



FACULTÉ DES SCIENCES

Rapport de fin d'étude SigLib

"Equipe GAFA" :
Adrien DIDIER
Loïc MARTIN
Pierre PORTAL
Aurélien TROUCHE

Tuteur :
Pascal PONCELET

ANNÉE 2019-2020

Remerciements

Nous adressons nos remerciements aux personnes qui nous ont aidés dans la réalisation de ce projet. Nous remercions Monsieur PONCELET Pascal, professeur à la faculté des sciences de Montpellier. En tant que tuteur du projet, il nous a guidé dans notre travail, nous a donné des conseils, n'a pas hésité à s'investir et à se rendre disponible tout au long du projet.

Nous tenons également à remercier toutes autres personnes qui ont su nous donner des conseils et nous ont aidé à la relecture de notre rapport.

Table des matières

1	Introduction	5
2	Rappel sur les réseaux de neurones : quelles signatures ?	6
3	État de l'art	8
4	Le projet et son contexte	11
5	Rapport technique	13
5.1	Conception	13
5.1.1	Présentation des choix technologiques	13
5.1.2	Contraintes techniques	14
5.2	Architecture du programme	15
5.2.1	Préparation des données	18
5.2.2	Création et apprentissage du modèle	19
5.2.3	Extraction des signatures	21
5.2.4	Discrétisation des signatures	24
5.2.5	Clusterisation des signatures	27
5.2.6	Affichage des résultats	30
5.3	Analyse des résultats	34
6	Rapport d'activité	36
6.1	Organisation du travail	36
6.1.1	Diagramme de Gantt	40
6.1.2	Rôles dans l'équipe	41
6.1.3	Itérations	41
6.2	Outils	42
6.2.1	Gestion des tâches - Trello	42
6.2.2	Discord - TeamViewer	44
6.2.3	Github	46
6.2.4	Anaconda	47
6.3	COVID-19	47
7	Conclusion : synthèse et bilan	49
7.1	Résultats obtenus	49
7.2	Perspectives d'évolution du projet	50
7.3	Bilan technique et humain	51

Glossaire

A

attention Inspiré par l'attention visuelle humaine, un mécanisme d'attention est la capacité d'apprendre à se concentrer sur des parties spécifiques d'une données complexes, par exemple une partie d'une image ou un mot dans un phrase. 50

C

CNN En apprentissage automatique, un réseau de neurones convolutif ou réseau de neurones à convolution 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. 50

D

Deep Learning Le deep learning est un type d'intelligence artificielle dérivé du machine learning où la machine est capable d'apprendre par elle-même (par un réseau de neurones) , contrairement à la programmation où elle se contente d'exécuter à la lettre des règles prédéterminées. 5

Discord Discord est un logiciel propriétaire gratuit de VoIP(technique de transmission de la voix sur les réseaux IP) conçu initialement pour les communautés de joueurs. 44

J

Jupyter Jupyter est une application web utilisée pour programmer dans plus de 40 langages de programmation dont Python. 47

K

Keras La bibliothèque Keras permet d'interagir avec les algorithmes de réseaux de neurones profonds et de machine learning. 5

L

LSTM La mémoire à court terme est une architecture de réseau neuronal récurrent artificiel utilisée dans le domaine de l'apprentissage en profondeur. 50

M

méthodes agile Les méthodes agiles sont des groupes de pratiques de pilotage et de réalisation de projets. 41

P

Product Owner Le Product Owner est responsable de la définition et de la conception d'un produit. Il est chargé de mener à terme un projet en utilisant la méthode scrum. 41

R

Resnet Un réseau neuronal résiduel est un réseau neuronal artificiel qui s'appuie sur des constructions connues des cellules pyramidales du cortex cérébral. Les réseaux de neurones résiduels le font en utilisant des connexions de saut ou des raccourcis pour sauter par-dessus certaines couches. 50

S

Scrum Master Le rôle de Scrum Master est de s'assurer de l'implication de chaque membre et de les aider à franchir les différents obstacles qu'ils pourraient rencontrer. 41

Spyder Spyder est un environnement de développement pour Python. 47

SVM Les machines à vecteurs de support ou SVM sont un ensemble de techniques d'apprentissage supervisé destinées à résoudre des problèmes de discrimination et de régression. 5

T

TeamViewer TeamViewer est un logiciel propriétaire de télémaintenance disposant de fonctions de bureau à distance, de téléadministration, de conférence en ligne et de transfert de fichiers. 44

TensorFlow TensorFlow est un outil open source d'apprentissage automatique développé par Google. 5

t-sne L'algorithme t-SNE est une technique de réduction de dimension pour la visualisation de données. 50

Table des figures

1	Illustration d'un réseau de neurones	6
2	Schéma d'un neurone	7
3	Clusterisation de MNIST	9
4	Évolution inter-couches MNIST après training	10
5	Exemple de signature	11
6	Visualisation d'une signature	12
7	Exemple d'images du jeu de données MNIST	13
8	Performance PC lors de l'utilisation de notre Programme	14
9	Fonctionnement de la solution	15
10	Résultat de la clusterisation d'une signature	16
11	Visualisation d'une signature	17
12	Aperçu d'un résultat de la fonction hidden_layer()	21
13	Résultat de la sauvegarde des signatures d'apprentissage pour un modèle à 3 couches cachées	22
14	Résultat de la sauvegarde des signatures de test	23
15	Exemple de discréétisation avec les intervalles	24
16	Exemple de discréétisation des signatures d'apprentissage	25
17	Exemple de clusterisation des signatures d'apprentissage	27
18	Résultat de la clusterisation d'une signature	28
19	Résultat de la clusterisation des signatures de test	29
20	Exemple d'un diagramme Sankey	30
21	Capture d'écran de l'interface avec l'affichage "vue d'ensemble" . .	31
22	Capture d'écran de l'interface avec l'affichage "signature distincte" .	32
23	Capture d'écran de la bulle "information"	33
24	Capture d'écran de l'interface sur train(0,1) et test(2)	34
25	Capture d'écran de l'interface montrant le surplus de couches . .	35
26	Fonctionnement Méthode Scrum	36
27	Valeurs Agile	38
28	Diagramme de Gantt	40
29	Copié d'écran du Trello	43
30	Copie d'écran du groupe de discussion Discord	44
31	Partage d'écran via TeamViewer	45
32	Historique des commits	46
33	Logo Anaconda	47
34	Charge de Travail	48
35	Capture d'écran de l'interface avec l'affichage "vue d'ensemble" .	49
36	Probabilité obtenu à l'aide du modèle attention	50
37	Compte rendu réunion	54

1 Introduction

De nos jours, avec l'explosion de l'utilisation de données et la disponibilité de ses dernières, le Deep Learning a apporté des améliorations significatives dans des domaines tels que la classification d'image, l'analyse des sentiments ainsi que la compréhension de la parole. Il existe aujourd'hui de multiples librairies comme Keras et TensorFlow qui permettent de créer des réseaux complexes très rapidement. Néanmoins, certains modèles d'apprentissage automatique s'appliquent à la manière d'une boîte noire, c'est-à-dire qu'il n'est pas possible de connaître en détail l'information qui a conduit à la prédiction obtenue.

En effet de nombreuses critiques sont émises pour des algorithmes d'apprentissage comme SVM ou les réseaux de neurones qui justement se comportent comme des boîtes noires. Avec le développement du Deep Learning, de plus en plus de personnes veulent en savoir plus sur ces modèles de "boîtes noires". Pourquoi ? Les raisons sont nombreuses. Nous pouvons prendre comme exemple la confiance réellement accordée à la prédiction du modèle. Il y a peu de temps des auteurs ont montré qu'un système entraîné à prédire le risque de pneumonie arrivait à des conclusions totalement fausses. Le modèle avait appris que les patients asthmatiques souffrant de problèmes cardiaques couraient un risque beaucoup plus faible de mourir de pneumonie que les personnes en bonne santé.

L'objectif de ce TER est donc de mieux comprendre le fonctionnement d'un modèle de réseau de neurones profond. Ainsi, nous devons développer des outils permettant l'analyse des prédictions de modèle. Pour se faire nous allons extraire une signature pour chaque prédiction, cette signature est constituée d'une trace de la façon dont le réseau de neurones a établi la prédiction.

Cette analyse permet de répondre à plusieurs questions :

- Si le jeu d'apprentissage ne contient que des 0 et des 1 quels sont les neurones qui sont activés et comment ? Que se passe-t-il si le modèle est appliqué sur un 2 ?

- Existe-t-il des signatures caractéristiques de certaines données ?

Dans ce rapport, nous répondons à ces questions. Une vidéo de démonstration de notre application est disponible depuis le lien suivant :

<https://www.youtube.com/watch?v=Jitaq5fWRUA>

Le reste du rapport est organisé de la manière suivante. Dans la partie 2 nous allons réaliser un rappel sur les réseaux de neurones ainsi que réaliser une présentation de l'état de l'art avec la partie. 3.

Nous détaillerons le projet et son contexte, ainsi que l'analyse que nous avons dû réaliser pour mener à bien le travail proposé dans la partie 4.

Nous continuerons par le rapport technique grâce à la partie 5, cette partie permettra de présenter nos choix technologiques et la réalisation de notre projet. La partie 5.3 permettra d'analyser les résultats obtenus.

De plus, il sera présenté une partie, dédiée à la gestion de ce projet. Nous conclurons en réalisant une synthèse et un bilan technique et humain.

2 Rappel sur les réseaux de neurones : quelles signatures ?

Dans cette partie, nous allons vous présenter ce qu'est un réseau de neurones afin d'acquérir les connaissances nécessaires pour la compréhension de ce projet.

Tout d'abord, un réseau de neurones est un système qui se rapproche beaucoup du fonctionnement du cerveau humain. Les réseaux de neurones font partie des méthodes d'apprentissage profond, elles-mêmes faisant partie de l'apprentissage automatique. Comme nous pouvons l'observer ci-dessous, un réseau de neurones se découpe en plusieurs couches (layers). Ces couches sont composées de multiples neurones qui travaillent parallèlement. Un neurone artificiel, tel qu'un neurone biologique, reçoit plusieurs stimulus via les poids. Il analyse les informations obtenues et fournit un résultat au neurone suivant.

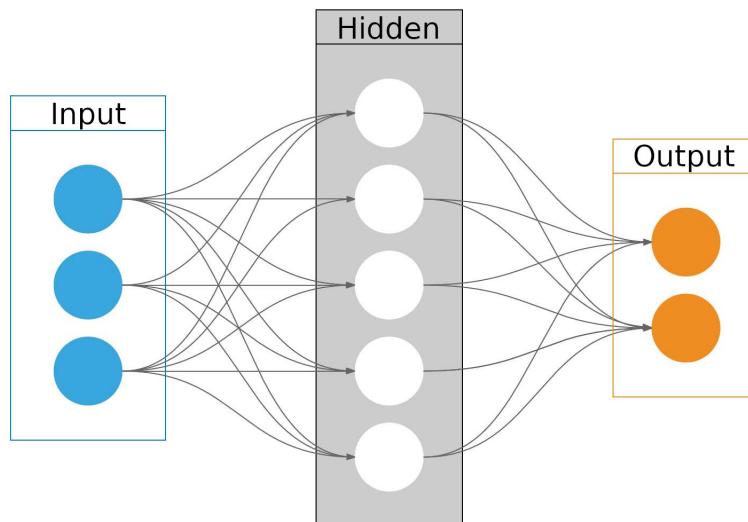


FIGURE 1 – Illustration d'un réseau de neurones

Le réel but du réseau de neurones est d'apprendre à partir d'un jeu de données tel qu'une collection d'images de chats et de chiens. Le réseau va utiliser ce jeu de données afin de s'entraîner sur ces images et va analyser les patterns pour mieux reconnaître un type d'images.

