Université de Montpellier - Licence 3
Informatique

TER

Signatures des Réseaux de Neurones

Auteurs :
Hamza JEBBAR
Younes JEBBAR
Doha CHEMAOU

Encadrant :
Pascal PONCELET

Année 2019-2020

Remerciements

Avant de commencer la présentation de ce rapport nous profitons de l'occasion pour remercier les personnes qui ont contribué à la réalisation de ce projet. Nous tenons à remercier notre encadrant de TER **M. Pascal PONCELET** pour ses conseils et son encadrement tout au long du projet, ainsi que toute personne qui a proposé des solutions aux erreurs sur internet, ce qui nous a beaucoup aidé lorsqu'on était face à des difficultés.

Table des matières

Table des figures

1 Introduction

Les domaines de l'analyse du signal sonore ou visuel et notamment de la reconnaissance faciale, de la reconnaissance vocale, de la vision par ordinateur, du traitement automatisé du langage ont connu des progrès importants et rapides grâce à l'apprentissage profond (*le deep learning*). Cette technique s'appuie sur un réseau de neurones artificiels, s'inspirant du cerveau humain, dans un processus d'apprentissage.

Un réseau de neurones apprend à partir des données qui lui sont fournies, et qui sont déjà classifiées. Grâce à cet apprentissage il arrive à prédire les classes des données qui lui sont passées en entrée. Ce type d'apprentissage s'appelle l'apprentissage supérvisé [1] (*Supervised Learning*).

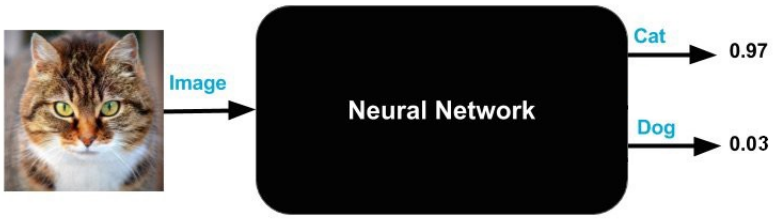

Figure 1 – La boîte noire

La figure 1 montre une boîte noire qui représente le réseau de neurones. Après le passage d'une image d'un chat par ce réseau, nous recevons comme prédiction 0.97 désignant que l'image appartient à un chat, tandis que la proportion que l'image soit appartenant à un chien est 0.03.

Si une donnée de catégorie totalement différente de ce que le modèle a connu pendant l'apprentissage lui est passée, le résultat du classement de celle-ci en sortie restera imprévisible. Par exemple en reprenant le réseau de neurones de la figure 1, qui ne connaît que les chats et les chiens, mais cette fois nous allons remplacer la donnée de l'entrée par image d'une voiture. La figure 2 illustre ce cas.

1. https ://www.24pm.com/117-definitions/512-apprentissage-supervise

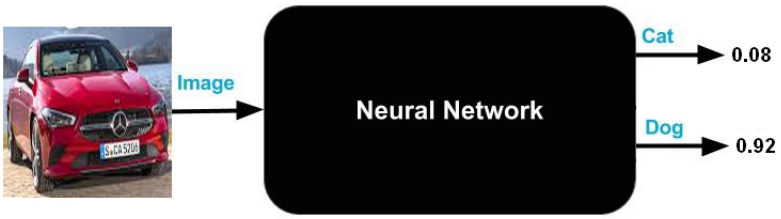

Figure 2 – La boîte noire avec image d'une voiture comme entrée

Nous constatons dans le résultat que les valeurs obtenus sont chat (0.08) et chien (0.92) et donc que notre voiture sera classée comme un chien.

Nous pouvons considérer le réseau de neurones comme une boîte noire que nous chercherons à ouvrir durant ce projet afin de découvrir les opérations qui se passent à l'intérieur et qui décident le résultat de la sortie.

Le reste du rapport est organisé de la manière suivante. Dans la section 2, nous proposons un rappel sur les réseaux de neurones. La section 3 porte sur les outils utilisés. En ce qui concerne la section 4, celle-ci comportera le travail réalisé. La section 5, quant à elle, présentera l'organisation du travail . Une analyse des résultats sera donnée dans la sections 6. Et pour finir, une conclusion dans la section 7.

2 Rappels sur les réseaux de neurones

Dans cette section nous proposons un rappel des principaux concepts des réseaux de neurones pour faciliter la lecture de la suite du document. Le lecteur souhaitant approfondir ce domaine peut se rapporter à la référence [13].

2.1 Définition

Un réseau de neurones artificiels (*Neural Network*) est un système informatique s'inspirant du fonctionnement du cerveau humain pour apprendre. Il consiste en des milliers voire des millions de neurones organisés en couches interconnectées et organisées en tiers. La figure 3 montre un exemple d'un réseau de neurones artificiels.

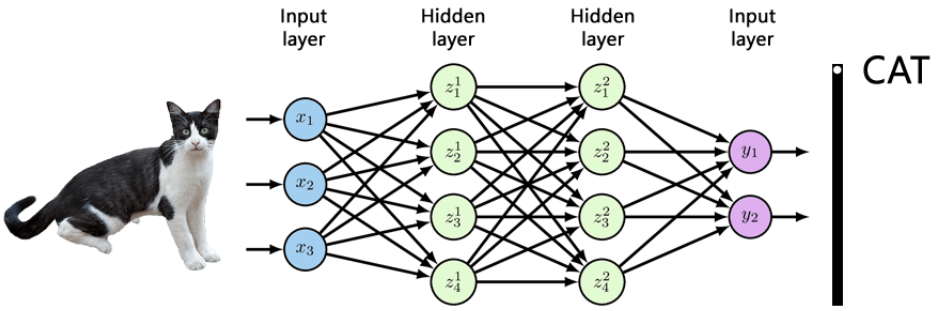

Figure 3 – Exemple d'un réseau de neurones artificiels

A gauche, nous avons la couche d'entrée (*input layer*) (en bleu sur le schéma). Cette dernière va recevoir les variables prédictives utilisées pour la classification. A droite (en violet), la couche de sortie (*output layer*) donne le résultat de la classification. Au milieu les différentes couches (en vert) correspondent aux couches cachées (*hidden layers*). Le fonctionnement général est le suivant. Tout d'abord les données d'apprentissage (variables prédictives) sont transmises à la couche d'entrée. Par la suite ces données sont multipliées par une matrice de poids W à laquelle une fonction d'activation est appliquée et ce principe est propagé sur tout le réseau (Cf. section 2.2). Le résultat est évalué à la fin et, étant donné que nous sommes dans un contexte d'apprentissage supervisée, l'erreur de prédiction est calculée et reportée dans tout le réseau pour mettre à jour les poids (Cf. section 2.3).

