Universite de Montpellier

Rapport de projet

Chemins spécifiques pour la classification dans les réseaux de neurones profonds

Belkassim BOUZIDI
Mélissa DADI
Chakib ELHOUITI
Massili KEZZOUL
Ramzi ZEROUAL

Encadrant :
M. Pascal Poncelet

25 mai 2021

Remerciements

Tout d'abord nous souhaitons adresser nos remerciements au corps professoral et administratif de la faculté des sciences de Montpellier qui déploient des efforts pour assurer à leurs étudiants une formation actualisée.

En second lieu, nous tenons à remercier notre encadrant M. Pascal Poncelet pour ses précieux conseils et son aide durant toute la période de travail.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre projet en acceptant d'examiner notre travail.

Nous remercions M. Yahia Zeroual pour sa relecture attentive de ce rapport.

Table des matières

Chapitre 1

Introduction au sujet

1.1 Les réseaux de neurones pronfonds

En termes simples, le Deep Learning est le domaine où les machines apprennent par elles-mêmes en imitant le cerveau humain. Imiter dans le sens où les machines peuvent effectuer des tâches nécessitant une intelligence humaine. Maintenant, comprenons comment ?

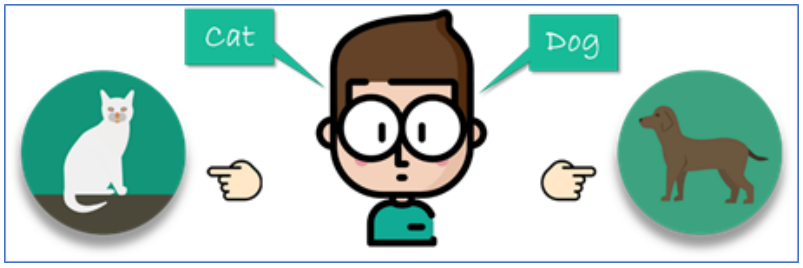

Schéma 1.1 –

Le cerveau humain peut facilement différencier un chat et un chien.

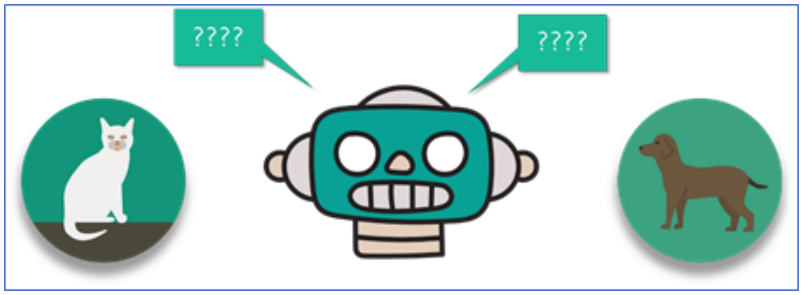

Schéma 1.2 –

Mais comment faire différencier une machine entre un chat et un chien ?

La majorité des méthodes d'apprentissage profond utilisent des architectures de réseau neuronal et c'est pourquoi les modèles d'apprentissage profond sont également connus sous le nom de réseaux neuronaux profonds. Un processus d'apprentissage en profondeur se compose de deux phases clé : l'entraînement et l'inférence.

La phase d'entraînement peut être considérée comme un processus d'étiquetage d'énormes quantités de données et d'identification de leurs caractéristiques correspondantes. Ici, le système

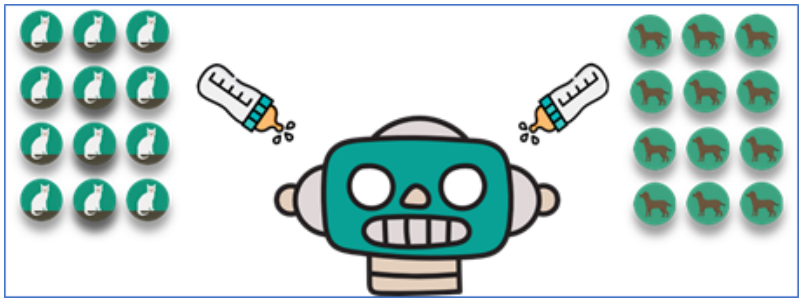

Schéma 1.3 –

compare ces caractéristiques et les mémorise pour arriver à des conclusions correctes lorsqu'il rencontre des données similaires la prochaine fois.

Au cours de la phase d'inférence, le modèle tire des conclusions et étiquette les données non exposées à l'aide des connaissances acquises précédemment.

Des exemples : Un grand nombre d'industries utilisent le deep learning pour tirer parti de ses avantages. Jetons un coup d'œil à quelques-uns d'entre eux.

Électronique l'apprentissage en profondeur est utilisé dans la traduction automatique de la parole. Vous pouvez penser aux appareils d'assistance à domicile qui répondent à votre voix et comprennent vos préférences.

Conduite automatisée grâce à l'apprentissage en profondeur, les chercheurs automobiles sont désormais en mesure de détecter automatiquement des objets tels que les feux de signalisation, les panneaux d'arrêt, etc. Ils l'utilisent également pour détecter les piétons, ce qui contribue à réduire les accidents.

Recherche médicale l'apprentissage en profondeur est utilisé par les chercheurs en cancérologie pour détecter automatiquement les cellules cancéreuses.

Si les réseaux de neurones profonds obtiennent souvent des bons résultats, ils ont cependant un point faible : leur fonctionnement apparaît comme bien plus opaque. Un phénomène appelé « boîte noire » (« black box » en anglais), dans le sens où l'on peut juger des données qui entrent dans la boîte et des résultats qui en sortent, mais sans savoir ce qui se passe à l'intérieur.

1.2 Jeu de données

Nous avons choisi d'utiliser la base de données MNIST [1] comme jeu de données. L'ensemble de données MNIST est l'un des ensembles de données les plus couramment utilisés pour la classification d'images et accessible à partir de nombreuses sources différentes. Tensorflow nous permet d'importer et de télécharger le jeu de données directement depuis leur API.

La base de données MNIST, une extension de la base de données NIST, est une collection de données de faible complexité de chiffres manuscrits utilisés pour entraîner divers algorithmes d'apprentissage automatique supervisées. MNIST est considéré comme le *Hello world* de l'apprentissage automatique.

MNIST est un ensemble de données étiqueté qui associe des images de chiffres écrits à la main avec la valeur du chiffre respectif. Elle contient 70 000 images en noir et blanc de 28 x 28 pixels représentant des chiffres de zéro à neuf. Chaque pixel de l'image est un entier de valeur comprise entre 0 et 255 représentant le niveau de gris.