Il doit être capable de prédire la valeur d'une autre image, car ses images sont étiquetées afin que le réseau puisse connaître la nature de l'image. Toutefois, si l'utilisateur lui propose des images de voitures à prédire, le réseau ne pourra

répondre que par chat ou chien. C'est ce que nous appelons de l'apprentissage supervisé, l'algorithme s'entraîne sur des données étiquetées et essaye d'obtenir le résultat souhaité.

Comme vu précédemment, un réseau de neurones classique est composé de plusieurs couches de neurones. La première couche est la couche d'entrée, elle possède un nombre de neurones égaux au nombre de features de nos données et chaque neurone prend une valeur en entrée. Par exemple, si nos données sont des images qui ont toutes la même taille, le nombre de pixels d'une image constitue le nombre de features et chaque neurone de la couche d'entrée prendra en entrée la valeur d'un pixel de l'image. Les couches suivantes sont les couches cachées, elles sont composées d'un nombre arbitraire de neurones. Enfin, nous avons la couche de sortie qui va produire une prédiction.

Chaque neurone de chaque couche reçoit des informations de l'ensemble des neurones de la couche précédente et envoie des informations à l'ensemble des neurones de la couche suivante.

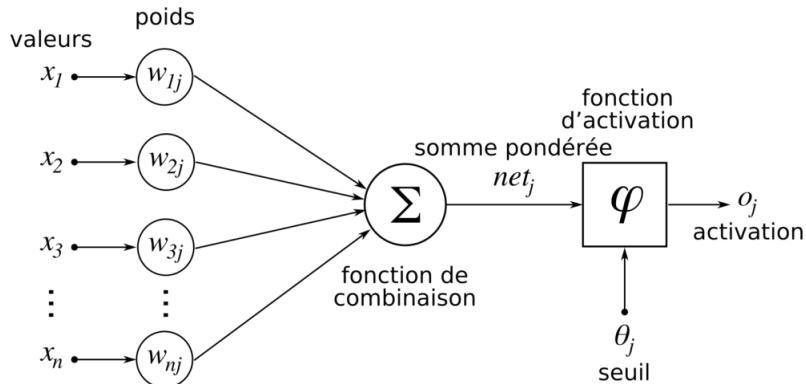


FIGURE 2 – Schéma d'un neurone

Sur la figure 2, les valeurs x_1, \dots, x_n représentent les valeurs envoyées par les neurones de la couche précédente. Chacune de ces valeurs est multipliée par un poids avant d'être envoyée au neurone. Le neurone va effectuer la moyenne de toutes ces valeurs pour enfin exécuter une fonction d'activation sur cette moyenne. Le résultat de la fonction d'activation sera envoyé à tous les neurones de la couche suivante.

La phase d'apprentissage consiste à nourrir le réseau de neurones avec des données d'apprentissage labélisées afin d'ajuster la valeur des différents poids au moyen d'un algorithme de descente de gradient. Une fois cette phase terminée, le réseau de neurones est capable de prédire les classes de données qu'on lui a fait apprendre. Le lecteur souhaitant approfondir cette partie peut se reporter à : *Réseau de neurones artificiels*.

3 État de l'art

Énormément de scientifiques et de passionnés tentent de comprendre le fonctionnement des réseaux de neurones ainsi que d'obtenir un moyen de visualiser efficacement les comportements obtenus. De ce fait, de multiples projets ayant pour sujet l'analyse de l'activité dans les couches cachées ont été réalisés.

Nous pouvons entre autres citer cet article : *Visualizing the Hidden Activity of Artificial Neural Networks* qui nous a été proposé par notre encadrant Monsieur PONCELET.

Cet article propose une analyse entre autres sur le jeu de données MNIST et pousse la recherche assez loin concernant la compréhension du comportement des couches cachées.

En effet, le travail réalisé se rapproche énormément de celui que nous avons effectué, mais nous pouvons toutefois observer une manière vraiment originale d'afficher le résultat de la clusterisation.

La figure 3 représente la clusterisation des différentes images proposées par MNIST. Nous pouvons observer qu'une bonne partie des images est assez bien classée même si certaines sont assez éparpillées alors que le modèle ne s'est pas encore entraîné dessus. Ceci s'explique par le fait que MNIST est une référence de classification assez simple.

Le modèle atteint une précision de 83.78% sans entraînement et il atteint 98.36% suites à l'entraînement. Nous pouvons observer que les clusters sont bien plus compacts et séparés les uns des autres. Mais il existe tout de même des erreurs telles que la prédiction de l'image représentant un 3 visible sur la figure 3-b (outlier).

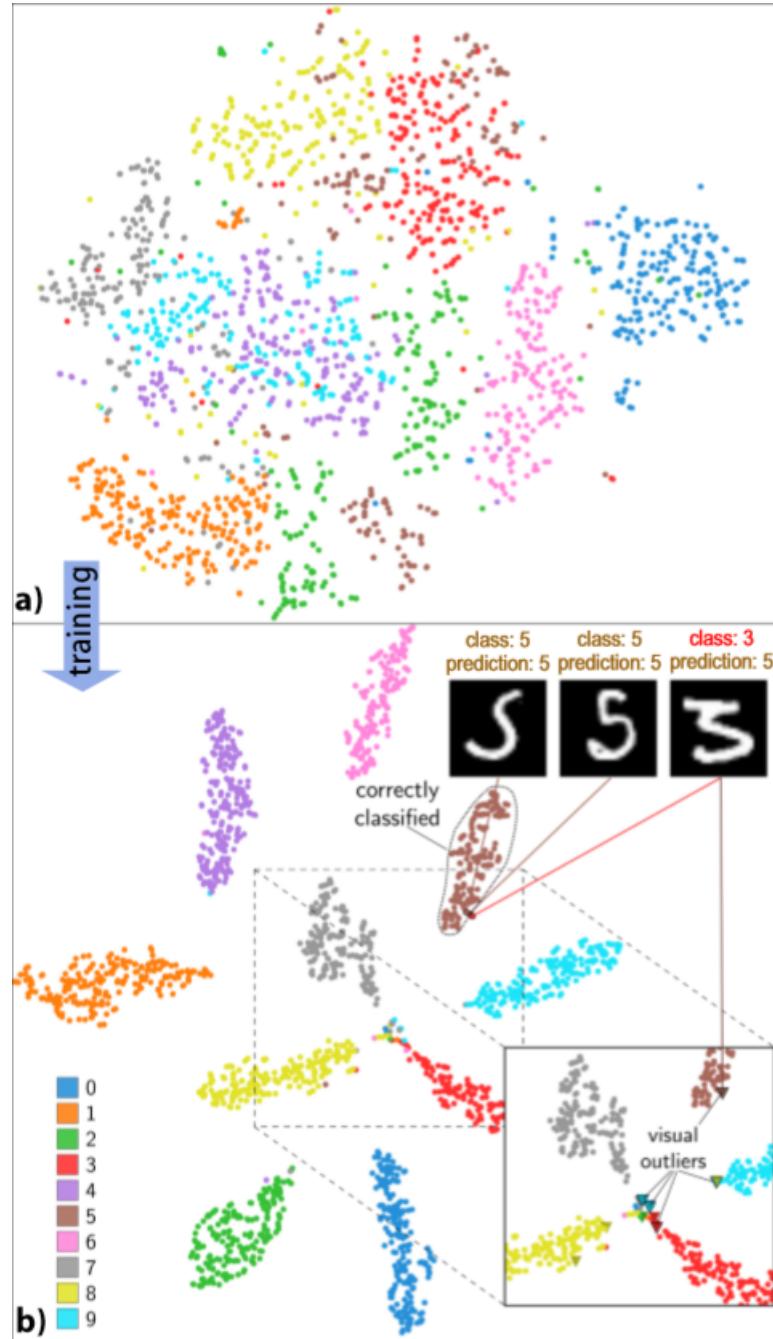


FIGURE 3 – Clusterisation de MNIST

La figure 4 représente quatre clusterisations, une par couche cachée. La grande figure représente la combinaison de ses quatre dernières. Les couleurs représentent les classes et la luminosité le numéro de couche. Ainsi, nous pouvons observer comment se comportent les données des fonctions d'activation au sein des quatre couches. Nous visualisons également que les clusters sont assez compactes et que le réseau réalise une bonne séparation des clusters dès la première couche. De ce fait, les couches de réseau ultérieures renforcent la cohérence et la séparation de classe obtenues par la première couche. De plus, nous pouvons observer que quelques arêtes relient d'autres clusters. Par conséquent, seules quelques données changent de clusters au fur et à mesure que les données d'activation circulent à travers le réseau. En résumé, il est déductible que les couches réseau après la première couche complètent la cohérence du cluster et les données sont majoritairement correctement classifiées dès la première couche.

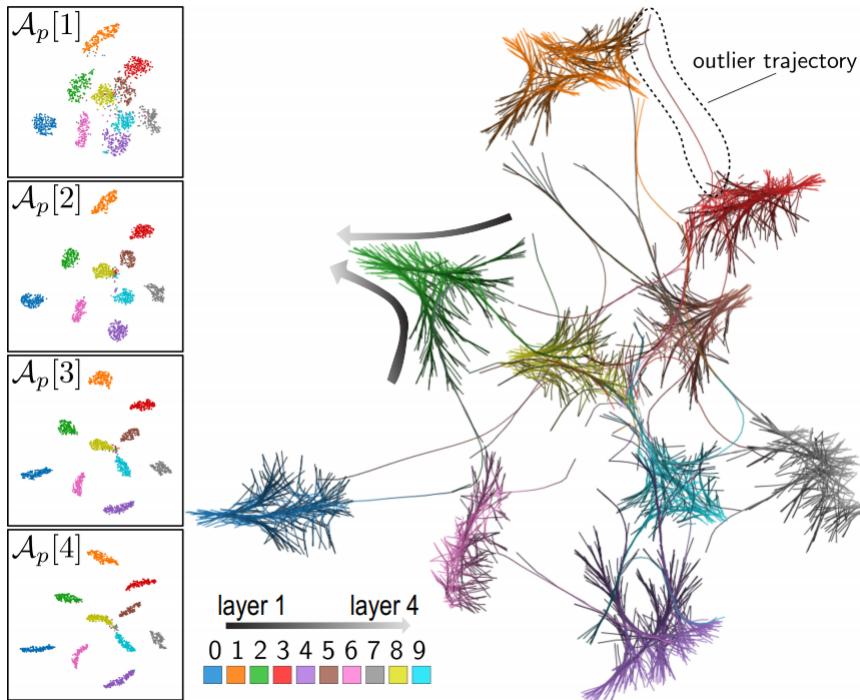


FIGURE 4 – Évolution inter-couches MNIST après training

4 Le projet et son contexte

Le but de notre projet de TER est de pouvoir comprendre ce qu'il se passe dans un réseau de neurones. Cette envie de compréhension s'explique par l'utilisation de librairies très efficaces comme Keras et TensorFlow qui facilitent grandement l'implémentation de modèles de Deep Learning et qui permettent d'obtenir des informations très précises sur leur fonctionnement.

La méthode d'apprentissage par réseau de neurones a la particularité d'être un modèle boîte noire, c'est-à-dire qu'il est impossible de relier simplement les entrées du réseau (les features) avec la sortie, la décision. Cela s'explique par la multitude d'interconnexions entre les différentes couches de neurones qui empêche de fournir une justification facile à interpréter. Dès lors, il semble approprié de se demander quelles pourraient être les pistes pour essayer de comprendre comment fonctionne ce mécanisme d'apprentissage.