2.2 Propagation vers l'avant

Le processus de la propagation vers l'avant (*Forward propagation*) décrit le chemin que traverse la donnée dès son entrée au réseau, jusqu'à son arrivé à la dernière couche qui décide la classe où elle sera classifiée.

Un neurone est défini par un vecteur contenant des poids, il calcule sa multiplication vectoriel avec les données qui lui ont été fourni, ajoute un terme de biais et applique la fonction d'activation adéquate. Ce résultat sera passé à chaque neurone de la couche suivante, jusqu'à l'aboutissement à la dernière couche qui calcule la probabilité d'appartenance relative à chaque classe. La figure 4 illustre ce calcule où W représente le vecteur de poids, b le biais, l et L les numéros des couches, g et sigma les fonctions d'activations.

$$
\begin{aligned}
z^{[l]} &= W^{[l]} a^{[l-1]} + b^{[1]} \\
a^{[l]} &= g(z^{[l]}) \\
&\dots \\
z^{[L]} &= W^{[L]} a^{[L-1]} + b^{[L]} \\
a^{[L]} &= \sigma(z^{[L]})
\end{aligned}
$$

Figure 4 – Forward propagation

Ce processus est indispensable pour le calcul de la fonction de coût, que les algorithmes d'optimisation cherchent à minimiser.

2.3 Algorithmes d'optimisation

Les algorithmes d'optimisation (descente de gradient [2], RMSProp [3], Adam [4] etc...) cherchent à minimiser une fonction de coût donnée. La première initialisation des poids (souvent aléatoirement ou à travers un algorithme d'initialisation) aboutit à un grand nombre de données mal classées.

2. https ://machinelearnia.com/descente-de-gradient/
3. https ://datafranca.org/wiki/RMSProp
4. https ://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

La fonction de coût, lors de l'apprentissage, représente la somme des erreurs produites lors de la classification en suivant la méthode de propagation vers l'avant. Les algorithmes d'optimisation cherchent à minimiser cette somme et par conséquent minimiser le nombre de données mal classées. La figure 5 montre la minimisation de la fonction de coût lors de chaque itération (epoche) jusqu'à l'arrivée à un minimum globale.

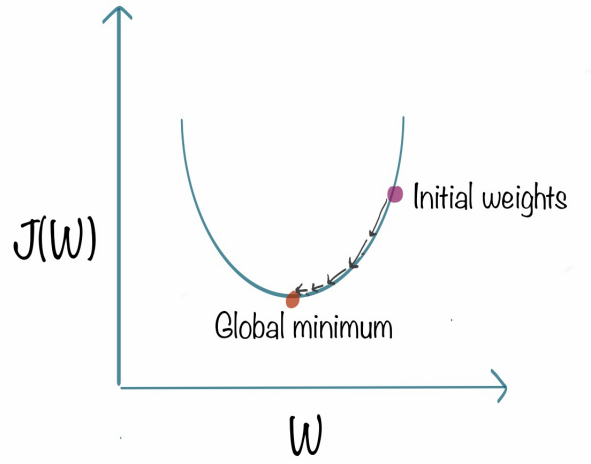

Figure 5 – Processus de minimisation de la fonction de coût

Ceci n'est possible qu'à travers l'algorithme du backward propagation qui mettra les poids à jour lors de chaque itération en fonction de la valeur de la fonction du coût.

2.4 Rétro-propagation

Contrairement à l'algorithme de propagation vers l'avant, qui va de la donnée d'entrée(*Input*) vers la classe prédite(*Output*), la rétro-propagation (*Backward propagation*) va de la classe prédite en revenant vers la donnée d'entrée. Tout au long de ce chemin, l'algorithme calcule la dérivée de la fonction de coût par rapport à chaque vecteur de poids, multiplie le résultat par un paramètre nommé "taux d'apprentissage" et retranche cette somme de la valeur initiale du vecteur. Le concept de cet algorithme est de punir chaque vecteur de poids pour sa contribution dans les erreurs. A chaque epoche, la valeur de la fonction de coût diminue, ce qui indique une augmentation de la précision du modèle et par conséquent la réduction de nombre des données mal classées.

2.5 Les fonctions d'activation

Les fonctions d'activation sont des fonctions dont l'objectif est d'aider le modèle à mieux apprendre et à converger plus rapidement, et plus important encore, elles empêchent le modèle de se transformer en fonction linéaire ce qui influencera sa performance négativement.
Le choix de la fonction d'activation dépend de l'architecture et de la couche, certaines d'entre eux souffre des problèmes relatifs au gradient (vanishing gradient [5], exploding gradient [6]), notamment la sigmoid et la reLU, pour ces raisons, le meilleur choix pour les couches de l'intérieur est la fonction Leaky-reLU. Quant à la couche dernière, ou bien la sigmoid si la classification est binaire ou bien la softmax, qui n'est qu'une généralisation de la sigmoid sur plusieurs classes de sortie, si le nombre de classes est plus grand que deux.

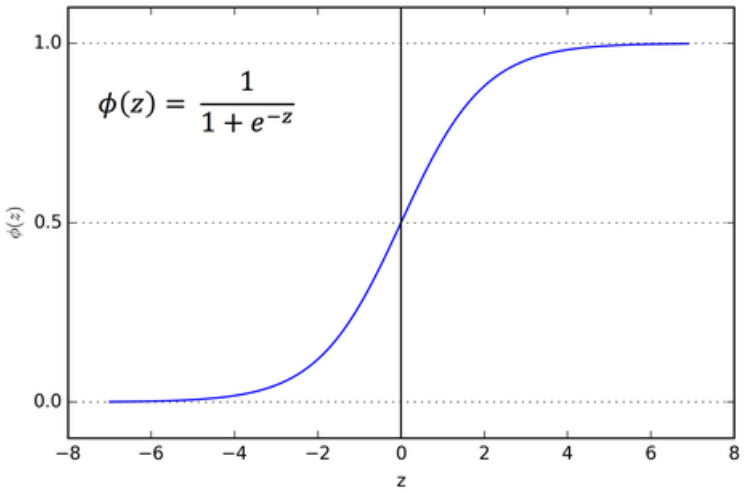

Figure 6 – Fonction d'activation sigmoid

5. https ://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484
6. https ://machinelearningmastery.com/exploding-gradients-in-neural-networks/

