrire avec plus de précision

l'image (grâce à l'ACP, le découpage en zones et le filtre de Sobel), nous avons conçu des pipelines de machine learning à partir de méthodes SVM à noyau linéaire dans un premier lieu, puis RBF.
Nous avons ensuite optimisé ces modèles en choisissant les meilleurs hyperparamètres, et évalués en utilisant différentes stratégies de classification multi-classes telles que One-vs-One (OvO) ou One-vs-Rest (OvR).
Pour finir, nous en avons appris davantage sur les réseaux de neurones, leur fonctionnement ainsi que leur mise en place en Python à l'aide de Keras. Ce projet nous a permis de renforcer nos compétences pratiques en programmation Python et nos connaissances sur le machine learning et l'apprentissage automatique. Nous avons aussi appris à comparer plusieurs approches, à optimiser des paramètres et nous avons eu un apperçu des différentes étapes nécessaires au développement de modèles efficaces.

9 Annexe

9.1 Répartition des tâches

Mardi : $\approx 8h$

Etan : SSIM
Comparaison de deux images
Visualisation SSIM PCA
Filtre Sobel

Perline : Visualisation des données
Data preprocessing
Visualisation PCA erreur
Filtre Sobel

Khaoula : PCA
variance cumulée
choix du nombre de composantes pour le PCA
Découpage en 3 zones
Filtre Sobel
Concaténation

Mercredi : $\approx 8h$

Etan : OvO
OvR
Comparaison des temps
Essai du programme avec un chiffre qui n'est pas dans la database (dessiné à la main

Perline : Train/test split et distributions
Dataset split summary
Matrices de confusion
Interprétations
Rapport

Khaoula : Fonctions pour les features
FeatureUnion, pipeline
SVC
Etude de la variation de la taille de l'échantillon de test
Cross validation

Jeudi : $\approx 6h$

Les 3 : Réseau de neurones avec keras et tensorflow
Plots loss vs epochs
Script pour l'oral
Diapo
Rédaction du rapport