La piste que nous avons décidé d'explorer est l'analyse des signatures des données prédites par le modèle. Une signature est une trace de la façon dont le réseau de neurones a établi une prédiction. Pour obtenir cette trace, il est nécessaire de sauvegarder les informations du modèle qui permettent de caractériser cette prédiction.

Comme nous avons vu précédemment, un réseau de neurones est composé de plusieurs couches de neurones. Chaque neurone renvoie une valeur par sa fonction d'activation aux neurones de la couche suivante. L'idée pour établir la signature d'une prédiction est d'enregistrer toutes les valeurs de retour des fonctions d'activation de chaque neurone de chaque couche.

Classe de données d'entrée	5
Valeurs de retour des fonctions d'activation pour chaque couche	[0.81, 0, ..., 0.14]
	[0.98, 1.37, ..., 0.21]
	[0.012, 0.11, ..., 0]
Classe prédite	0

FIGURE 5 – Exemple de signature

Une fois ces signatures sauvegardées nous pouvons commencer à les analyser. Cette phase d'analyse se fait au moyen d'algorithmes mathématiques de discréétisation et de clusterisation. On cherche à dégager des classes de signatures dans le but de se les représenter plus simplement et de trouver des similarités ou des anomalies entre les signatures des différentes classes d'instance que l'on a voulu prédire. Ces classes de signatures pourront alors être représentées au moyen d'une interface graphique.

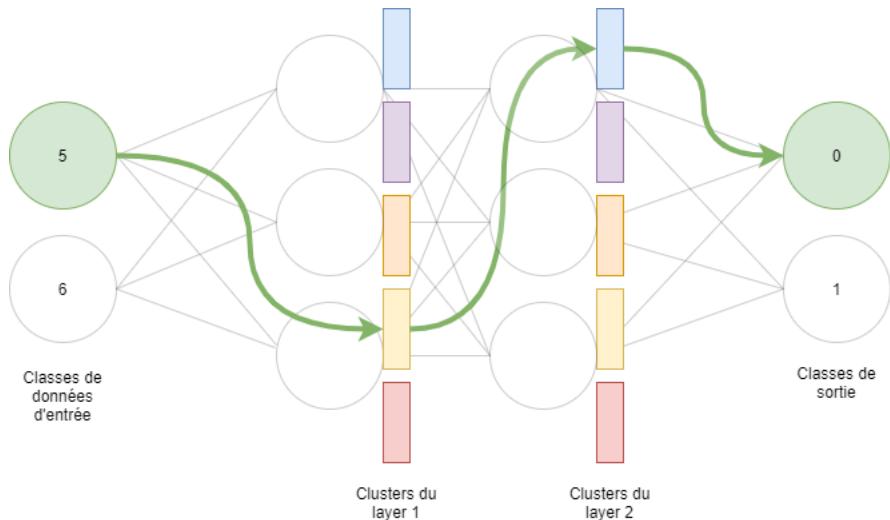


FIGURE 6 – Visualisation d'une signature

Le logiciel est destiné au cadre universitaire. Il sera utilisé par d'autres groupes de TER travaillant également sur des réseaux de neurones. Il pourra également être fourni comme support de compréhension expliquant une partie des réseaux de neurones pour des étudiants débutant dans ce domaine de l'informatique.

L'encadrant de ce projet, ici Monsieur Poncelet, nous a donné un certain nombre d'objectifs à atteindre au terme de notre projet. Premièrement la compréhension des réseaux de neurones est ici indispensable pour comprendre le fonctionnement de ces boîtes noires. En parallèle de la lecture des documents fournis, nous avons implémenté une classe MyDeepLearning en Python qui nous a permis de mettre en application nos connaissances.

Ensuite, nous avons étudié les méthodes d'extraction des signatures des modèles obtenues ainsi que des façons de représenter et comparer ces signatures.

Une fois l'implémentation terminée nous avons mis en place un système de visualisation web permettant une meilleure compréhension générale du programme.

5 Rapport technique

5.1 Conception

5.1.1 Présentation des choix technologiques

Outre la partie affichage graphique, ce projet est réalisé uniquement à l'aide du langage Python. Le choix de ce langage nous a été imposé, mais on peut parfaitement le justifier du fait de la présence d'une multitude de documentations, d'outils et de librairies spécialisées autour de la science des données et de l'intelligence artificielle.

En tant qu'environnement de programmation Python, nous utilisons le logiciel Spyder qui est inclus dans la distribution Anaconda. Spyder est conçu pour les scientifiques, les ingénieurs et les data-analystes. Il offre de nombreuses fonctionnalités de débogages telles que, par exemple, l'exploration des valeurs des variables en temps réel. Spyder inclut nativement des librairies scientifiques populaires que nous utilisons et que nous allons présenter.

Nous utilisons essentiellement la base de données MNIST comme jeu de données, elle regroupe 60000 images sous la forme de chiffres écrits à la main et elle est très utilisée dans le domaine de l'apprentissage automatique. Ce jeu de données servira d'exemple tout au long de la présentation du rapport technique.

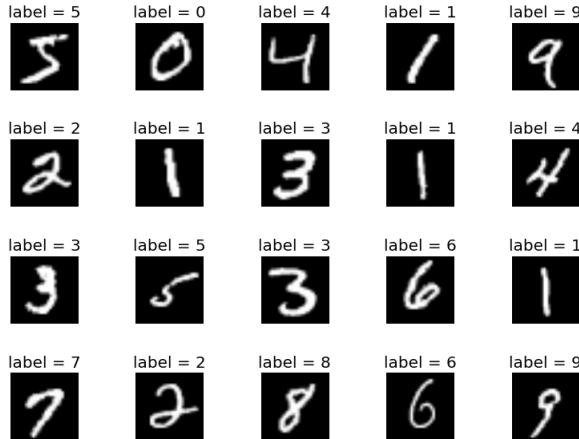


FIGURE 7 – Exemple d'images du jeu de données MNIST

Pour la création des réseaux de neurones, nous utilisons la librairie Keras qui est construite sur TensorFlow (l'API open-source de Google) et qui permet de prototyper simplement et rapidement un modèle de Deep Learning tout en offrant de très bonnes performances.

La structure de données que nous utilisons pour représenter les données extraites (signatures au format brut/discrétisées/clusterisées) est la classe DataFrame de la librairie Pandas, cette librairie offre une panoplie d'outils permettant l'analyse et la manipulation des données. La structure d'un DataFrame correspond à un dictionnaire dont les clés sont les noms de colonnes et dont chaque ligne est une séquence de données.

Nous utilisons aussi la librairie numpy qui met à disposition la classe array qui permet la création d'un tableau à n-dimension très puissant et sur lequel il est possible de construire un DataFrame. De plus, numpy propose de nombreux outils pour manipuler ces tableaux.

5.1.2 Contraintes techniques

Tout au long de ce projet, nous avons rencontré des contraintes plus ou moins handicapantes pour le développement de celui-ci. Cela commence par la taille de l'ensemble des CSVs produits, en effet la taille de ceux-ci a parfois atteint plusieurs Go. Heureusement, l'ensemble de l'équipe possédant des ordinateurs assez puissants, cela n'a pas posé forcement de problèmes.

Lié à ce phénomène de place, les ordinateurs ont été soumis à de grosses charges de calculs, cela se calcule à approximativement 6 Go de RAM pendant l'exécution du programme.

In use (Compressed)	Available	Speed:	2400 MHz
6.1 GB (41.6 MB)	9.6 GB	Slots used:	2 of 4
Committed	Cached	Form factor:	DIMM
9.2/18.3 GB	4.5 GB	Hardware reserved:	50.6 MB
Paged pool	Non-paged pool		
597 MB	283 MB		

FIGURE 8 – Performance PC lors de l'utilisation de notre Programme

5.2 Architecture du programme

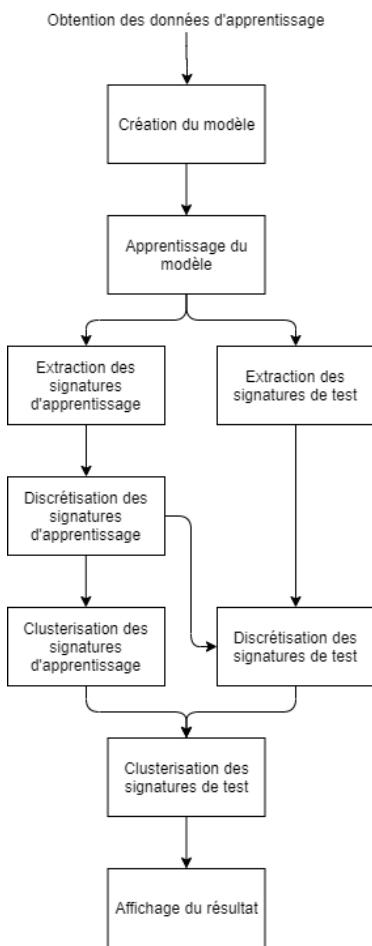


FIGURE 9 – Fonctionnement de la solution

Ce graphe décrit le déroulement des différentes étapes de notre solution qui seront pleinement expliquées dans les parties suivantes de ce rapport.

Les explications sur l'obtention des données d'apprentissage, sur la création du modèle et sur l'apprentissage du modèle ne semblent peut-être pas appropriées du fait que notre projet se concentre sur l'extraction des signatures et des étapes qui viennent ensuite. Cependant, nous avons jugé judicieux d'inclure ces explications pour permettre une vue d'ensemble et de donner un exemple complet d'application de notre solution avec l'utilisation du jeu de données MNIST.

L'extraction des signatures se fait en deux temps. On va d'abord extraire les signatures des données qui ont permis l'apprentissage du modèle puis on va extraire les signatures des données utilisées pour tester le modèle. Ces données peuvent être différentes, on peut avoir lancé un apprentissage sur les classes de données [0, 1, 2] puis tester le modèle sur les classes de données [5, 6, 7], cela a des fins d'expérimentation.

Une signature est constituée de la classe de données en entrée, des données de retour des fonctions d'activation pour chaque couche et enfin de la classe de données en sortie. La taille d'une signature dépend donc du nombre de couches du réseau de neurones.

Toutes les analyses de signature des données de test se feront en fonction des données d'apprentissage. La discrétisation des données de test se fera en fonction des intervalles de discrétisation générés par la discrétisation des données

d'apprentissage. Pour la clusterisation, on va d'abord clusteriser les données d'apprentissage puis on va prédire, pour chaque signature des données de test, le cluster, précédemment calculé, qui correspond à cette signature.

La phase de discréétisation et de clusterisation se fait sur les données de sortie pour chaque couche cachée. Ainsi, à la fin de ces phases, on est censé obtenir une famille de clusters pour chaque couche. À partir de ces familles de clusters, on pourra dès lors définir une signature simplifiée pour chaque signature de données de test.

	Signature brute		Signature simplifiée
Classe de données en entrée	5		
Valeurs de retour des fonctions d'activation pour chaque couche	[[0, 0.125, 0.654, ..., 0, 0.512, 0.123], [0, 0, 0.214, ..., 0.112, 0.91, 0], [0, 0.150, 0.421, ..., 0.012, 0.512, 0]]		[0, 2, 4]
Classe de données en sortie	0		0

FIGURE 10 – Résultat de la clusterisation d'une signature

Chaque partie du programme demande une puissance de calcul conséquente et beaucoup de temps, même sur des ordinateurs performants. De ce fait, il semble nécessaire de mettre une emphase sur la sauvegarde des données pour chacune des parties.