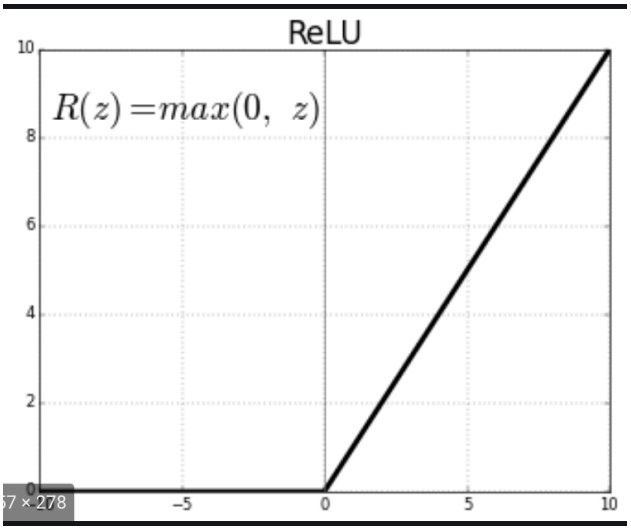

Figure 7 – Fonction d'activation reLU

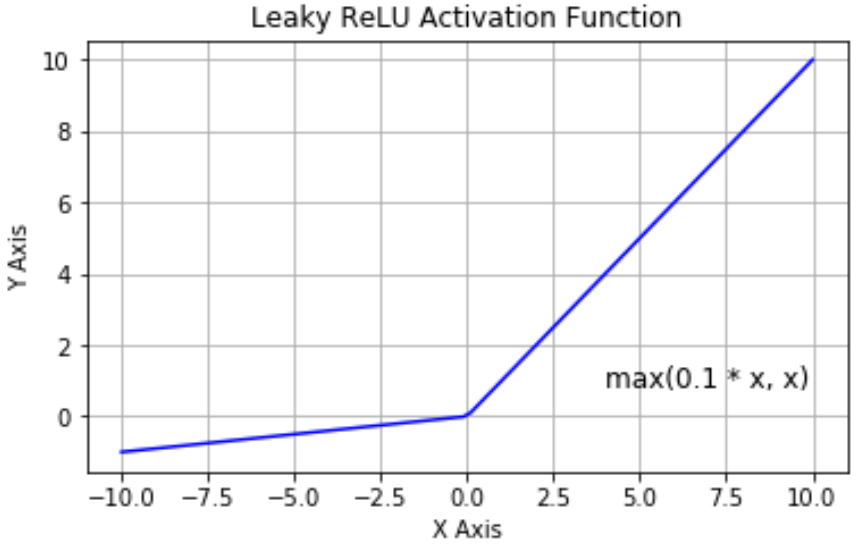

Figure 8 – Fonction d'activation leaky-reLU

3 Outils utilisés

Les outils techniques utilisés comptent les langages de programmation, les bibliothèques et les framework.

— **Python**[1] : est un langage de programmation interprété. C'est le langage open source le plus employé par les informaticiens. Ce langage s'est propulsé en tête de la gestion d'infrastructure, d'analyse de données ou dans le domaine du développement de logiciels. Python permet de se concentrer sur ce qu'il faut faire plutôt que sur la manière dont il faut le faire . Ainsi, développer du code avec Python est plus rapide qu'avec d'autres langages.

— **Javascript**[2] : est un langage de programmation qui permet d'implémenter des mécanismes complexes sur une page web, de contrôler le contenu multimédia, d'animer des images, ...

— **os**[4] : est une bibliothèque qui fournit une manière portable d'utiliser les fonctionnalités dépendantes du système d'exploitation.

— **re**[3] : est une bibliothèque qui fournit des opérations sur les expressions rationnelles similaires à celles que l'on trouve dans Perl.

— **random**[5] : est une bibliothèque qui permet la génération de nombres aléatoires.

— **numpy**[6] : est une bibliothèque numérique apportant le support efficace de larges tableaux multidimensionnels, et de routines mathématiques de haut niveau (fonctions spéciales, algèbre linéaire, statistiques, etc.).

— **pandas**[7] : est une bibliothèque qui permet la manipulation et l'analyse des données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles.

— **matplotlib**[8] : est une bibliothèque qui permet de produire des graphes de qualité.

— **scikit-learn**[9] : est une bibliothèque libre destinée à l'apprentissage automatique. Elle comprend notamment des fonctions pour estimer des forêts aléatoires, des régressions logistiques, des algorithmes de classification, et les machines à vecteurs de support.

— **keras**[10] : est un framework qui permet d'interagir avec les algorithmes de réseaux de neurones profonds et de machine learning comme Tensorflow.

— **D3**[11] : est une bibliothèque graphique JavaScript qui permet l'affichage de données numériques sous une forme graphique et dynamique.

— **FLASK**[12] : est un framework open-source de développement web en Python. Son but principal est d'être léger, afin de garder la souplesse de la programmation Python, associé à un système de templates.

4 Travail réalisé

La figure 9 illustre le fonctionnement général de notre application. Tout d'abord, nous lisons les données et appliquons un modèle afin de récupérer les fonctions d'activation. Par la suite, nous utilisons un algorithme de clustering pour regrouper ces dernières en fonction de leur classe d'appartenance. Enfin, nous générons des fichiers Json pour avoir une visualisation sur une interface web. Les sous sections suivantes décrivent ces différents composants.

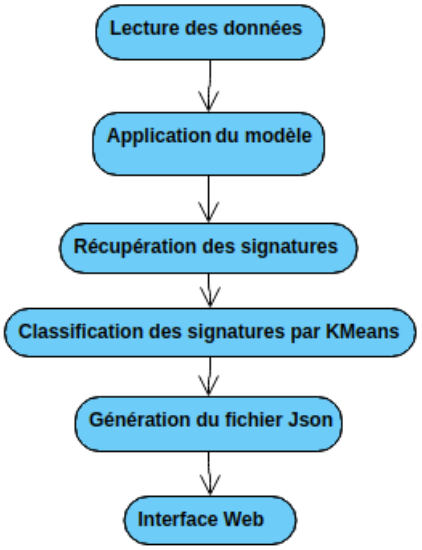

Figure 9 – Schéma du fonctionnement général de l'application

4.1 Application du modèle

Pour que la reconnaissance d'un certain type d'image soit possible, la création et l'entraînement d'un réseau de neurones reste indispensable. Une classe parmi celles apprises doit être assignée à l'image passée en entrée du réseau.

4.1.1 Création et apprentissage

Le modèle est créé puis entraîné à l'aide de Keras. La création du modèle est faite en choisissant le nombre de couches ainsi que le nombre des neurones correspondants à chacune d'elles. Ensuite, celui-ci doit être entraîné à l'aide d'un jeu de données. Le code suivant illustre la construction de l'architecture du modèle en ajoutant les couches et le nombre de neurones.

```
model = Sequential()
model.add(Dense(64, input_dim = input_dim , activation = 'relu'))
model.add(Dense(32, activation = 'relu'))
model.add(Dense(16, activation = 'relu'))
model.add(Dense(len(list(set(train_y))), activation = 'softmax'))
```

Listing 1 – Création du modèle

Dans cet extrait de code, 'relu', une fonction d'activation, est utilisée pour toutes les couches internes, sauf la dernière qui utilise la fonction d'activation 'softmax' au lieu d'utiliser la fonction 'sigmoid', car il s'agit du cas de classification en classes multiples.