1. Modified ou Mixed National Institute of Standards and Technology

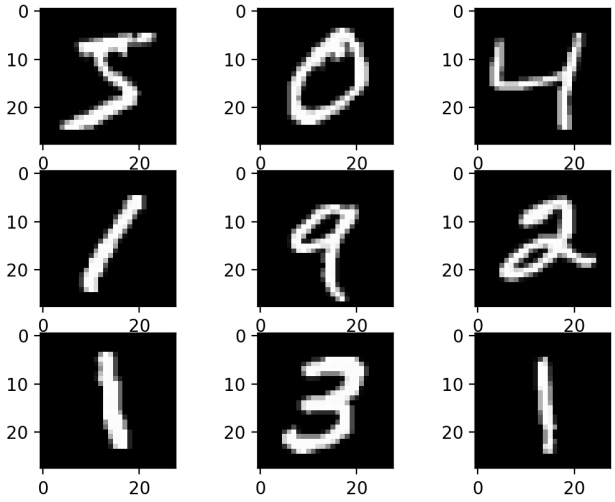

Schéma 1.4 – Exemple de quelques éléments de la base de données MNIST

1.3 Problèmatique

L'objectif du projet est d'essayer de mieux comprendre le fonctionnement interne d'un réseau de neurones. Il s'agit de repérer, selon les données d'entrée, des signatures d'activation de neurones. Une signature correspond au pattern (ou chemin) emprunté par un objet au sein du réseau avant d'arriver à une conclusion. Concrètement, on veut analyser les différentes signatures obtenues par différents types de données et de modèles.

Par exemple, si on entraîne un modèle à reconnaître des images de *1* et de *7*, on voudrait répondre aux questions suivantes :

— À partir de quelle couche le modèle change de comportement pour reconnaître une image ?
— Les signatures des images de *7*, sont-elles différentes de ceux des *1* ?
— Si on passe une image de *3* au modèle, à quoi va ressembler sa signature ?

1.4 Solution proposée

Dans un premier temps, on doit se familiariser avec la base de données MNIST et développer plusieurs outils afin de les manipuler. Ensuite, nous passerons à la construction des modèles d'apprentissage pour pouvoir analyser leurs comportements internes en récupérant les soties de chaque couche cachée. À partir de ces sorties, l'objectif sera d'extraire les signatures grâce à des algorithmes de *clustering*[2]. Enfin, nous réaliserons une interface de visualisation afin d'analyser les résultats et de répondre aux questions posées ci-dessus.

2. Méthode en analyse des données. Elle vise à diviser un ensemble de données en différents paquets (*clusters*) tel que chaque sous-ensemble partagent des caractéristiques communes.

Chapitre 2

Organisation du projet

2.1 Méthodes d'organisation

Afin de mener à bien le développement du projet, nous avons décidé de travailler un maximum de temps ensemble et de manière très régulière. Nous nous sommes réunis deux à trois fois par semaine, en vue de faire le point sur l'avancement du projet et de définir les objectifs restants à atteindre.

Ainsi, selon l'état de progression du projet, nous réalisâmes les tâches en retard durant le week-end pour ne pas cumuler de retard et respecter l'intégralité du cahier des charges.

Toutes les deux semaines, nous nous sommes réunis avec notre encadrant, M. Pascal Poncelet. Lors de ces réunions de mises au point relatif au projet, de précieux conseils nous furent prodigués.

2.2 Découpage du projet

Nous avons découpé la réalisation du projet en trois grandes phases :

2.2.1 Phase d'analyse des données

Durant cette étape, nous nous sommes concentrés sur l'analyse des données que nous allions utiliser. Notamment l'étude de leur structure ainsi que la définition des différents outils utiles pour leur manipulation. Nous avons également choisi les outils de travail collaboratifs et les principales technologies utilisées.

2.2.2 Phase de développement

Durant cette phase, nous avons commencé à implémenter plusieurs modèles d'apprentissage automatique, les outils d'extraction des informations internes à ce dernier ainsi que les interfaces de visualisation des résultats.

2.2.3 Phase d'analyse et présentation des résultats

Cette étape a consisté en l'analyse des résultats obtenues durant la phase précédente. Nous nous sommes aussi penché sur la réalisation d'une interface web afin d'effectuer des expérimentations et de pouvoir, le plus intuitivement possible, visualiser les résultats.

2.3 Outils de collaboration

Afin de s'organiser, nous avons décidé d'utiliser le logiciel *Git* à travers le serveur *Github*. En effet le logiciel libre a facilité grandement la collaboration entre nous. Vous trouverez d'ailleurs l'intégralité du code source sur ce depôt Github.

En ce qui concerne la rédaction de ce rapport, nous avons utilisé LaTeX, système de composition de documents créé par Leslie Lamport, pour faciliter la rédaction à plusieurs.

Schéma 2.1 – Logo du GitLab

Schéma 2.2 – Logo de Latex

Chapitre 3

Analyse des données

En récupérant les données brutes de *MNIST*, qui sont des images de 28 x 28 pixels stockées sous la forme d'une matrice de 2 dimensions, avec une valeur entre 0 et 255 pour chaque case. Il est clair **qu'on ne peut pas** passer ces données à un réseau de neurones, vu que le réseau de neurones prends en input layer un tableau à une seule dimension. Pour pouvoir traiter, entraîner et visualiser ces données, on a mis en place une série de prétraitements.

3.1 Prétraitements

3.1.1 Découpage des données

Cette partie se résume par la fonction *cut_ data*, qui nous permet de découper les données et garder un nombre précis d'images pour un ensemble de chiffres définis. Par exemple, on peut garder 100 images représentants des 0 et des 1 (100 pour chaque chiffre). On a réalisé cette fonction pour faciliter la phase de développement et de pouvoir mieux visualiser les résultats sur un petit ensemble de données.

3.1.2 Scaling, Flattening

La phase de prétraitements est la plus importante dans le processus de création d'un réseau neuronal. Pour cela, on a mis en place la fonction *normalize_ dataset* qui fait, en premier du Scaling (standardisation ou normalisation et c'est ce qu'on a utilisé dans notre cas), qui consiste à mettre les valeurs des images entre 0 et 1 au lieu de 0 et 255, cela va permettre à notre modèle de converger rapidement. Ensuite, on a aplati les images pour avoir un tableau d'une seule dimension au lieu d'une matrice à deux dimensions (du Flattening). En dernier on a transformé les labels correspondant à chaque image en un vecteur contenant que des 0 et des 1 et d'une taille égale au nombre de labels unique à garder (Ex : 1,3,7), on mettra un 1 à la case du label correspondant et des 0 aux autres cases, on commence en premier par trier les labels à garder pour avoir un ordre croissant avec les indices du vecteur. Par exemple, si on décide de garder que les images représentants les 1, les 3 et les 7, on aura des vecteurs de taille 3 de la forme suivante :

— Pour un 1 $\implies$ [1,0,0].
— Pour un 3 $\implies$ [0,1,0].
— Pour un 7 $\implies$ [0,0,1].