L'affichage consiste à visualiser les résultats de la clusterisation des signatures. On souhaite d'abord représenter la structure du réseau de neurones, c'est-à-dire le nombre de features, le nombre de couches et le nombre de neurones dans chaque couche. Par-dessus cette visualisation, on souhaite montrer les différents clusters pour chaque couche et enfin les signatures simplifiées qui passent d'une classe de données en entrée puis qui passent par les clusters de chaque couche pour arriver vers une des classes de sortie.

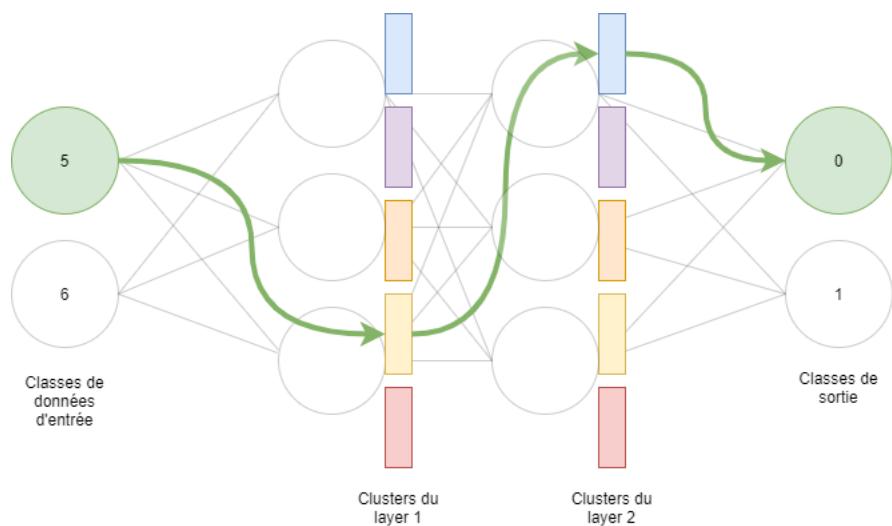


FIGURE 11 – Visualisation d'une signature

5.2.1 Préparation des données

Comme nous avons vu précédemment, les données qui vont être utilisées proviennent de la base de données MNIST qui contient des images de chiffres écrits de façon manuscrite. Ces images sont en noir et blanc et leur format est 28x28, chaque pixel contient une valeur comprise entre 0 et 255. Pour charger ces données, il est d'abord nécessaire d'installer la librairie MNIST puis d'utiliser la fonction `load_data()`.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Elle renvoie 4 objets `numpy.array` différents :

- `X_train`, un tableau à trois dimensions au format [60000 x 28 x 28], il contient l'ensemble des valeurs des pixels pour l'ensemble des 60000 images de test.
- `y_train`, un tableau à une dimension de taille 60000 qui contient la classe de chaque image.
- `X_test` et `y_test`, ces tableaux sont du même format que les précédents et contiennent les données de 10000 images de test qui serviront à tester la précision de notre réseau de neurones.

Les features de notre réseau de neurones vont correspondre aux valeurs des pixels de chaque image. Ces features doivent être représentées sous la forme d'un tableau à une dimension dont la taille est le nombre de features. Cependant, les données d'une image sont modélisées par un tableau à double dimension au format [28 x 28]. Il est alors nécessaire de transformer ces tableaux à deux dimensions (28 x 28) en tableaux à une dimension ($28 * 28 = 784$). Pour cela, nous utilisons la fonction `reshape` fournie par `numpy`, on a simplement à passer en argument le format de tableau souhaité. Aussi, on adapte les features pour qu'elles correspondent à des valeurs entre 0 et 1.

Puisque ce projet est réalisé à des fins d'expérimentation, il est nécessaire d'avoir un moyen simple pour sélectionner les données que l'on veut en fonction de leurs classes. Ainsi, si on veut expérimenter notre modèle uniquement sur certaines classes de données, on utilise la fonction `get_dataset_values` qui va sélectionner ces dernières.

Un exemple pour sélectionner les données d'apprentissage correspondantes aux classes [1, 3, 9] et les données de test correspondantes aux classes [3, 4, 5] :

```
train_values = [1,3,9]
test_values=[3,4,5]
X_train_values,
y_train_values = misc.get_dataset_values(train_values, X_train, y_train)
X_test_values,
y_test_values = misc.get_dataset_values(test_values, X_test, y_test)
```

Ainsi, on a sélectionné les données d'apprentissage que l'on souhaite utiliser

et qu'on a mis au bon format. On va maintenant voir comment implémenter un modèle de réseau de neurones.

5.2.2 Création et apprentissage du modèle

La création d'un modèle se fait au moyen de la classe Sequential de keras qui permet la création d'un réseau de neurones classique. Ce modèle implémente une pile à laquelle on peut ajouter un nombre arbitraire de couches de neurones.

```
model = Sequential()
```

Pour ajouter une couche à cette pile, on utilise la fonction add() sur le modèle en mettant la couche à ajouter en argument. Il existe différents types de couches de neurones modélisées par différentes classes, mais nous utiliserons uniquement la classe Dense qui permet de créer une couche de neurones classique. Dense prend comme arguments le nombre de neurones de la couche ainsi que le type de fonction d'activation. Il existe de nombreux types de fonctions d'activation comme relu, sigmoid et softmax. Ici, on utilise relu pour les couches cachées et softmax pour la couche de sortie.

Softmax permet d'implémenter un réseau de neurones à classes multiples, c'est-à-dire que le réseau de neurones peut prédire une classe parmi plus de deux classes. Softmax va d'abord établir une probabilité pour chacune des classes puis il va sélectionner la probabilité la plus haute pour faire sa prédiction. Imaginons un modèle avec une couche de sortie utilisant softmax avec 3 neurones, lors de la prédiction d'une valeur il y aura un seul neurone de la couche de sortie avec la valeur 1 et le reste avec la valeur 0 ce qui donne, par exemple, une prédiction sous la forme d'un tableau [0, 0, 1]. Si on applique l'index de la valeur 1 du tableau aux valeurs d'apprentissage précédemment sélectionnées (train_values = [1, 3, 9]), on peut conclure que le modèle a prédit que la valeur est un 9.

Il n'est pas nécessaire de créer une couche d'entrée, celle-ci est induite en passant une valeur à l'argument input_dim dans la première couche ajoutée dans le modèle.

```
model.add(Dense(512, input_dim = 754 , activation = 'relu'))  
model.add(Dense(512, activation = 'relu'))  
model.add(Dense(10, activation='softmax'))
```

Une fois les couches de neurones ajoutées, on initialise le modèle avec la fonction compile qui prend en argument une fonction de cout et un optimiseur, ce dernier permet de déterminer la méthode de descente de gradient qui va être utilisé. Ici on utilise l'optimiseur adam mais il en existe bien d'autres.

```
model.compile(loss = 'binary_crossentropy', optimizer = 'adam')
```

L'apprentissage du modèle consiste à l'exécution de la fonction fit.

```
model.fit(train_X, train_y, epochs = 40, batch_size = 100)
```

On lui passe en paramètre les données d'apprentissage (`X_train` et `y_train`) ainsi que les paramètres d'apprentissage suivant :

- le paramètre `epochs`, un nombre qui définit le nombre d'itérations d'apprentissage
- le paramètre `batch_size`, un nombre qui définit le nombre de données d'apprentissage par itération

Ces paramètres sont d'abord définis arbitrairement, puis on les affine en fonction de la précision du modèle. Lors de l'apprentissage, la précision du modèle (`accuracy`) est affichée à chaque fin d'itération. On s'aperçoit que cette valeur évolue rapidement lors des premières itérations puis elle stagne. Cela permet de connaître le nombre d'`epochs` optimal et ainsi réduire le temps d'apprentissage pour nos futurs modèles.

```

Epoch 1/15
6000/6000 [=====] - 2s 258us/step - loss: 0.0655 - acc: 0.9769
Epoch 2/15
6000/6000 [=====] - 1s 125us/step - loss: 0.0141 - acc: 0.9963
Epoch 3/15
6000/6000 [=====] - 1s 123us/step - loss: 0.0062 - acc: 0.9976
Epoch 4/15
6000/6000 [=====] - 1s 123us/step - loss: 0.0028 - acc: 0.9989
Epoch 5/15
6000/6000 [=====] - 1s 126us/step - loss: 0.0097 - acc: 0.9967
Epoch 6/15
6000/6000 [=====] - 1s 131us/step - loss: 0.0014 - acc: 0.9996
Epoch 7/15
6000/6000 [=====] - 1s 131us/step - loss: 0.0011 - acc: 0.9996
Epoch 8/15
6000/6000 [=====] - 1s 126us/step - loss: 0.0103 - acc: 0.9969
Epoch 9/15
6000/6000 [=====] - 1s 122us/step - loss: 0.0012 - acc: 0.9996
Epoch 10/15
6000/6000 [=====] - 1s 122us/step - loss: 9.3245e-05 - acc: 1.0000
Epoch 11/15
6000/6000 [=====] - 1s 123us/step - loss: 3.3326e-05 - acc: 1.0000
Epoch 12/15
6000/6000 [=====] - 1s 123us/step - loss: 1.6759e-05 - acc: 1.0000
Epoch 13/15
6000/6000 [=====] - 1s 120us/step - loss: 1.1572e-05 - acc: 1.0000
Epoch 14/15
6000/6000 [=====] - 1s 123us/step - loss: 8.7055e-06 - acc: 1.0000
Epoch 15/15
6000/6000 [=====] - 1s 123us/step - loss: 6.8682e-06 - acc: 1.0000

```

Listing 1 – Affichage des itérations lors de l'apprentissage d'un modèle

Une fois que l'apprentissage du modèle a été effectué, on doit maintenant générer les signatures des données d'apprentissage, c'est-à-dire extraire les valeurs de retour des fonctions d'activation de chaque couche pour chaque donnée d'apprentissage.

5.2.3 Extraction des signatures

Keras permet l'implémentation de fonctions backend qui permettent de retourner certaines informations du modèle. Les informations que l'on souhaite obtenir sont les valeurs de retour des fonctions d'activation de chaque couche. On implémente cela dans la fonction get_result_layers.

```
def get_result_layers(model, X):
    result_layers = []
    for i in range(len(model.layers) - 1):
        hidden_layers = keras.backend.function(
            [model.layers[0].input],
            [model.layers[i].output])
        result_layers.append(hidden_layers([X])[0])
    return result_layers
```

On crée une fonction backend de Keras pour chaque couche du modèle qu'on appelle hidden_layers(), cette fonction va retourner deux informations : les classes de valeurs en entrée (model.layers[0].input) ainsi que les valeurs de retour des fonctions d'activation de la couche correspondante (model.layers[i].output). Pour utiliser cette fonction, on lui passe en entrée les données à prédire (X).