4.1.2 La vérité terrain

Le jeu de données MNIST, que nous avions dû récupérer, est celui que nous utilisons. Il contient des images de numéros, allant de zéro à neuf, écrits à la main. La figure 10 montre quelques exemples du jeu de données.

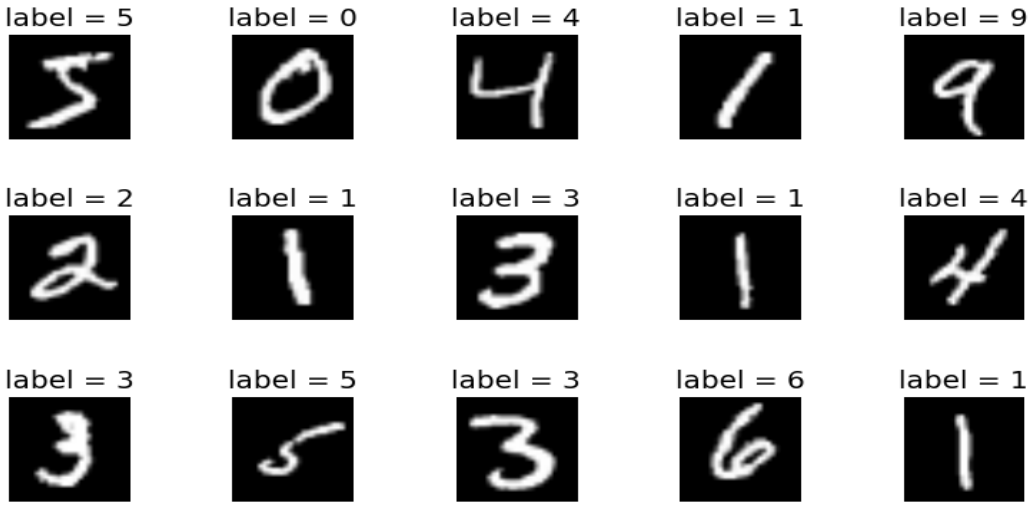

Figure 10 – Les données Mnist

Nous effectuons l'apprentissage du modèle que sur un certain nombre de classes appartenant à ce jeu de données (par exemple 2,5,6,8). Par conséquent, si le modèle reçoit une données appartenant à une autre classe (par exemple la classe 11), sa prédiction sera obligatoirement une classe parmi celles apprises par le modèle (2,5,6,8). La classe 11 joue le rôle de la vérité terrain.

Le modèle reçoit des données en entrée qui doivent être classifiées en sortie. Lors de la prédiction, les valeurs récupérées des fonctions d'activation de chaque neurone sont enregistrées dans un fichier csv sous le nom 'mnist_l1_64_l2_32_l3_16_.csv' par exemple. Chaque ligne de ce fichier commence par la classe prédite, suivi des valeurs des fonctions d'activation, séparés par des virgules. L'étape qui suit consiste à récupérer les signatures pour pouvoir les exploiter.

4.2 Récupération des signatures

Nous avons choisi une approche permettant de traiter indépendemment le résultat des fonctions d'activation de chaque couche. Cette approche nous a aidé à comprendre le comportement de chacune d'entre elles en détails ce qui nous a permis de découvrir ce qui se passe à l'intérieur de la boîte noire.

Le nom du fichier comporte toutes les informations sur le nombre des couches et le nombre des neurones. L'exploitation de celui-ci a permis l'amélioration de l'approche mentionnée précédemment et cela en la généralisant. Ainsi notre programme arrive à traiter différentes architectures avec différents nombres de couches et de neurones.

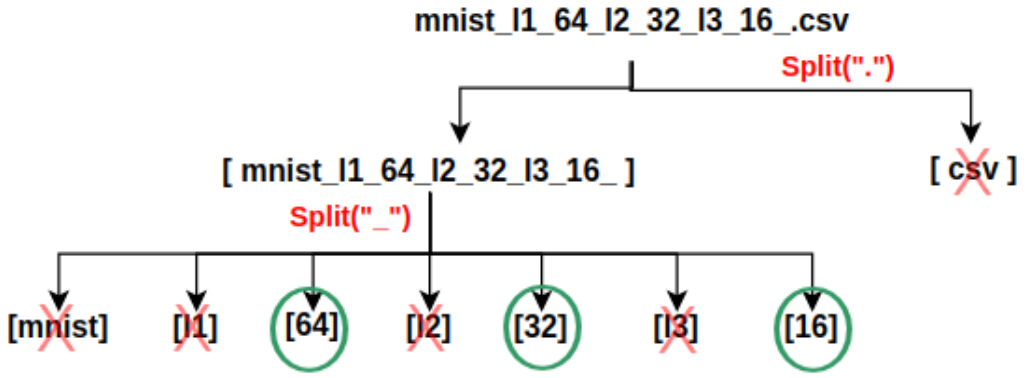

Figure 11 – Processus de récupération des informations de l'architecture du modèle

La figure 11 représente le processus de récupération de nombre de couches ainsi que les nombres de neurones par chaque couche. D'abord, nous procédons à diviser le nom du fichier par le caractère (point ".") pour avoir deux chaînes de caractères dont une va être ignorée ("csv" car elle ne comporte aucune information utile) et l'autre sera à son tour divisée par le caractère (*underscore* "_"), la chose qui va nous permettre de retrouver les nombres de neurones par chaque couches séparément de la chaîne représentant le nom du fichier.

Après avoir récupérer le nombre de couches ainsi que les nombres de neurones, nous procédons à récupérer les signatures à partir du fichier csv pour faire la classification. Le code utilisé pour le chargement des données et leur répartition est donné en annexe.

4.3 Classification des signatures par KMeans

Cette phase consiste à regrouper les signatures du modèle à l'aide de KMeans. Ce dernier est un algorithme de clustering permettant de regrouper les données en un ensemble de clusters selon un entier passé en paramètres qui représente le nombre de clusters souhaité. Ci-dessous les instructions permettant de réaliser cette tâche.

```
1     kmeans = KMeans(n_clusters=nb_clusters,random_state=0).fit(
      layers[i])
2     cluster = kmeans.predict(layers[i])
```

Listing 2 – le clustering par KMeans

Tout d'abord, nous avons commencé par regrouper les signatures des données appartenant aux classes apprises (par exempe 2,5,6,8 pour MNIST) par le modèle, ensuite nous avons fait la même chose sur la nouvelle classe (par exemple 11 pour MNIST). A l'aide de l'outil PCA [7], qui est un algorithme souvent utilisé pour but de réduire les dimensions des données, nous avons pu faire un plot de nuage de points illustrant le regroupement des données en clusters (figure 12).