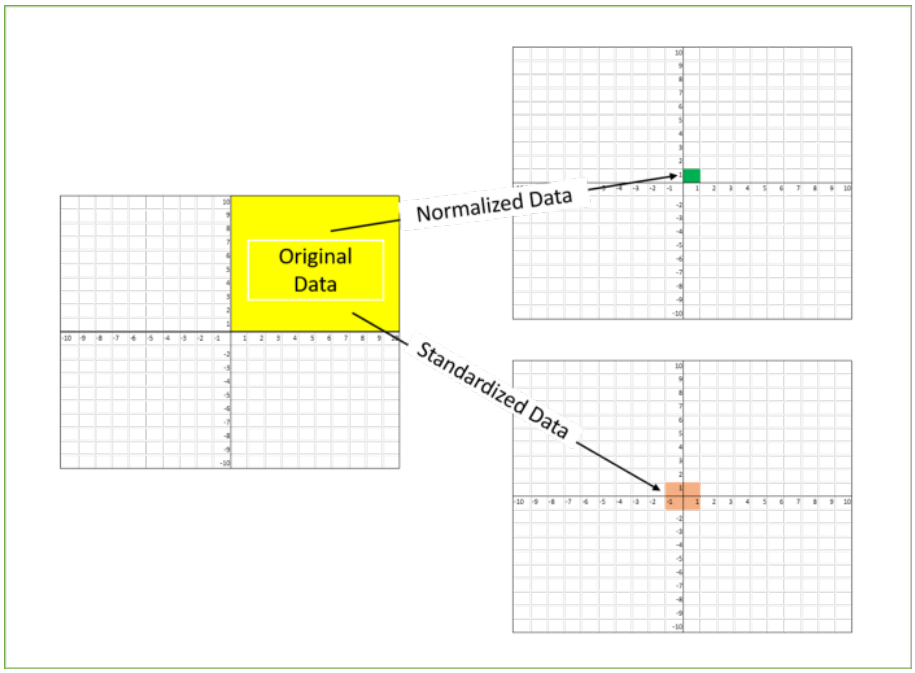

Schéma 3.1 – Scaling

Chapitre 4

Développement de l'architecture

4.1 Technologies utilisées

4.1.1 Jupyter notebook

Les notebooks Jupyter offrent un excellent moyen d'écrire et d'itérer sur du code Python. C'est un outil incroyablement puissant pour développer et présenter de manière interactive des projets de science de données. Jupyter notebook intègre le code et sa sortie dans un document unique qui combine des visualisations, du texte, des équations mathématiques et d'autres médias riches. Le flux de travail intuitif favorise un développement itératif et rapide, faisant de Jupyter notebook un choix de plus en plus populaire au cœur de la communauté de la science des données contemporaine et de l'analyse des données.

4.1.2 Tensorflow

Ce serait un défi de nos jours de trouver un ingénieur en apprentissage automatique qui n'a rien entendu sur TensorFlow. Initialement créé par l'équipe Google Brain à des fins internes, telles que le filtrage du spam sur Gmail, il est devenu open-source en 2015.

Tensorflow est souvent utilisé pour résoudre des problèmes d'apprentissage profond et pour la formation et l'évaluation des processus jusqu'au déploiement du modèle. Outre les objectifs d'apprentissage automatique, TensorFlow peut également être utilisé pour créer des simulations, basées sur des équations dérivées partielles. C'est pourquoi il est considéré comme un outil polyvalent pour les ingénieurs en apprentissage automatique.

4.1.3 Keras

Keras est une API de réseau neuronal open source écrite en Python. Il peut fonctionner sur plusieurs frameworks d'apprentissage en profondeur et d'apprentissage automatique, notamment TensorFlow, Microsoft Cognitive Tool (CNTK) et Theano. Keras est une interface plutôt qu'un framework autonome comme TensorFlow. Il offre des abstractions intuitives de haut niveau qui permettent une expérimentation rapide.

Keras est utilisé par environ 200 000 utilisateurs, allant des chercheurs universitaires et des ingénieurs des startups et des grandes entreprises aux étudiants diplômés et aux amateurs. Keras est utilisé chez Google, Netflix, Uber, Microsoft, Square et de nombreuses startups travaillant sur la grande variété de problèmes d'apprentissage automatique.

4.2 Modèles d'apprentissage

Avant toute chose, il nous faut commencer par définir un réseau de neurones. Nous avons eu le choix entre plusieurs types de réseau. Ils diffèrent par plusieurs paramètres :

— la topologie des connexions entre les neurones ;
— la fonction d'agrégation utilisée (somme pondérée, distance pseudo-euclidienne...) ;
— et bien d'autres paramètres qui nous ne serons pas très utiles dans le déroulé de ce projet.

Nous avons choisi de commencer par utiliser un réseau à maillage *dense* et avec des paramètres de base. Ceci afin de simplifier, dans un premier temps, les expérimentations. Un réseau à maillage *dense* est un réseau tel que chaque neurone d'une couche est relié à tous les neurones de la couche précédente et de la couche suivante.

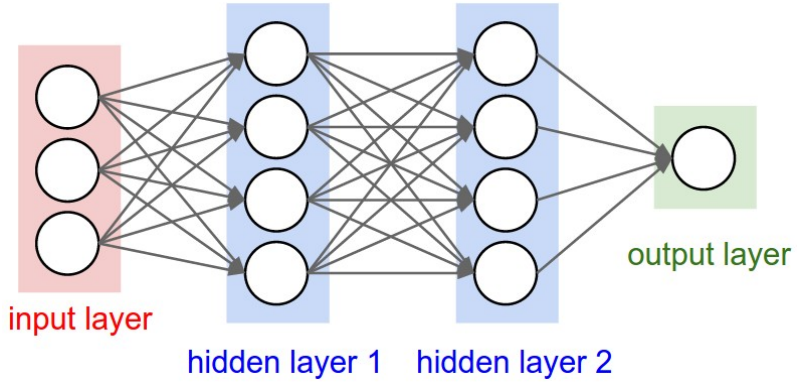

Schéma 4.1 – Réseau de neurones à maillage *dense*

Puis dans un deuxième temps, nous pourrons changer de modèle afin de passer à un réseau de neurones à convolution [1](ou CNN). Ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations. Les réseaux neuronaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo.