Classes de valeurs d'entrée	Valeurs de retour des fonctions d'activation d'une couche à 10 neurones									
1	0.369597	0	3.080488	3.490323	5.510371	0.960733	0	5.532372	3.405935	1.004072
9	0.664087	0	0	5.517802	2.555494	2.454282	3.433251	3.340768	5.155586	3.627028
1	0.258077	0	2.179366	3.801541	5.352765	0.683848	0	4.70731	2.31834	0
3	2.991318	0	0	5.234942	8.467753	5.673901	1.380358	2.534644	7.663965	2.468694
1	0.297309	0	1.948201	2.409205	3.845151	0.449998	0	3.733992	1.97775	0
3	3.771743	0	0	5.477636	7.557581	4.045252	1.595868	3.910891	6.631868	1.735779
3	4.099842	0	0	2.242885	10.19904	5.694582	0	0.012272	5.175723	0
1	0.348747	0	2.453466	2.766645	4.281275	0.50688	0	4.019807	1.827492	0
9	2.624012	0	1.685075	4.441686	3.682135	2.792336	1.731102	2.328723	3.906555	2.310038
9	1.423208	0	1.57865	4.01103	2.637979	2.237055	1.790494	2.193746	3.010198	2.726072

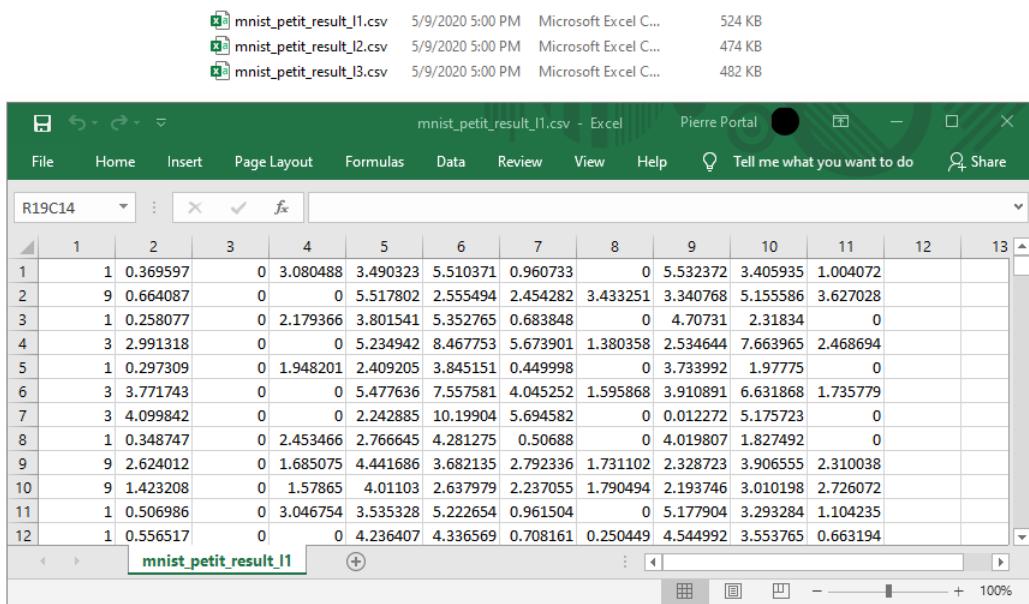
FIGURE 12 – Aperçu d'un résultat de la fonction hidden_layer()

À la fin de la fonction get_result_layers, le tableau retourné contient l'ensemble des valeurs de retour des fonctions d'activation pour toutes les couches du modèle.

La fonction `get_result_layers` est implémentée dans la fonction `generate_train_signatures` qui permet d'extraire puis de sauvegarder les signatures d'apprentissage. Cette phase d'extraction se fait uniquement sur des valeurs qui ont été prédites correctement. Cela permet d'avoir une meilleure homogénéité entre les signatures d'apprentissage afin d'obtenir de meilleurs résultats lors de la clusterisation de ces dernières.

```
generate_train_signatures(train_X, train_y,
                           model, train_values,
                           directory_name)
```

Les valeurs de retour de fonctions d'activation sont sauvegardées dans des fichiers csv distincts pour chaque couche.



The screenshot shows the Microsoft Excel interface with three CSV files listed in the ribbon:

- `mnist_petit_result_l1.csv` (5/9/2020 5:00 PM, Microsoft Excel C..., 524 KB)
- `mnist_petit_result_l2.csv` (5/9/2020 5:00 PM, Microsoft Excel C..., 474 KB)
- `mnist_petit_result_l3.csv` (5/9/2020 5:00 PM, Microsoft Excel C..., 482 KB)

The `mnist_petit_result_l1.csv` file is currently open in a new Excel window. The data is presented in a grid format:

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	0.369597	0	3.080488	3.490323	5.510371	0.960733	0	5.532372	3.405935	1.004072		
2	9	0.664087	0	0	5.517802	2.555494	2.454282	3.433251	3.340768	5.155586	3.627028		
3	1	0.258077	0	2.179366	3.801541	5.352765	0.683848	0	4.70731	2.31834	0		
4	3	2.991318	0	0	5.234942	8.467753	5.673901	1.380358	2.534644	7.663965	2.468694		
5	1	0.297309	0	1.948201	2.409205	3.845151	0.449998	0	3.733992	1.97775	0		
6	3	3.771743	0	0	5.477636	7.557581	4.045252	1.595868	3.910891	6.631868	1.735779		
7	3	4.099842	0	0	2.242885	10.19904	5.694582	0	0.012272	5.175723	0		
8	1	0.348747	0	2.453466	2.766645	4.281275	0.50688	0	4.019807	1.827492	0		
9	9	2.624012	0	1.685075	4.441686	3.682135	2.792336	1.731102	2.328723	3.906555	2.310038		
10	9	1.423208	0	1.57865	4.01103	2.637979	2.237055	1.790494	2.193746	3.010198	2.726072		
11	1	0.506986	0	3.046754	3.535328	5.222654	0.961504	0	5.177904	3.293284	1.104235		
12	1	0.556517	0	0	4.236407	4.336569	0.708161	0.250449	4.544992	3.553765	0.663194		

FIGURE 13 – Résultat de la sauvegarde des signatures d'apprentissage pour un modèle à 3 couches cachées

L'extraction et la sauvegarde des signatures de test se font au moyen de la fonction `generate_test_signatures`. Son fonctionnement consiste d'abord à prédire les données de test pour obtenir les classes prédictes. Ensuite, on utilise la fonction `get_result_layers` pour obtenir les valeurs de retour des fonctions d'activation. On enregistre ces informations dans un dossier distinct pour chaque classe de données de test.

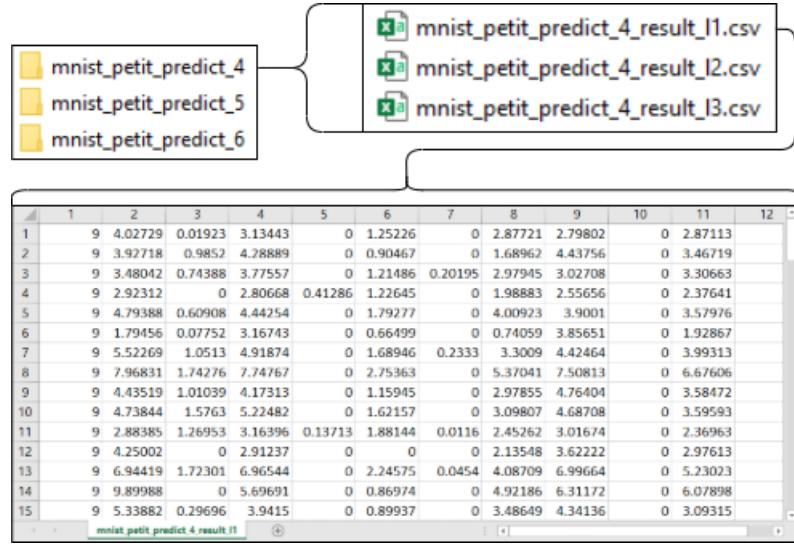


FIGURE 14 – Résultat de la sauvegarde des signatures de test

On note que la première colonne représente la classe prédictée, et le reste des colonnes sont les valeurs de retour des fonctions d'activation de la couche correspondante.

5.2.4 Discrétisation des signatures

La discrétisation des signatures est une étape intermédiaire à la clusterisation. Elle est considérée comme facultative, car elle n'a pas encore donné de résultats probants lors de nos expérimentations par rapport à la clusterisation sans discrétisation. Elle consiste à classer les valeurs de retour des fonctions d'activation dans des intervalles. On l'implémente dans la fonction `discretise_dataset`.

La fonction `discretise_dataset` utilise la fonction `cut` de Pandas qui permet de grouper des données dans des classes correspondant à des intervalles discrets (binning). Cette fonction prend, parmi ses arguments, le nombre d'intervalles par lequel on souhaite segmenter ces classes (`nb_bins`).

```
nb_bins = 5
disc_X,
ret_bins = pd.cut(X, nb_bins, labels=np.arange(nb_bins), retbins=True)
```

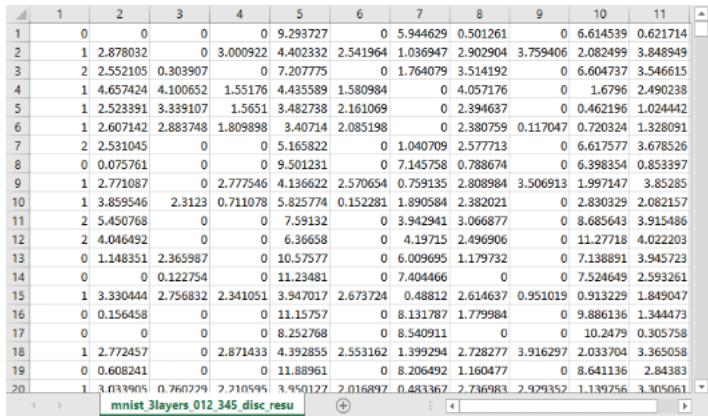
L'argument `retbins` permet de récupérer les intervalles de discrétisation. Ceux-ci seront utiles lorsqu'il faudra discrétiser les signatures de test avec les mêmes intervalles que la discrétisation des signatures d'apprentissage.

Données	Données discrétisées	Intervalles de discrétisation	Classes
1.51	3	[0, 0.5]	0
0.84	1]0.5, 1]	1
2.32	3]1, 1.5]	2
2.81	4]1.5, 2]	3
0.06	0]2, 2.5]	4
1.13	2		
0.98	1		
0.41	0		
1.57	3		

FIGURE 15 – Exemple de discrétisation avec les intervalles

La discréétisation des signatures d'apprentissage se fait au moyen de la fonction `discretize_training_signatures` qui implémente la fonction `discretise_dataset`. Elle prend en argument le nom du dossier dans lequel se trouvent les signatures d'apprentissage ainsi qu'un nombre qui représente le nombre d'intervalles par lequel on souhaite discréétiser. La discréétisation se fait sur chacun des fichiers qui correspondent aux valeurs des fonctions d'activation des données d'apprentissage pour chaque couche.

```
discretized_training_signatures ,  
nbs_bins = discretize_training_signatures (512 , directory_name)
```



A screenshot of a Jupyter Notebook cell containing two data tables. The top table, titled "mnist_3layers_012_345_disc_resu", displays numerical values for 20 data points across 12 columns (labeled 1 to 12). The bottom table, titled "mnist_3layers_012_345_l1_disc", shows the same 20 data points but with their values converted into integer counts for bins 1 through 19. A large downward arrow points from the top table to the bottom table.