7. https ://docs.aws.amazon.com/fr_fr/sagemaker/latest/dg/how-pca-works.html

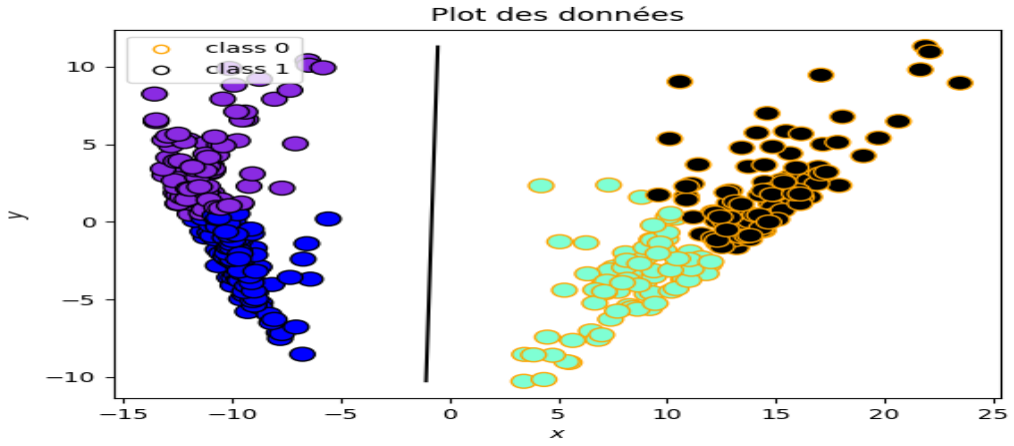

Figure 12 – Plot 2D des clusters (avec 4 clusters comme paramètre de KMEANS)

Après le clustering, nous appliquons une fonction qui calcule les pourcentages des clusters d'une couche dans chaque classe afin d'éliminer ceux qui ont un faible pourcentage dans une classe donnée. Ceci a pour but la réduction du bruit, ce qui nous sera utile lors de la réalisation de l'interface web.

4.4 Interface web

Après avoir éliminé les clusters de faible pourcentage, nous avons procédé à établir une communication entre le serveur et l'interface pour pouvoir transmettre les informations à afficher. Une solution simple et rapide était d'utiliser le format Json [8], qui est un type de données intermédiaire, pour sauvegarder ces données dans un fichier lors du traitement, et puis le charger lors de l'affichage.

Ce fichier représente un dictionnaire à deux clefs :

— **Les nœuds :** cette partie représente les clusters et les classes à travers des rectangles. Chaque classe a une couleur prédéfinie, et chaque cluster prend la couleur de la classe à laquelle il appartient. Les clusters appartenant à plusieurs classe auront une couleur spéciale (gris).

8. https ://www.json.org/json-fr.html

— **Les liens :** ce tableau définit les liens entre les noeuds, ils sont paramétrés par un attribut source et un attribut destination pour pouvoir tracer le chemin, un attribut valeur qui sera utile pour définir l'épaisseur ainsi qu'un attribut indiquant la classe du lien qui définira la couleur.

Un exemplaire de ce fichier est proposé en annexe.

Finalement, nous chargeons ce fichier dans le côté client et, en utilisant la bibliothèque D3js [9], nous exploitons ces informations afin d'avoir un dessin clair du chemin des données de chaque classe, comme l'illustre la figure 13.

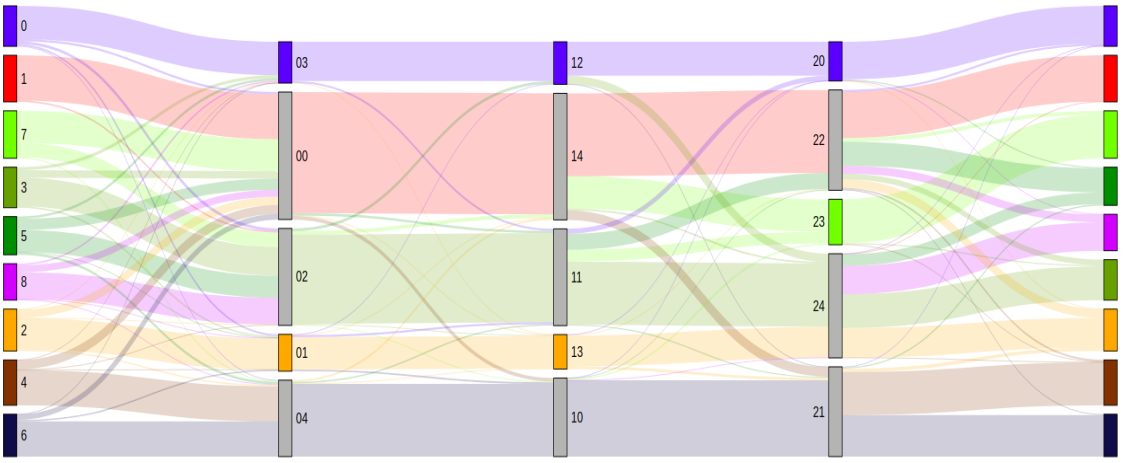

Figure 13 – Exemple de signatures obtenues pour le jeu de données MNIST

La figure 13 montre un exemple de ce que nous obtenons comme résultat d'affichage. Les rectangles à gauche représentent les classes des données d'entrée avec chaque classe est désignée par sa propre couleur. Les rectangles du milieu représentent les clusters où chaque cluster prend la couleur de la classe à laquelle il appartient. Les rectangles en gris correspondent aux clusters en communs c'est à dire qui appartiennent à plusieurs classes. Les rectangles à droite désignent les classes prédites par le modèle pour ce jeu de données avec les mêmes couleurs des classes d'entrée. Les liens entre ces rectangles montrent les chemins des données à l'intérieur du modèle avec les couleurs des classes auxquelles appartiennent les données après le passage par chaque couche.

9. https ://d3js.org/

5 Organisation et répartition des tâches

Nous avons pu, par le biais de ce projet, apprendre à nous organiser en faisant participer chaque membre du groupe à la réalisation du travail, en assignant à chacun la tâche dans laquelle il excelle le mieux et en discutant les problèmes et les solutions prenant en compte l'avis de chacun.