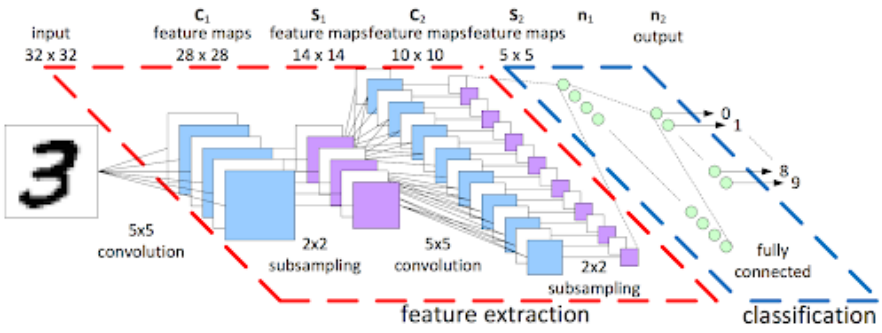

Schéma 4.2 – Réseau de neurones à convolution

Afin d'abstraire l'utilisation des réseaux de neurones dans notre Notebook Jupyter, nous avons défini une classe Python (nomée *MTQModel*) paramétrable qui se charge de construire un modèle, de l'entraîner et d'extraire les signatures. Point qu'on va aborder plus en détails dans la section qui suit.

1. Le CNN est un type de réseau de neurones artificiels acycliques, dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux.

4.3 Signature

Nous avons défini un réseau de neurones à maillage *dense* avec deux couches cachées, comme à la figure 4.1. La première couche cachée est composée de 32 neurones, la deuxième de 64. Une image qui traverse ce réseau laisse une signature S qu'on peut définir comme un vecteur H_1 de 32 valeurs, associées à la première couche et un vecteur H_2 de 64 valeurs, associées à la deuxième couche. La signature d'une image dans un réseau est donc définie ainsi : $S = (H_1, H_2)$[5]

Afin d'extraire la signature de chaque image, nous avons intégrer une méthode à notre classe Python *MTQModel*[2] qui permet de le faire automatiquement pour chaque image du *dataset*.

4.3.1 Clustering

Une fois la signature des images extraite, nous passons à l'analyse de ces dernières. Pour cela, nous utilisant un algorithme de *clustering*[3], le **K-means**. Ce dernier prend en paramètres les données et un certain K donnée par l'utilisateur, puis construit K clusters qui regroupent les données qui sont proches[4][3].

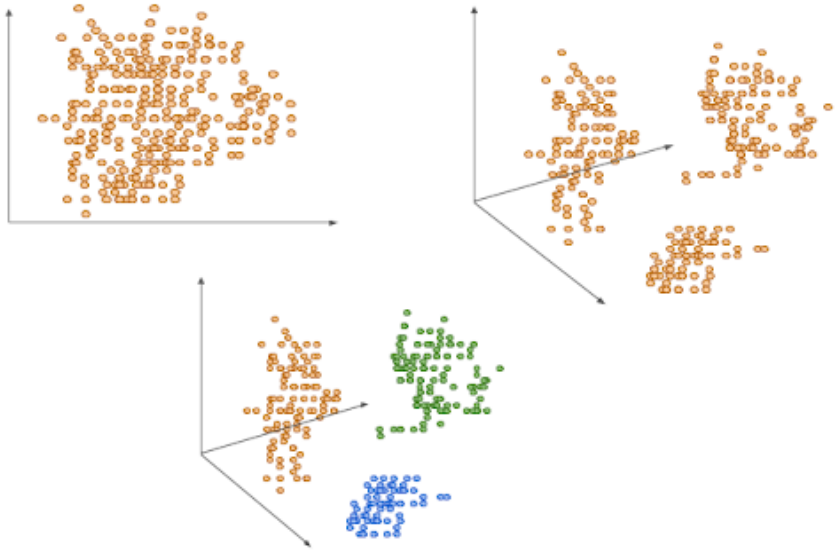

Schéma 4.3 – Clustering en action

Afin de choisir un K optimal, nous avons utilisé la méthode *Silhouette* qui consiste à calculer, pour chaque K de 2 à 10, la moyenne du score *Silhouette* du clustering. Le meilleur K étant celui qui donne la meilleure moyenne. Le score *Silhouette* $s(i)$ mesure combien un point est proche de son cluster comparer aux autres clusters[5]. Concrètement, le score d'un point i est $s(i) = \frac{b(i)-a(i)}{max(a(i),b(i))}$ avec $a(i)$ est la mesure de la similarité du point i avec son cluster et $b(i)$ le mesure de dissemblance du point i par rapport aux autres clusters.

4.4 Interface de visualisation

Pour pouvoir voir, analyser, évaluer et faire des expérimentations sur nos modèles, il faut avoir une interface de visualisation qui permet la projection des différentes visualisations des données et

2. La méthode en question s'appelle *get_hidden_layers_outputs(x)*
3. ou le Partitionnement de données en français
4. Proche en terme de distance euclidienne
5. C'est pour cela que K doit être supérieur ou égal à 2