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	9.293727	0	5.944629	0.501261	0	6.614539	0.621714
2	1	2.878032	0	3.000922	4.402332	2.541964	1.036947	2.902904	3.759406	2.082499	3.848949
3	2	2.552105	0.303907	0	7.207775	0	1.764079	3.514192	0	6.604737	3.546615
4	1	4.657424	4.100652	1.55176	4.435589	1.580984	0	4.057176	0	1.6796	2.490238
5	1	2.523391	3.399107	1.5651	3.482738	2.161069	0	2.394637	0	0.462196	1.024442
6	1	2.607142	2.888748	1.809894	3.40714	2.085198	0	2.380759	0.117047	0.720324	1.328091
7	2	2.531045	0	0	5.165822	0	1.040709	2.577713	0	6.617577	3.678526
8	0	0.075761	0	0	9.501231	0	7.145758	0.788674	0	6.398354	0.853397
9	1	2.771087	0	2.777546	4.136622	2.570654	0.759135	2.808984	3.506913	1.997147	3.85285
10	1	3.859546	2.3123	0.711078	5.825774	0.152281	1.890584	2.382021	0	2.830329	2.082157
11	2	5.450768	0	0	7.59132	0	3.942941	3.066877	0	8.685643	3.915486
12	2	4.046492	0	0	6.36658	0	4.19715	2.496906	0	11.27718	4.022203
13	0	1.148351	2.365987	0	10.57577	0	6.009695	1.179732	0	7.138891	3.945723
14	0	0	0.122754	0	11.23481	0	7.404466	0	0	7.524649	2.593261
15	1	3.330444	2.756832	2.341051	3.947017	2.673724	0.48812	2.614637	0.951019	0.913229	1.849047
16	0	0.156458	0	0	11.15757	0	8.131787	1.779984	0	9.886136	1.344473
17	0	0	0	0	8.252768	0	8.540911	0	0	10.2479	0.305758
18	1	2.772457	0	2.871433	4.392855	2.553162	1.399294	2.728277	3.916297	2.033704	3.365058
19	0	0.608241	0	0	11.88961	0	8.206492	1.160477	0	8.641136	2.84383
20	1	3.037905	0.760729	2.710595	3.950127	2.016897	0.483367	2.736983	2.929352	1.139756	3.305061

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	288	0	184	15	0	205	19
2	1	89	0	93	136	78	32	90	116	64	119
3	2	79	9	0	223	0	54	109	0	205	110
4	1	144	127	48	137	49	0	126	0	52	77
5	1	78	103	48	108	67	0	74	0	14	31
6	1	80	89	56	105	64	0	73	3	22	41
7	2	78	0	0	160	0	32	80	0	205	114
8	0	2	0	0	295	0	221	24	0	198	26
9	1	86	0	86	128	79	23	87	108	62	119
10	1	119	71	22	180	4	58	73	0	87	64
11	2	169	0	0	235	0	122	95	0	269	121
12	2	125	0	0	197	0	130	77	0	350	124
13	0	35	73	0	328	0	186	36	0	221	122
14	0	0	3	0	348	0	229	0	0	233	80
15	1	103	85	72	122	83	15	81	29	28	57
16	0	4	0	0	346	0	252	55	0	307	41
17	0	0	0	0	256	0	265	0	0	318	9
18	1	86	0	89	136	79	43	84	121	63	104
19	0	18	0	0	369	0	254	36	0	268	88
20	1	44	21	68	122	62	15	84	90	35	102

FIGURE 16 – Exemple de discréétisation des signatures d'apprentissage

La fonction renvoie les signatures discrétisées ainsi qu'un tableau qui contient les intervalles de discrétisation (nbs_bins). Ces intervalles de discrétisation vont servir lors de la discrétisation des données de test. En effet, il est nécessaire de discrétiser de la même manière les données d'apprentissage et les données de test pour garder le rapport qu'il existe entre ces différentes données et pour avoir une clusterisation qui a du sens.

La discrétisation des signatures de test se fait au moyen de la fonction discretize_test_signatures qui implémente la fonction discretise_dataset. Elle prend en argument le même nom de dossier que la précédente fonction, les valeurs de test (cet argument est uniquement nécessaire pour la sauvegarde des données) ainsi que les intervalles de discrétisation des données de test (nbs_bins).

```
discretized_test_signatures =  
    discretize_test_signatures(directory_name, test_values, nbs_bins)
```

Les signatures de test discrétisées sont sauvegardées dans le même dossier que les signatures de test.

5.2.5 Clusterisation des signatures

Il existe de nombreuses méthodes de clusterisation (e.g. Hierarchique, par densité (dbscan), etc). Dans le cadre de notre application, nous avons choisi Kmeans qui est fourni par la librairie sklearn. Toutefois, il est très simple de changer ce clustering. Kmeans nous a été conseillée par notre encadrant pour sa simplicité d'utilisation. Cette méthode de clusterisation permet notamment de choisir le nombre de clusters que l'on souhaite obtenir. Pour clusteriser des données, on crée un modèle Kmeans puis on exécute la fonction fit sur ces données.

```
kmeans = KMeans(n_clusters=3, random_state=0).fit(signatures)
```

La prochaine figure 17 montre le résultatat de la clusterisation par Kmeans des signatures d'apprentissage du jeu de données MNIST dans un modèle possédant 3 couches cachées à 50 neurones chacune. Ce modèle a appris à reconnaître les classes 0, 1 et 2. On a demandé à Kmeans de trouver trois clusters pour voir si on voyait bien 3 zones délimitées correspondant aux prédictions des 3 classes. Pour afficher les signatures (possédant ici 50 dimensions) sur un plan à deux dimensions, on utilise un algorithme de réduction de la dimensionnalité appellée PCA.

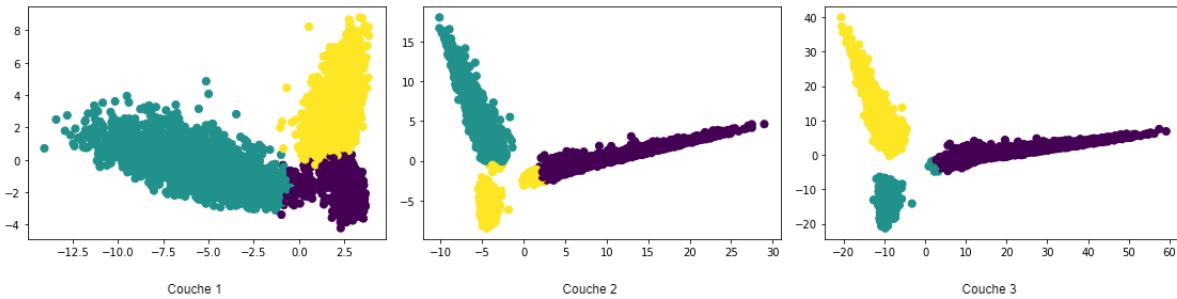


FIGURE 17 – Exemple de clusterisation des signatures d'apprentissage

On observe qu'il y a peu de différences entre la clusterisation de la couche 2 et celle de la couche 3. On peut déjà conjecturer qu'on a utilisé trop de couches.

Pour la clusterisation des signatures d'apprentissage, on utilise la fonction `clusterize_training_signatures` qui implémente Kmeans et qui prend en entrée les signatures d'apprentissage qu'elles soient discrétisées ou non.

```
clusterized_train_signatures ,  
layers_kmeans = clusterize_training_signatures( train_signatures , train_values )
```

La fonction renvoie les signatures clusterisées, c'est-à-dire un tableau contenant les classes de cluster pour chaque signature et pour chaque couche. La fonction renvoie un autre tableau qui contient les instances de Kmeans utilisées pour la clusterisation de chaque layers, celles-ci seront utiles lors de la clusterisation des signatures de test. En effet, il est nécessaire de clusteriser les signatures de test à partir de la clusterisation des signatures d'apprentissage et pour chaque couche.

Valeurs de retour des fonctions d'activation	Clusterisation
Couche 1 [0, 0.125, 0.0654, ..., 0, 0.512]	0
Couche 2 [0, 0, 0.214, ..., 0.112, 0.91, 0]	2
Couche 3 [0.15, 0, 0.421, ..., 0.012, 0.51]	4

FIGURE 18 – Résultat de la clusterisation d'une signature

La clusterisation de signatures de test se fait au moyen de la fonction `clusterize_test_signatures` qui prend en entrée les signatures de test ainsi que le tableau des instances de Kmeans que l'on a généré précédemment. On utilise la fonction `predict` de Kmeans sur les signatures de test pour obtenir leurs classes de cluster pour chaque couche. Toutes ces données sont enfin sauvegardées dans un fichier unique qui contient l'ensemble des signatures clusterisées des données de test.

```
clusterize_test_signatures( test_signatures ,  
                            layers_kmeans ,  
                            test_values ,  
                            project_name )
```

	1	2	3	4	5	
1	input_class	cluster_in_layer1	cluster_in_layer2	cluster_in_layer3	output_class	
2	5	4	0	1	0	
3	5	2	2	0	1	1
4	5	2	0	1	0	
5	5	2	0	1	0	
6	5	4	0	1	0	
7	5	2	2	0	0	
8	5	2	2	0	1	
9	5	3	3	2	2	
10	5	2	0	0	1	
11	5	2	2	0	1	
12	5	3	3	2	2	
13	5	2	0	1	0	
14	5	2	2	0	2	
15	5	2	2	0	1	
16	5	0	0	0	1	
17	5	4	0	1	0	
18	5	2	2	0	2	
19	5	2	2	0	1	
20	5	0	4	2	2	

FIGURE 19 – Résultat de la clusterisation des signatures de test

On a maintenant une véritable liste de signatures des prédictions de notre réseau de neurones. On cherche maintenant à les afficher d'une certaine façon de sorte à pouvoir les analyser.

5.2.6 Affichage des résultats

Pour afficher le résultat de la clusterisation des signatures de test, nous avons développé une interface de visualisation en HTML/CSS/Javascript construite sur NodeJS. On implémente le framework Bootstrap qui facilite grandement le développement de tout l'aspect front-end de notre solution (boutons, tableaux, etc ...).

On utilise aussi le framework D3.js qui offre un large choix de visualisations de données sous une forme graphique et dynamique. La visualisation que nous avons choisi d'utiliser est le diagramme Sankey. Ce type de graphique permet de représenter des flux.

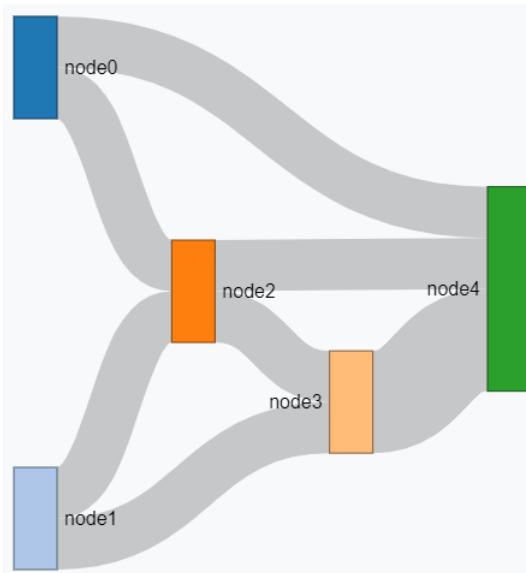


FIGURE 20 – Exemple d'un diagramme Sankey

Notre librairie Python possède une fonction `generate_interface_data` qui permet de générer toutes les données dont nous aurons besoin pour visualiser nos résultats à l'aide de notre interface. Ainsi, un dossier `[nom_du_projet]_DATA` est généré et on copie celui-ci dans le dossier `data` de notre interface. Ce dossier contient 3 fichiers :

- `clusterized_values.csv` qui possède la liste des signatures clusterisées
- `clusters_info.json` qui possède des informations sur les clusters (nombre d'occurrences des classes d'apprentissage, nombre de clusters par couche, etc ...)
- `network_info.json` qui possède des informations sur le modèle (nombre de couches cachées, types de fonction d'activation, nombre de classes de sortie, nombre de features, etc ...)

Le serveur NodeJS commence par charger le fichier clusterized_values.csv et opère directement un GROUP BY sur les signatures. En effet, une grande partie des signatures sont identiques et il semble beaucoup plus intéressant de compter leurs occurrences au lieu de les garder telles quelles. NodeJS charge ensuite les fichiers restants. Les informations contenues dans ces fichiers permettent la construction de la page HTML de l'interface.