Dans cette section nous présentons la manière que nous avons suivi tout au long de la période du projet afin d'avoir une bonne organisation. Dans un premier temps nous introduisons un diagramme de Gantt désignant toutes les tâches ainsi que leurs période de réalisation. Ensuite, dans la deuxième sous section, nous décrivons la méthode que nous avons utilisé pour répartir les tâches entre les membres du groupe.

5.1 Organisation

Cette sous section présente l'organisation du projet sous forme d'un diagramme de Gantt [10]. Ce dernier est un outil permettant la visualisation dans le temps des différentes tâches composant le projet. La figure 14 introduit un diagramme de Gantt de note projet.

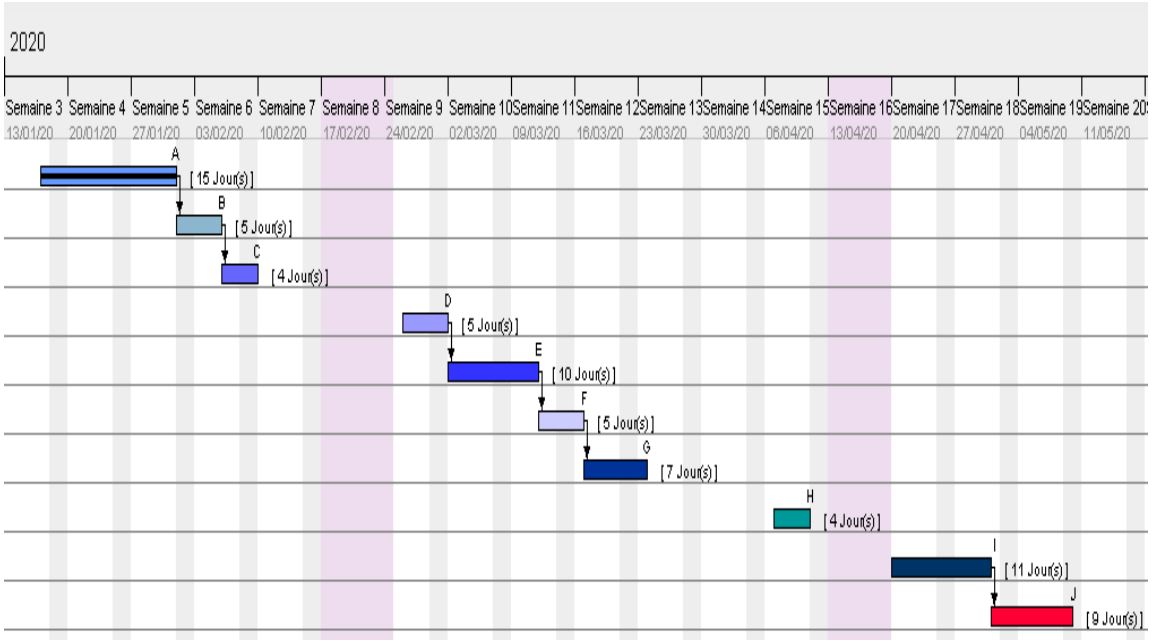

Figure 14 – Diagramme de Gantt

10. https ://www.gantt.com/

Le début du projet était en mi-janvier 2020 par une réunion dans laquelle nous avons retenu une idée générale sur sujet. Ensuite, nous avons commencé à faire des recherches afin d'acquérir les connaissances nécessaires pour la réalisation de ce projet (A). L'étape suivante était la récupération des signatures du modèle dans un fichier csv pour faire des histogrammes visualisant ces signatures par layer (B, C). Dans la période marquée par (D), nous avons exploité le classifieur DBSCAN après avoir calculer la matrice des distances de Levenshtein des signatures. Ensuite, nous avons généré un fichier csv contenant les signatures des clusters, classifiées par Kmeans, avec élimination des clusters qui ont un faible pourcentage (E, F, G). La période suivante (H), nous l'avons servi à l'implémentation de la fonction "plot2D" qui a pour but la visualisation des clusters en 2 dimensions. la phase finale du projet était la réalisation d'une interface web montrant les chemins traversés par les données à l'intérieur de notre réseau de neurones (I) avant la rédaction du rapport (J).

5.2 Répartition des tâches

Nous avons utilisé différents outils afin de mieux répartir les tâches. Tout d'abord, nous avons créé un projet sur Trello [11]. Ce dernier est un outil de gestion de travail en ligne, il repose sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leurs avancement.Dans notre projet sur Trello nous avons élaboré quatre sections comme le montre la figure 15.

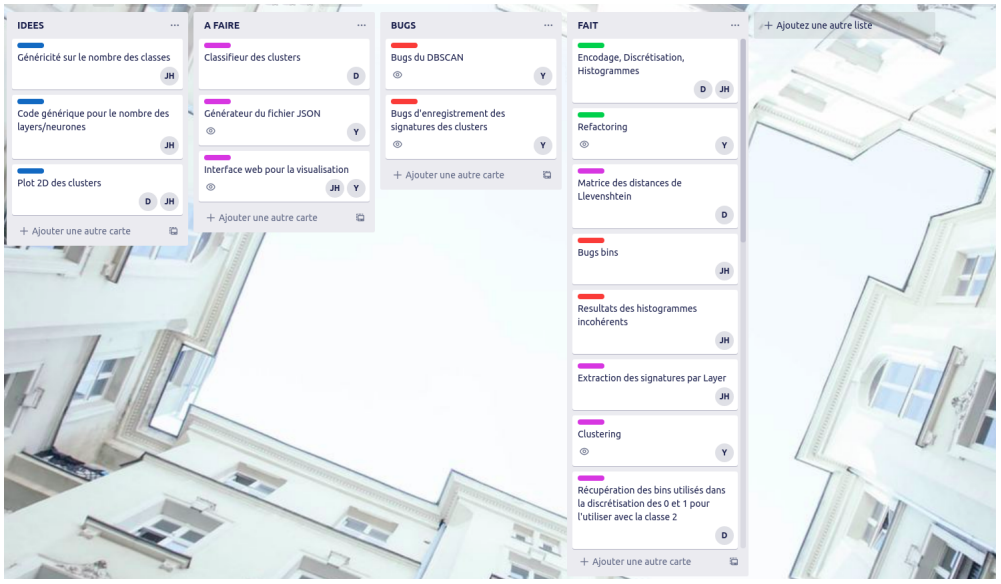

Figure 15 – Projet Trello pour la répartition des tâches

11. https ://trello.com/

Chaque carte représente une tâche et est associée aux membres qui souhaiteront la réaliser.