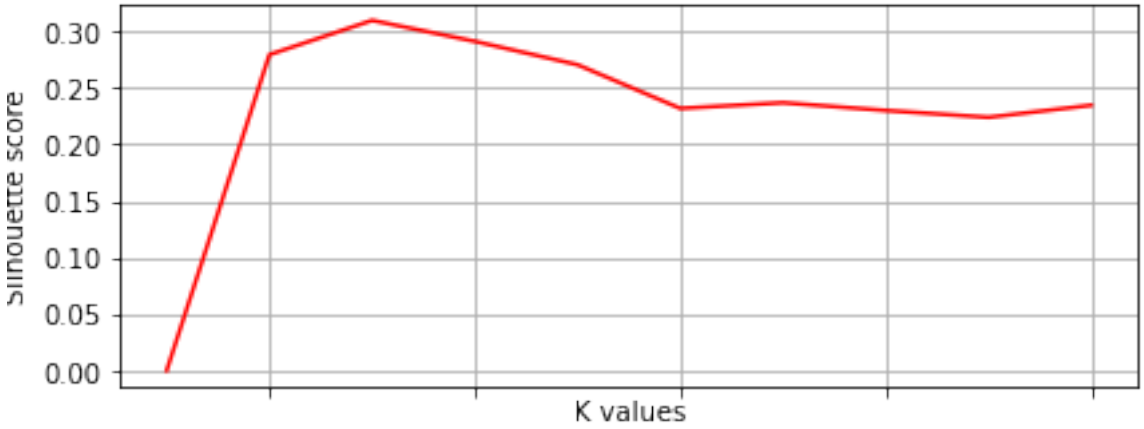

Schéma 4.4 – Le score silhouette

de pouvoir faire le lien entre ces visualisations. Notre but était de pouvoir projeter le résultat de nos extractions de signatures avec UMAP et le diagramme de Sankey et de pouvoir sélectionner un ou plusieurs éléments à partir de UMAP pour suivre sa ligne dans le diagramme de Sankey et vice-versa. Pour enfin réaliser une application web qui nous permet de faire plusieurs expérimentations, on va commencer par découvrir ce qui est UMAP et le diagramme de Sankey.[4]

4.4.1 UMAP (Uniform Manifold Approximation and Projection)

La réduction de dimensionnalité est un outil puissant permettant aux praticiens de l'apprentissage automatique de visualiser et de comprendre des ensembles de données volumineux et de grande dimension[2]. L'une des techniques de visualisation les plus largement utilisées est UMAP. UMAP, à sa base utilise des algorithmes de mise en page graphique pour organiser les données dans un espace de faible dimension. Dans le sens le plus simple, UMAP construit une représentation graphique de haute dimension des données puis l'optimise en un graphique de faible dimension pour être aussi structurellement similaire que possible. Dans notre cas, on va passer de 32 et 64 dimensions à 2 dimensions.[1]

4.4.2 Diagramme de Sankey

Un diagramme de Sankey est un type de diagramme de flux [6] dans lequel la largeur des flèches est proportionnelle au flux représenté. Les diagrammes de Sankey visualisent les contributions à un flux en définissant la source pour représenter le nœud source, la cible pour le nœud cible, la valeur pour définir le volume du flux et le label indiquant le nom du nœud. Dans notre cas, on a utilisé le diagramme de Sankey pour visualiser le chemin de chaque donnée (image), en démarrant de l'input layer, en passant par son cluster de chaque couche cachée du modèle et en arrivant à l'output layer (la prédiction).

4.4.3 Application web

Pour concevoir une application web, on a utilisé l'outil *Voilà*, qui permet de transformer un notebook jupyter en une application web autonome. Nous sommes parties d'un notebook présentant les différentes visualisations afin de générer une page web.

6. Un diagramme de flux est un type de représentation graphique destiné à présenter des données associant des entrées et des sorties, figurant des flux.

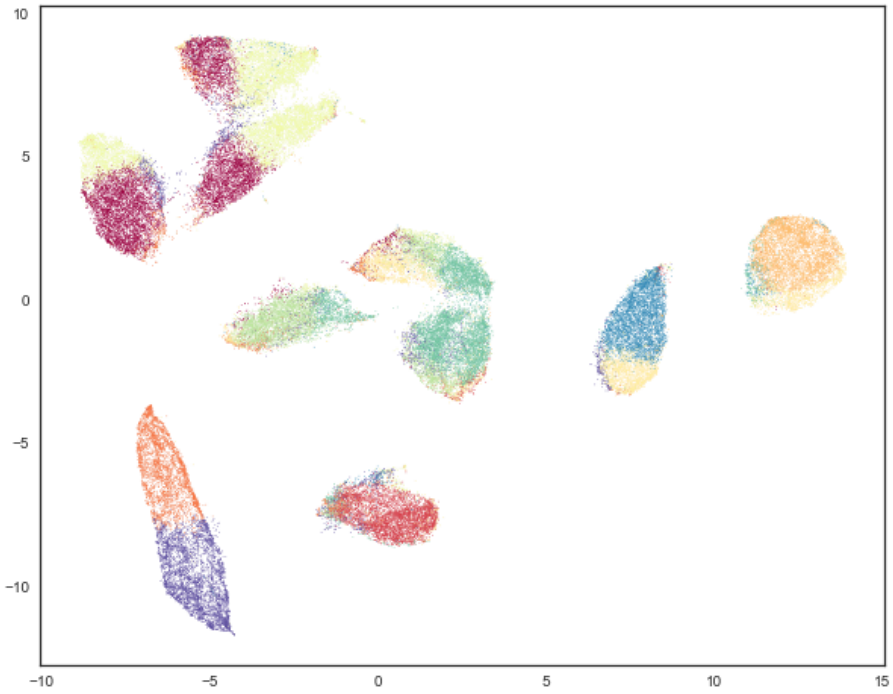

Schéma 4.5 – UMAP visualisation

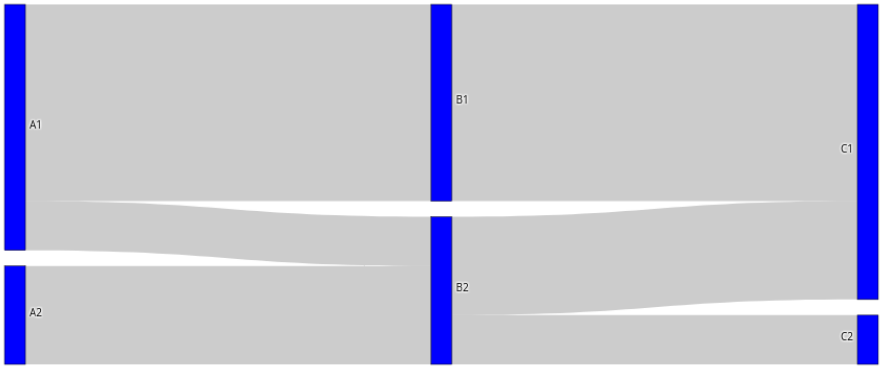

Schéma 4.6 – Exemple diagramme de Sankey

Voilà permet aussi de changer l'interface graphique de la page d'accueil grâce à des templates, on a donc créer notre propre template pour avoir une page d'accueil personnalisée à notre goût, qui présente nos differentes expérimentations.