La page HTML est construite de la façon suivante :

- sur la gauche on a le tableau des signatures avec leurs occurrences, il est possible de les trier selon n'importe laquelle des colonnes (input_class, signature, output_class, occurrences)
- à côté, on a la visualisation des clusters et des couches d'entrée et de sortie. Les différentes classes d'entrée et de sortie sont représentées par des noeuds tout comme les clusters de chaque couche. Les signatures sont représentées par des liens entre ces noeuds.

Avec la vue d'ensemble, on peut représenter l'ensemble des signatures. L'épaisseur d'un lien est proportionnelle au nombre d'occurrences d'une signature et l'épaisseur d'un noeud est proportionnelle au nombre de signatures qui passe par ce noeud.

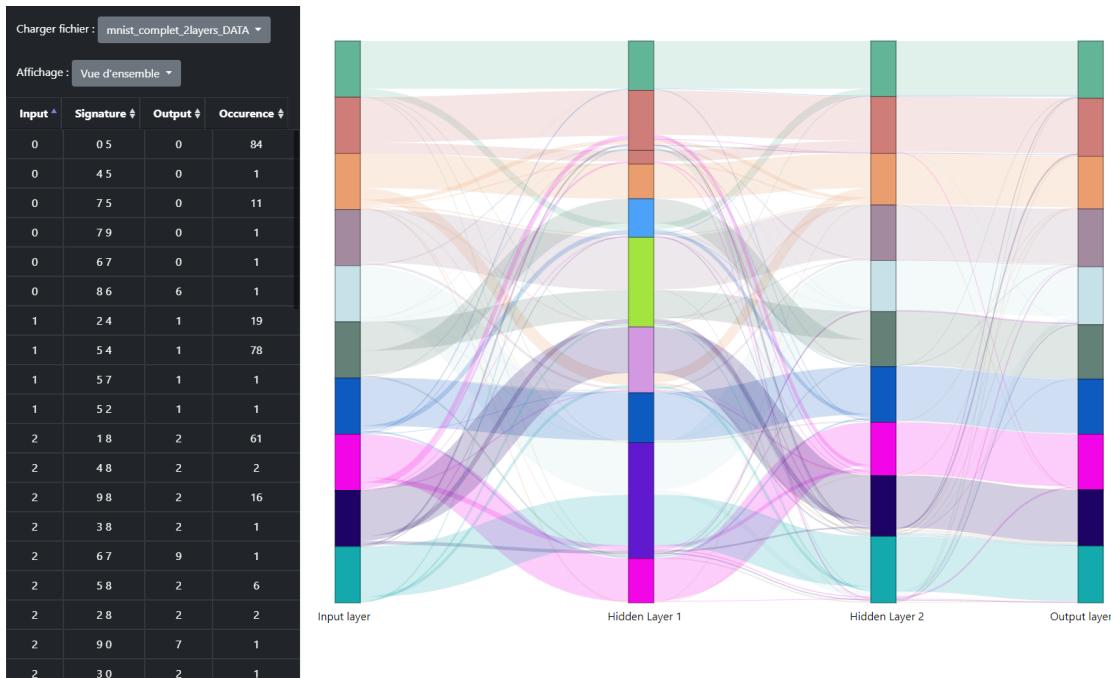


FIGURE 21 – Capture d'écran de l'interface avec l'affichage "vue d'ensemble"

En cliquant sur une ligne du tableau à gauche, on peut visualiser chaque signature séparément.

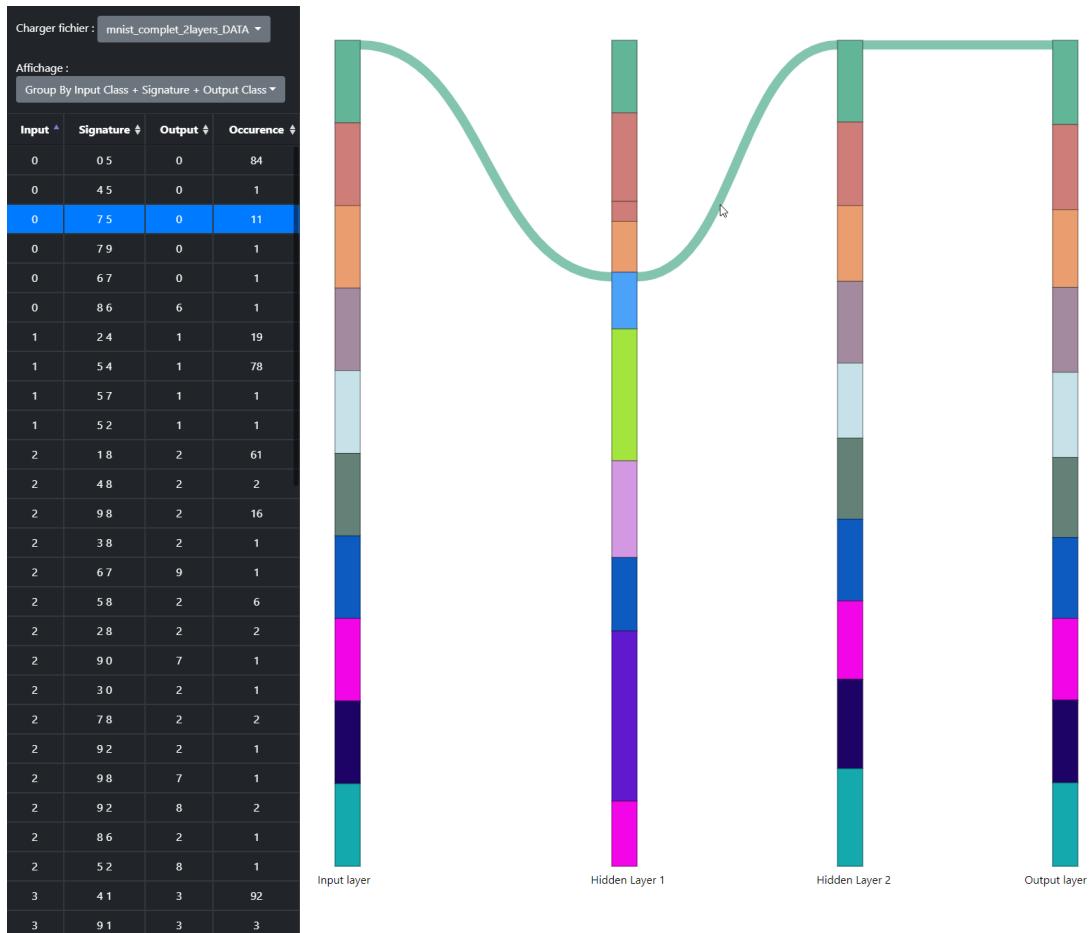


FIGURE 22 – Capture d'écran de l'interface avec l'affichage "signature distincte"

De plus, un bouton "Informations sur le modèle" permet d'obtenir une bulle, figure 23, fournissant une explication sur l'affichage. Nous pouvons obtenir le nombre de neurones par couches, le nombre de clusters ainsi que les classes d'apprentissage et de tests.

Ce modèle est un réseau de neurones à
2 couches cachées possédant les
caractéristiques suivantes:

- 784 features dans la couche d'entrée
- 512 neurones dans la couche cachée n°1
- 512 neurones dans la couche cachée n°2
- 2 neurones dans la couche de sortie

Le modèle a été entraîné sur les classes :
[0,1]

Le modèle a été testé sur les classes : [2]

On a choisi 5 clusters par couche cachée

FIGURE 23 – Capture d'écran de la bulle "information"

5.3 Analyse des résultats

Nous avons créé un outil qui montre les traces des fonctions d'activation. À l'aide de notre interface, nous pouvons déterminer comment se répartit une classe inconnue pour notre modèle. Par exemple, si nous l'entraînons sur des 0 et 1, nous pouvons observer le comportement de notre modèle en lui demandant de prédire des 2. Le résultat est visible sur la figure 24.

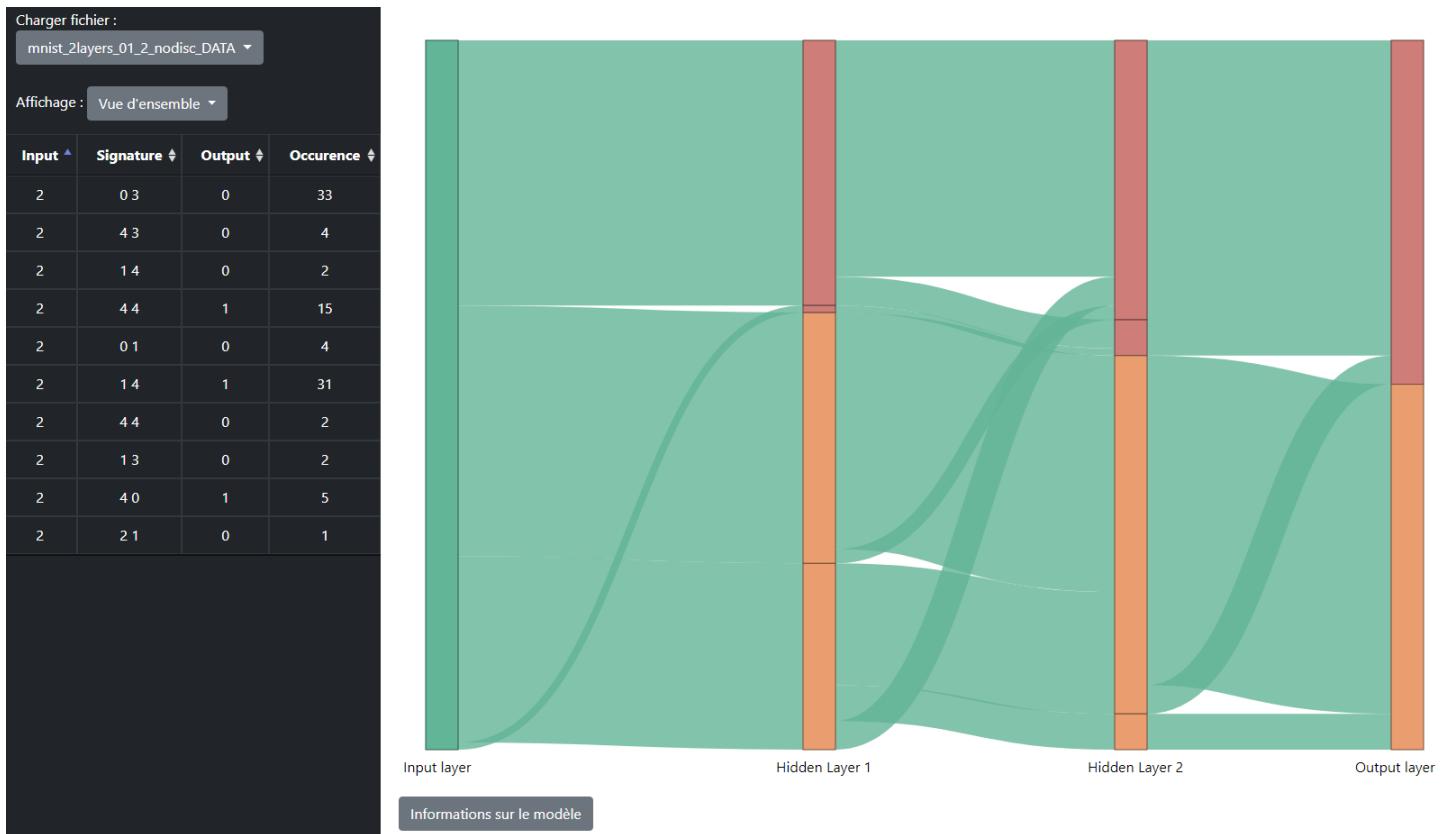


FIGURE 24 – Capture d'écran de l'interface sur train(0,1) et test(2)

L'interface nous permet également de déterminer si notre modèle est composé de trop de couches. En effet, dans certains cas, le résultat final est obtenu dès les premières couches et on observe une symétrie dans le comportement des signatures des couches supérieures.