- **Section "IDEES" :** Dans cette section, chaque membre du groupe peut créer une carte qui sera étiquetée en bleu. Le titre et la description de cette carte est une idée proposé par un membre du groupe qui a pour but l'amélioration du projet.
- **Section "A FAIRE" :** Cette section est composée des cartes représentant les tâches sur lesquelles nous sommes mis d'accord à réaliser avant la prochaine réunion. Ces cartes sont étiquetées en violet.
- **Section "BUGS" :** Cette section représente les bugs que nous devons fixer sous forme des cartes étiquetées en rouge.
- **Section "FAIT" :** cette section est la destination de toutes les cartes qui étaient déjà parmi les cartes d'une des trois sections précédentes et qui ont été accompli.

A Côté de Trello, nous avons créé un répertoire sur Github [12]pour faciliter l'accès au code pour chacun des membres du groupe. Git est un logiciel de gestion de versions décentralisé, libre et distribué selon les termes de la licence publique générale GNU version. Github nous a permis de travailler tous en parallèle sur le projet, chaque membre duplique la branche master et puis commence à travailler sur ses tâches dans sa propre branche. A la fin du travail, nous assemblons toutes les branches dans la branche principale master. la figure 16 montre un extrait de l'historique des commits listant la contribution des membres du groupe dans le projet.

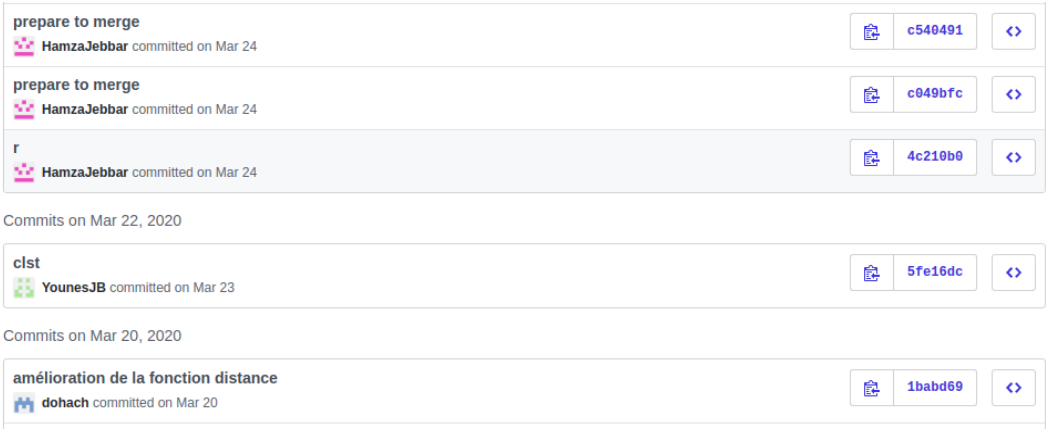

Figure 16 – Un extrait de l'historique des commits

12. https ://github.com/

6 Analyse des résultats

6.1 Vérité terrain

Le réseau doit passer par une phase d'apprentissage qui jouera un rôle dans la classification. Quand le modèle reçoit une donnée d'une classe qui ne fait pas partie de celles qu'il connaisse, il se basera sur l'apprentissage qu'il a effectué pour la classer dans l'une de ces classes.

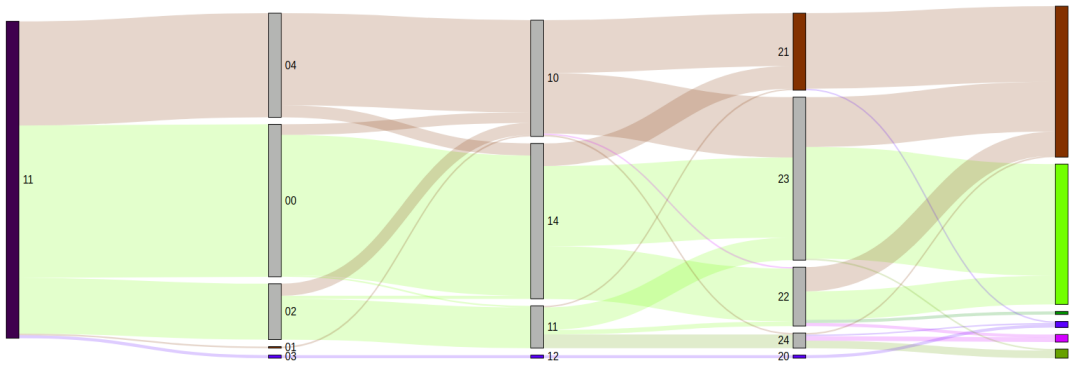

Figure 17 – Exemple de signatures obtenues pour la nouvelle classe (classe 11)

La figure 17 illustre le passage des données d'une nouvelle classe (nommé classe 11) dans le réseau ainsi que les classes qui leurs ont été attribuées. La seule référence qui nous permet de comprendre le résultat du classement serait le passage de la donnée par les clusters d'une couche à l'autre. Quand on voit par exemple que la nouvelle donnée passe des clusters de la classe 0 dans la couche 1 aux clusters de la classe 1 dans la couche 2, des questions sur l'architecture et le fonctionnement du réseau se posent.

6.2 Taille du modèle

Il n'y a pas de règle qui détermine le nombre de couches à utiliser. Ceci dépend de chaque modèle. Pour chaque modèle, il faut travailler avec un nombre 'n' de couches afin d'avoir un résultat de classification des données. À partir d'une couche 'k' les résultats restent inchangeable. Il faut se débarrasser de toutes les couches à partir de la couche 'k+1'. Ceci permettra d'avoir un modèle optimale, performant, plus rapide et plus léger nous évitant toute surcharge inutile.

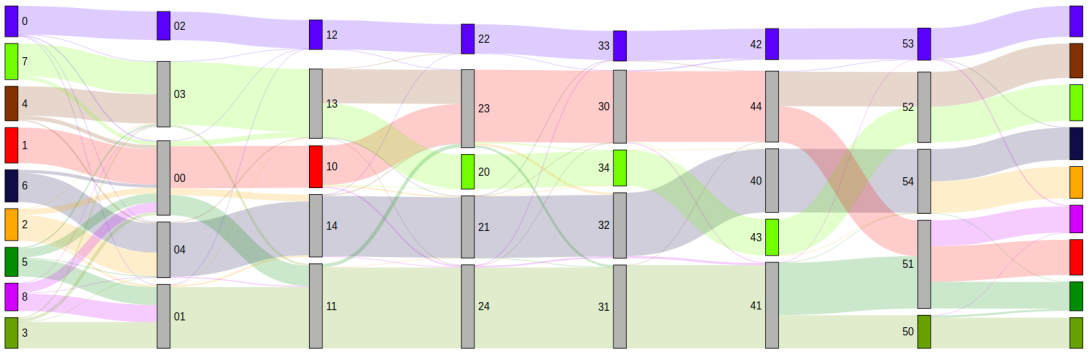

Figure 18 – Architecture d'un modèle avec des couches inutiles

La figure 18 montre qu'à partir de la couche 3, les données se comportent de la même manière tout au long du chemin jusqu'à l'arrivée à la dernière couche, ceci indique que ces couches influencent très peu la décision et peuvent être supprimées tout en gardant la même précision du modèle.