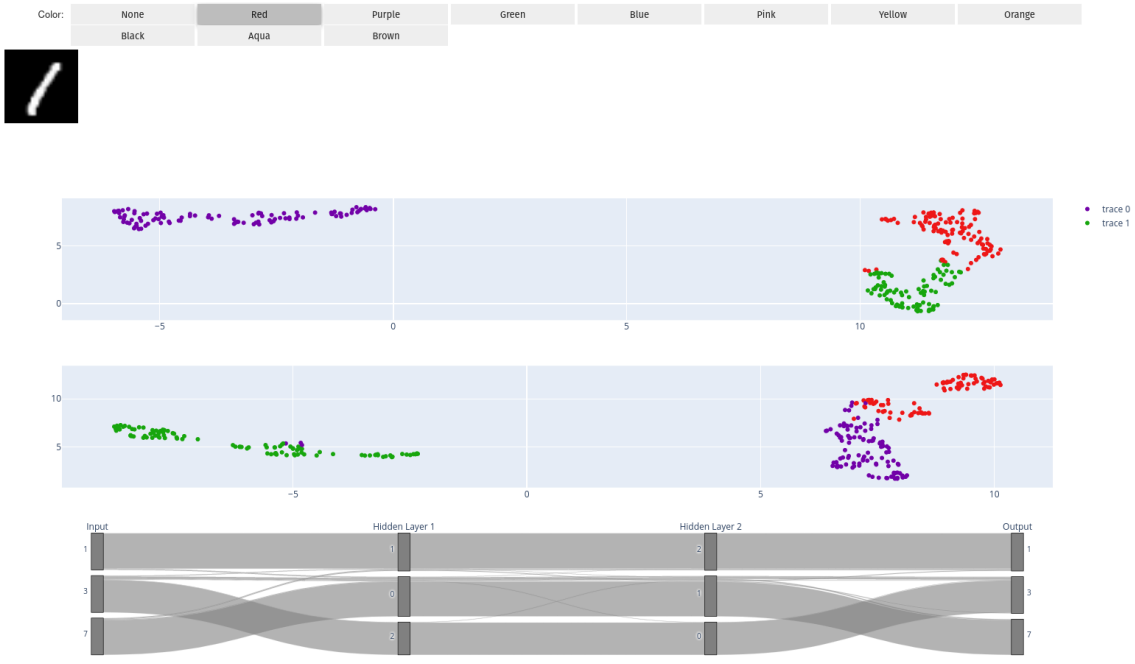

Schéma 4.7 – Page web

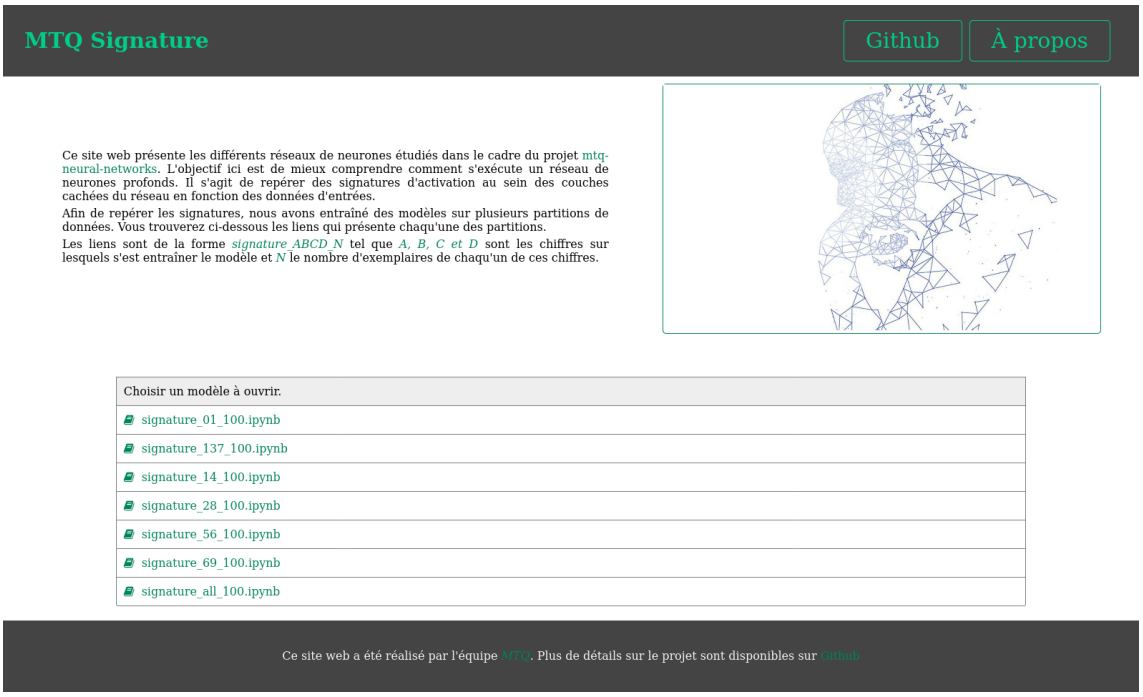

Schéma 4.8 – Page d'accueil

Chapitre 5

Analyse des résultats

Cette partie est très importante vu qu'elle va nous permettre de répondre aux questions posées au début de ce rapport 1.3. Mais aussi, nous essaierons d'avancer des hypothèses sur le fonctionnement interne d'un réseau de neurones.

Pour répondre à ces questions, nous avons entraîné notre modèle pour reconnaître des images de 1 et de 7. Nous avons obtenu la visualisation de la figure 5.1.