7 Conclusion

Que se passe-t-il quand une nouvelle donnée est passée à un modèle qui ne reconnaît pas son type mais qui la classe selon l'apprentissage qu'il a reçu ? C'est à cette question que nous avons essayé de répondre avec notre application.

Nous avons eu l'occasion de travailler sur un sujet qui porte sur l'apprentissage profond. Le sujet du projet tourne autour de la boîte noire. Il fallait principalement savoir ce qui se passait à l'intérieur de celle-ci afin de découvrir les raisons qui se cachaient derrière la classification d'une nouvelle donnée passée en entrée. Cela peut servir à améliorer la performance du réseau classifiant ces données.

L'apprentissage profond, thème du projet et sujet auquel nous étions confrontés pour la première fois, impliquait l'utilisation de nouveaux outils. Cela dit, rencontrer des difficultés était une évidence. En voici quelques unes qui nous ont marqué :

— Optimisation des algorithmes incontournable vu la grande quantité de données, qui peut être passée au modèle, et le nombre d'opérations effectuées dessus qui peut être considérablement grand.

— Adaptation des fonctions à chaque nouvelle approche tout au long du projet car chaque approche utilise ses propres structures et suit sa propre logique.

— Amélioration de l'affichage de l'interface afin de faciliter le suivi de chaque ensemble de données.

— Réalisation de la communication entre le modèle et l'interface grâce à Flask et à l'utilisation des fichiers Json comme format de données intermédiaire.

Durant ces trois mois de projet nous avons pu :

— Découvrir le réseau de neurones et son fonctionnement relevant ainsi le mystère qui entoure la boîte noire.

— Connaître deux méthodes de résolution : la première via l'interface graphique et la deuxième dans la programmation du plot 2D utilisant l'algorithme PCA.

— Détecter les anomalies en utilisant des classes de vérité terrain et cela via l'interface web.

— Optimiser le modèle en détectant les couches inutiles.

Derrière le fonctionnement de l'interface se cache un code générique qui fonctionne sur différentes architectures indépendamment du nombre de chacun des couches, des neurones et des classes. Par contre, le niveau de généricité du code que nous parvenions à avoir ne permet pas de le tester sur des réseaux complexes tels que les réseaux CNN [13], LSTM [14]...etc Une généricité plus élevée est à prévoir pour ces réseaux dont le fonctionnement diffère selon le domaine d'application.

Une vidéo de démonstration de notre application est disponible à l'URL suivant : `https://www.youtube.com/watch?v=8eeDPJ-iuqc`

Le dépôt Git utilisé durant toute cette période de projet pour y partager le code : `https://github.com/HamzaJebbar/NNPatterns`

13. https ://natural-solutions.eu/blog/la-reconnaissance-dimage-avec-les-rseaux-de-neurones-convolutifs/
14. https ://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm/

Bibliographie

[1] https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1445304-python-definition-et-utilisation-de-ce-langage-informatique/ , [Dernière consultation le 10/05/2020]

[2] https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/What_is_JavaScript , [Dernière consultation le 10/05/2020]

[3] https://docs.python.org/fr/3/library/re.html , [Dernière consultation le 10/05/2020]

[4] https://docs.python.org/fr/2.7/library/os.html , [Dernière consultation le 10/05/2020]

[5] http://www.python-simple.com/python-modules-math/random.php , [Dernière consultation le 10/05/2020]

[6] https://informatique-python.readthedocs.io/fr/latest/Cours/science.html , [Dernière consultation le 10/05/2020]

[7] https://fr.wikipedia.org/wiki/Pandas , [Dernière consultation le 10/05/2020]

[8] https://he-arc.github.io/livre-python/matplotlib/index.html , [Dernière consultation le 10/05/2020]

[9] https://fr.wikipedia.org/wiki/Scikit-learn , [Dernière consultation le 10/05/2020]

[10] https://fr.wikipedia.org/wiki/Keras , [Dernière consultation le 10/05/2020]

[11] https://fr.wikipedia.org/wiki/D3.js , [Dernière consultation le 10/05/2020]

[12] https://fr.wikipedia.org/wiki/Flask_(framework) , [Dernière consultation le 10/05/2020]

[13] https://www.lebigdata.fr/reseau-de-neurones-artificiels-definition , [Dernière consultation le 10/05/2020]

Annexe

```
def makes_Layers(filename) :
    name = filename.split("/")
    name = name[-1].split("_")[1:-1]
    disc_X,y = readXy(filename)
    layers = []
    i=1
    while i<len(name):
        layer = []
        if i == 1:
            ma = int(name[i])
            for X in disc_X:
                layer.append(X[:ma])
        else :
            mi = ma
            ma += int(name[i])
            for X in disc_X:
                layer.append(X[mi:ma])
        layers.append(layer)
        i+=2
    return layers,y
```

Listing 3 – Récupération des signatures

```
app = Flask(__name__)
app.static_url_path='/static'

@app.route("/")
def home():
  return render_template("index.html")
if __name__ == "__main__":
  app.run(debug=True)
```

Listing 4 – Démarrage du serveur Flask

```
{
    "links": [
        {"source": "X0", "target": "C03", "value": "157", "numclass": "C0"},
        {"source": "C03", "target": "C11", "value": "9", "numclass": "C0"},
        {"source": "C11", "target": "C20", "value": "23", "numclass": "C0"},
        ...
        ...
        {"source": "C20", "target": "8", "value": "1", "numclass": "C8"},
        {"source": "X8", "target": "C03", "value": "8", "numclass": "C8"},
        {"source": "C10", "target": "C24", "value": "1", "numclass": "C8"}],
    "nodes": [
        {"name": "X0", "numclass": "C0", "shared": "false"},
        {"name": "C03", "numclass": "C0", "shared": "false"},
        {"name": "C11", "numclass": "C8", "shared": "true"},
        ...
        ...
        {"name": "6", "numclass": "C6", "shared": "false"},
        {"name": "7", "numclass": "C7", "shared": "false"},
        {"name": "8", "numclass": "C8", "shared": "false"}
    ]
}
```

Listing 5 – Exemplaire du fichier Json