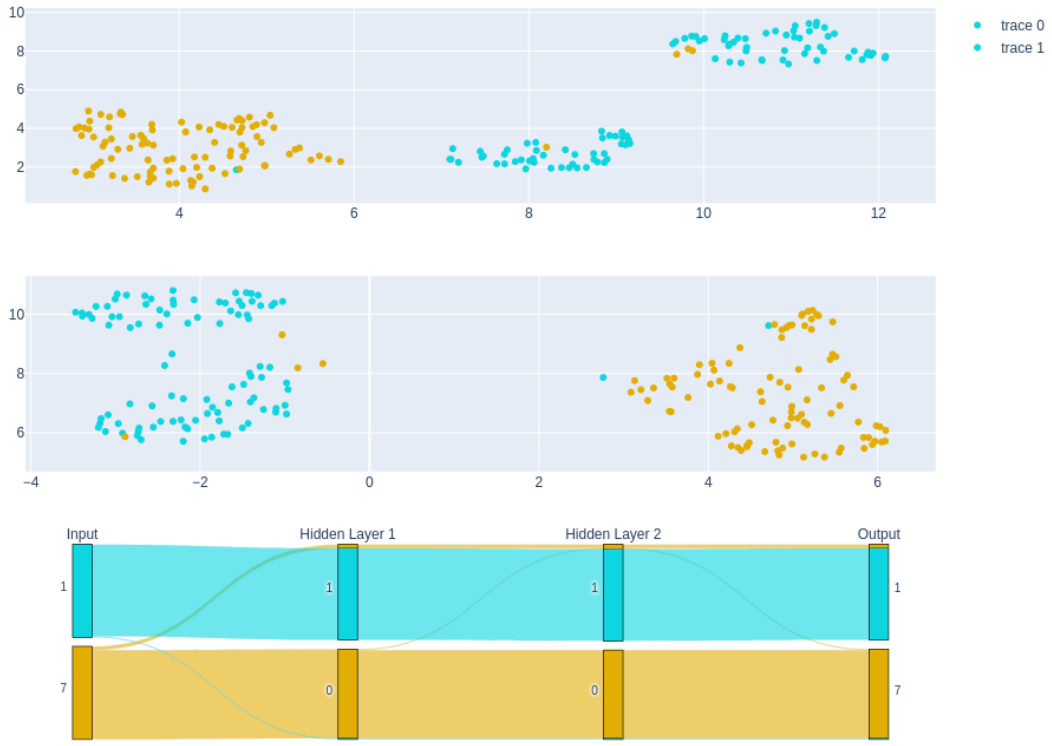

Schéma 5.1 – Visualisation des signatures

Dans cette visualisation, les deux premiers graphes représentent la projection faite par *UMAP* des deux vecteurs (H_1, H_2) définis dans la partie 4.3. Chaque point est donc une projection en deux dimensions de son vecteur initial. La couleur des points est associée à leurs labels.

Le troisième graphe correspond au diagramme de *Sankey*. On voit clairement dans ce dernier, le parcours de chaque image.

Dans *UMAP*, deux points qui sont visuellement proches (resp. loins), ne le sont pas réellement, en distance euclidienne. Les clusters qui apparaissent dans la projection de *UMAP* ne correspondant pas forcément au clustering de *K-means*. On peut remarquer ce fait dans le premier graphe. On observer visuellement trois clusters, alors que le clustering *K-means* en donne deux qu'on voit dans le diagramme de *Sankey*.

Changement de comportement En regardant le diagramme de *Sankey*, on remarque que la majorité des images de 1 (Resp. de 7) sont regroupées dans le même cluster (97% en moyenne). Ce qui nous indique que notre modèle arrive, dès la première couche cachée, à reconnaître une image.

Différence de signatures Toujours grâce au diagramme de *Sankey*, on observe (figure 5.1) que les signatures des 1 sont majoritairement différentes de celles des 7. Sauf pour quelques rares exceptions. Par exemple, quatre images de 7 ont été regroupées dans le cluster des 1, dès la première couche cachée. En effet, ces quatre images de 7 ressemblent vraiment à des 1, comme le montre la figure 5.2.

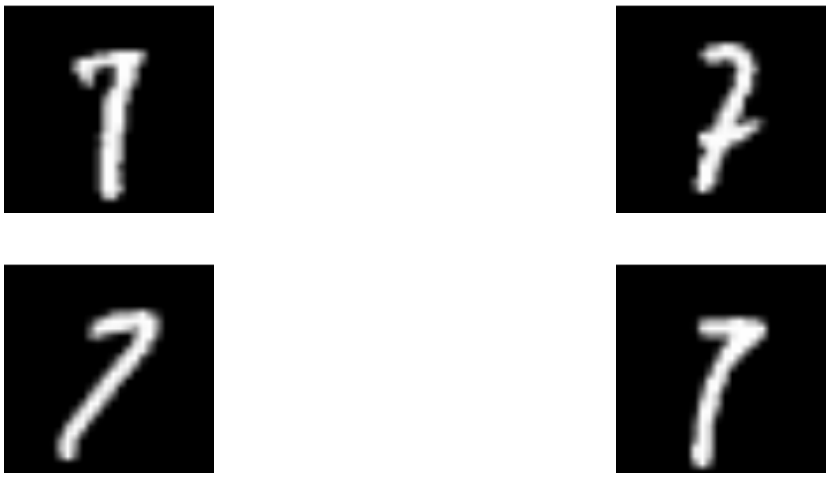

Schéma 5.2 – Les images de 7 ressemblant à des 1

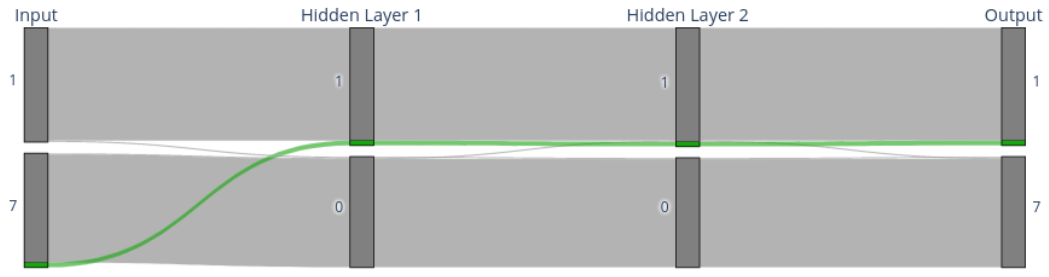

Schéma 5.3 – Signature des 7 ressemblant à des 1

Insertion d'anomalies Afin de répondre à la dernière question, nous avons réalisé deux expérimentations dans lesquelles nous avons inséré des anomalies. C'est-à-dire, pour un modèle entraîner à reconnaître des images de 1 et de 7, on lui passe :

Des images de 4 On observe dans ce premier cas (voir la figure 5.4), que la majorité des images de 4 sont regroupées avec les images de 7 et ce depuis la première couche cachée. Cela revient probablement au fait que les images de 4 ressemblent plus aux images de 7 qu'aux images de 1.

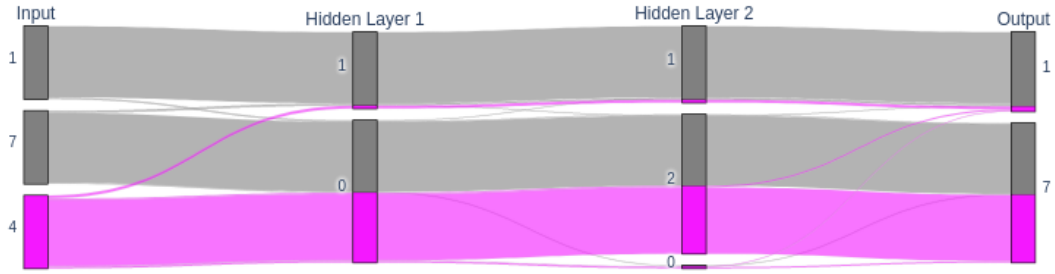

Schéma 5.4 – Insertion d'images de 4

Des images de 3 Par contre, comme les images de 3 ne ressemblent ni aux images de 1, ni à ceux de 7, elles ont été regroupées de manière hasardeuse (voir la figure 5.5).

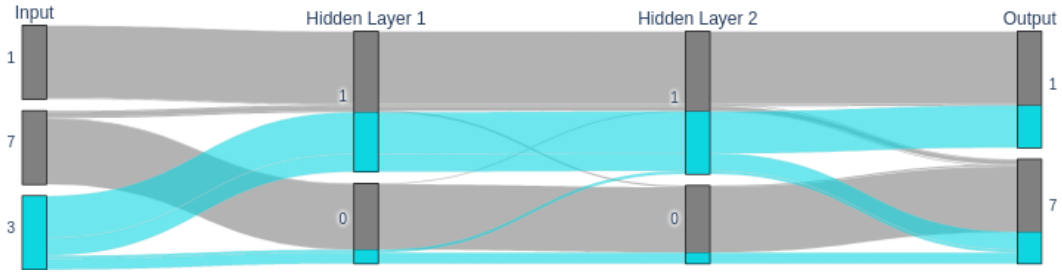

Schéma 5.5 – Insertion d'images de 3

D'après ces deux expérimentations, on conclu que le modèle essai tant bien que mal de regrouper les anomalies avec les objets qu'il juge les plus ressemblants à ces dernières.

5.1 Conclusion

Les observations précédentes, nous poussent à croire que les ANN [1] se basent sur une sorte de détection de patterns afin de généraliser un certain concept. Prenons exemple sur l'insertion des images de 4, on observe que le modèle a détecté une ressemblance entre le chiffre 4 et le chiffre 7 qui l'a induit à classer la plupart des images de 4 en 7.

Si on prend un cas générale de classification de données ayant n attributs de A_1 à A_n. On cherche à entraîner un modèle à classifier ces données en deux classes C_1 et C_2. Il est possible que durant la phase d'entraînement, le modèle détecte des similarités entre les objets sur un attribut A_i qui indique qu'il existe peut-être *une corrélation*, mais pas forcément *une causalité*, avec les deux classes C_1 et C_2. Ce qui mènera notre modèle à de possibles mauvaises classifications sur des données qu'il n'a jamais vues.

1. Artificial Neural Network

On ne pourra donc jamais affirmer avec certitude qu'un modèle a réussi à généraliser un certain concept, et ce, malgré de bons résultats durant les deux phases d'entraînement et de validation. Ce qui accentue encore plus l'importance des prétraitements des données et de l'explication des résultats. Chose qu'un réseau de neurones ne peut faire.

Bibliographie

[1] Leland McInnes. *Plotting UMAP*. URL : https://umap-learn.readthedocs.io/en/latest/plotting.html. (accessed : 05.05.2021).

[2] Leland McInnes. *UMAP Clustering*. URL : https://umap-learn.readthedocs.io/en/latest/clustering.html. (accessed : 05.05.2021).

[3] Dedocoton Olivier. *Faire du Clustering avec l'algorithme K-means*. URL : https://ledatascientist.com/faire-du-clustering-avec-lalgorithme-k-means/. (accessed : 05.05.2021).

[4] Parul Pandey. *Visualizing Kannada MNIST with UMAP Technique*. URL : https://www.kaggle.com/parulpandey/part3-visualising-kannada-mnist-with-umap?scriptVersionId=0. (accessed : 05.05.2021).

[5] Pascal Poncelet. *Pattern Signatures*. URL : https://www.lirmm.fr/~poncelet/RN/. (accessed : 05.05.2021).

Annexe A

Code source

A.1 Code source de la méthode *cut_ data*

```
def cut_data(x, y, keep=[0,1], n=None):
  new_x = x[np.where(y == keep[0])][:n]
  new_y = y[np.where(y == keep[0])][:n]
  for i in keep[1:]:
      new_x = np.concatenate((new_x , x[np.where(y == i)][:n]))
      new_y = np.concatenate((new_y , y[np.where(y == i)][:n]))

  return new_x, new_y
```

A.2 Code source de la méthode *normalize_ dataset*

```
def normalize_dataset(x_train, y_train):
  # Scale images to the [0, 1] range
  x_train = x_train.astype("float32") / np.amax(x_train)
  # Flatten the images.
  x_train = x_train.reshape((-1, len(x_train[0]) * len(x_train[0][0])))
  # convert class vectors to binary class matrices
  # Ex: 1 will become [0, 1] if len(np.unique(y_train))==2
  y_train = np.array(list(map(lambda x:  [1 if x == k else 0 for k in np.sort(np.
  unique(y_train))], y_train)))

  return x_train , y_train
```

A.3 Code source de la méthode *get_ best_ k*

```
def get_best_k(hidden_layers, kmax=7, verbose=False):
  best_k = []

  fig, axes = plt.subplots(nrows=len(hidden_layers), ncols=2, sharex=True)
  fig.set_size_inches(15, 9)

  for x, ax in zip(hidden_layers,axes):
      sil = []
      sse = []

      for k in range(1, kmax+1):
          kmeans = KMeans(n_clusters = k, random_state=24).fit(x)
          labels = kmeans.labels_
          if k == 1:
              sil.append(0)
```

```
            else:
                sil.append(silhouette_score(x, labels, metric = 'euclidean'))

            centroids = kmeans.cluster_centers_
            pred_clusters = kmeans.predict(x)
            curr_sse = 0
            # calculate square of Euclidean distance of each point from its cluster
     center and add to current WSS
            for i in range(len(x)):
                curr_center = centroids[pred_clusters[i]]
                curr_sse += np.sqrt(np.sum((x[i] - curr_center) ** 2, axis=0))
            sse.append(curr_sse)

        best_k.append(np.array(sil).argmax()+1)

        if verbose:
            # ploting
            ax[0].plot(range(1,kmax+1),sil,color='r')
            ax[0].set_xlabel('K values')
            ax[0].set_ylabel('Silhouette score')
            ax[1].plot(range(1,kmax+1),sse, color='g')
            ax[1].set_xlabel('K values')
            ax[1].set_ylabel('Elbow')

            ax[0].grid(True)
            ax[1].grid(True)

    if verbose: #Verbose
        print("Best K values: ", best_k)
        plt.show()
    else:
        plt.close()

    return best_k
```