Dans la figure 25, on a visualisé un modèle avec 3 couches cachées. On observe une symétrie du comportement des signatures entre la couche 2 et la couche 3 et entre la couche 3 et la couche de sortie. Cela signifie que le modèle a déjà pris sa décision à partir de la couche 2 et donc que la couche 3 est inutile.

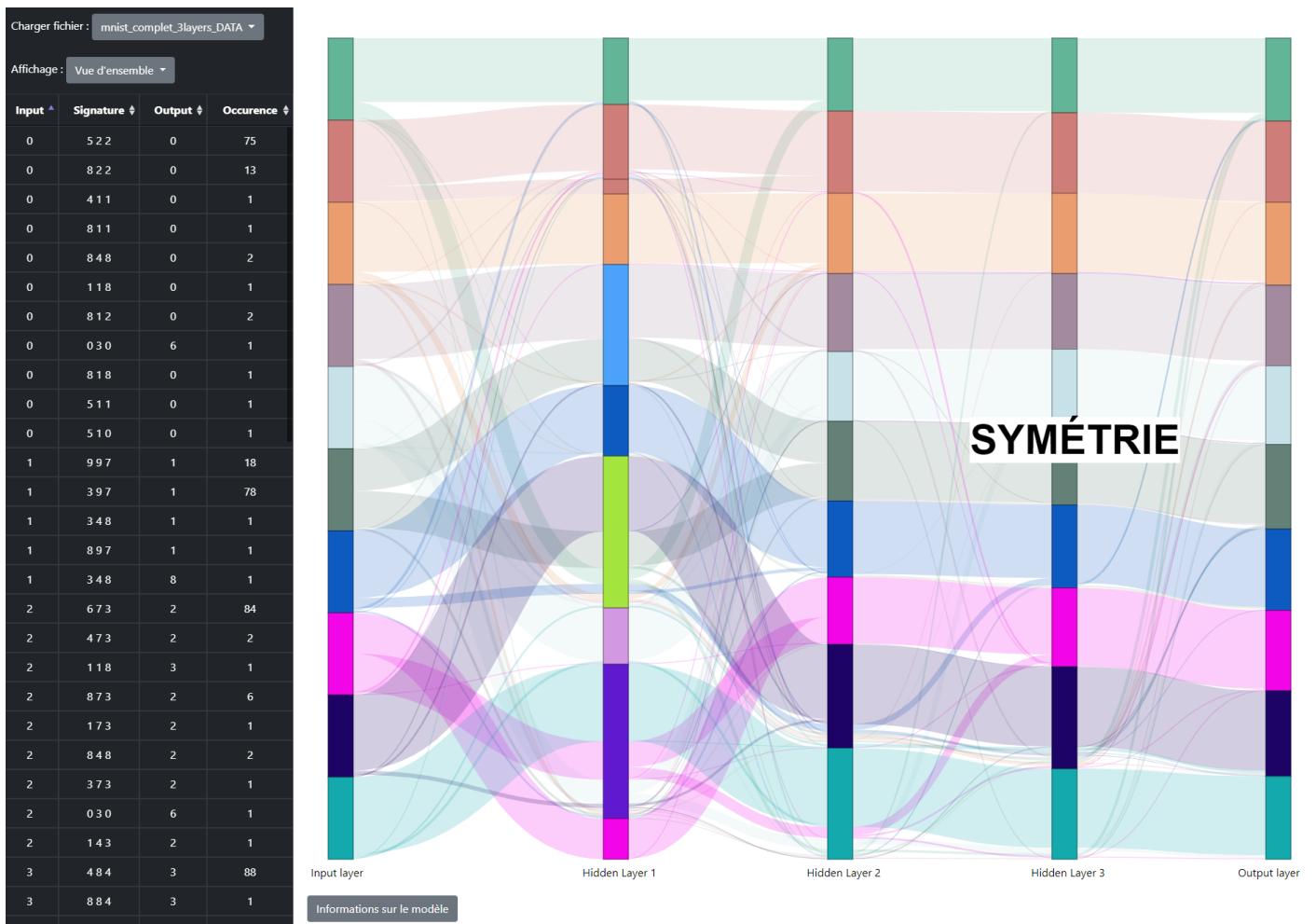


FIGURE 25 – Capture d'écran de l'interface montrant le surplus de couches

6 Rapport d'activité

6.1 Organisation du travail

On peut opposer les méthodes de travail dans ce TER avec celle du cadre industriel. En effet, ce projet de TER s'est effectué dans un cadre d'un travail d'enseignement. Au début du développement, notre idée des attentes de notre tuteur était encore floue. Il nous a fallu expérimenter et comprendre ce qu'il nous était demandé. Notre tuteur ne nous a pas donné de cahier des charges précis. La demande de notre encadrant a varié en fonction de nos avancés dans la compréhension du sujet ainsi que de nos réalisations, c'est pourquoi la première réalisation du Gantt a été faite de façon utopique en début de projet, pour finalement être réajusté tout au long de ce semestre.

Méthodes agiles

Le projet a été réalisé avec la méthode Scrum, cette méthode allie valeurs et principes agiles. Un Scrum Master a été choisi pour qu'il se porte garant du processus Scrum.

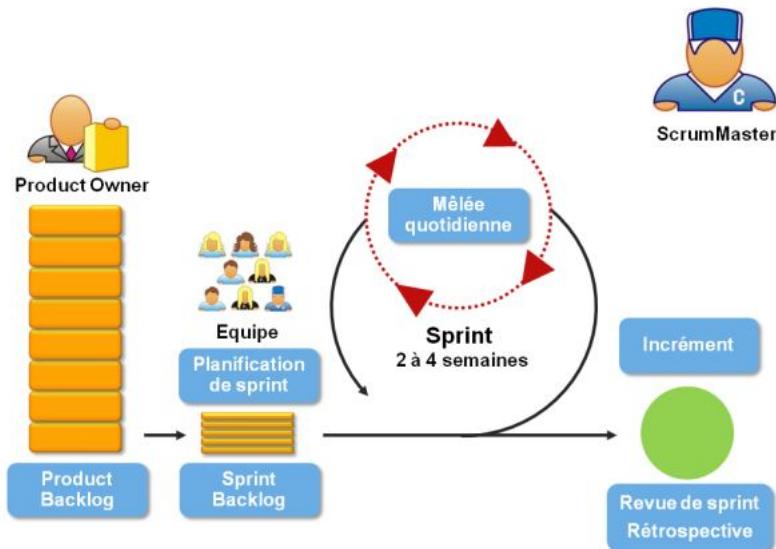


FIGURE 26 – Fonctionnement Méthode Scrum

Les valeurs agiles

Nous avons instauré dès le début du projet une bonne communication au sein du groupe afin de pouvoir répondre aux besoins de notre tuteur. Au fur et à mesure des projets réalisés par notre groupe de travail, nous nous sommes rendu compte que la clé pour réussir un projet était de bien communiquer entre les membres de l'équipe, mais aussi avec le tuteur.

Les méthodes agiles favorisent le rendement de logiciels opérationnels plutôt que de la documentation exhaustive. En effet, le principe même vient de l'utilisabilité des fonctionnalités. Pour le tuteur, le plus important est d'avoir une librairie utile et opérationnelle, plutôt que d'avoir du code dont la plupart des fonctionnalités ne seront jamais utilisées. Au cours du projet, nous avons essayé de planifier la livraison de fonctionnalités que notre tuteur nous avait demandée. Les premières itérations furent compliquées dues au manque de connaissance sur ce sujet. Il nous aura fallu quelques itérations pour commencer à comprendre ce que notre tuteur voulait nous faire découvrir. Au final, nous pouvons retenir que les premières livraisons ont été un peu légère, mais essentiel pour le bon déroulement de la suite du projet. Les livraisons suivantes ont été beaucoup plus concluantes pour le rendu final.

Les valeurs agiles valorisent aussi la collaboration avec notre tuteur plus que la négociation contractuelle. C'est-à-dire qu'il vaut mieux discuter et échanger avec le responsable du projet des idées plutôt que de tout fixer avec un cahier des charges à respecter. Cela permet de satisfaire au mieux le tuteur et lui permet également de pouvoir soumettre de nouvelles idées à tout instant. La collaboration permet aussi un meilleur rendement de la part de l'équipe développement. Ainsi la collaboration et la négociation avec le tuteur peuvent être couplées avec la communication. Durant le projet, il ne nous a pas fallu faire des choix sur le développement des fonctionnalités, car notre encadrant était là pour nous guider, de ce fait aucune négociation de fonctionnalité à eux lieu.

La gestion du projet nous a demandé de nous adapter aux changements. En effet, en cours d'itérations le product-owner pouvait recevoir de nouvelle demandent de fonctionnalités pour le sprint en cours. Il a fallu que nous nous adaptions pour pouvoir répondre à ses besoins. Nous avons dû faire des choix pour savoir si oui ou non nous allions faire les nouvelles fonctionnalités au cours d'une itération déjà planifiée et commencée. Il fallait aussi prendre en compte le rythme de développement et de compréhension pour ne pas être débordé et pouvoir rendre des fonctionnalités de qualités. En ayant effectué le projet avec les méthodes agiles, cela nous a permis, grâce à la planification itérative et adaptive de pouvoir prendre en compte la demande du tuteur.

Notre projet a été développé avec certaines valeurs agiles que nous nous sommes efforcés à respecter. Nous avons mis en place le principe de l'amélioration continue en effectuant des « mélées » quotidiennes (autrement dit Daily Meeting), et en faisant des rétrospectives après chaque itération afin d'identifier les problèmes rencontrés et de pouvoir les résoudre, pour pouvoir nous améliorer sur l'itération suivante. Le principe de l'amélioration continue a eu un effet bénéfique sur la planification des sprints ainsi que sur la répartition du travail au sein du groupe. Nous avons pu ainsi devenir plus efficaces au fil des itérations et améliorer notre rendement.

Nous pouvons donc en retenir qu'il est primordial dans un projet agile de favoriser la cohésion de groupe, de livrer des fonctionnalités opérationnelles, de communiquer et de collaborer avec le client, et enfin de pouvoir s'adapter au changement.

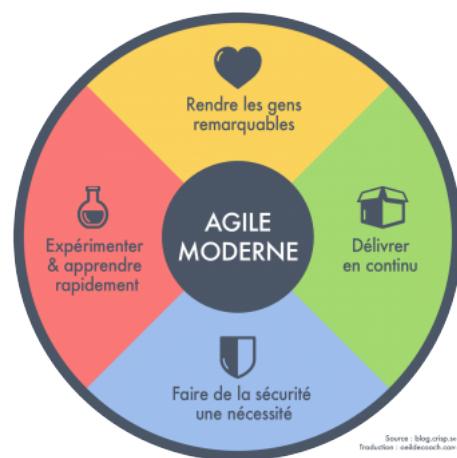


FIGURE 27 – Valeurs Agile

Les principes agiles

Les principes agiles reposent sur l'utilisation de la méthode Scrum. En effet, le projet a été guidé par les méthodes agiles de gestion de projet. Ainsi, notre projet s'est déroulé de façon itérative. Il a été mis en place un ScrumMaster, qui devait être garant du processus Scrum ainsi qu'un Product Owner, qui était le représentant du produit.

Le projet s'est déroulé en sept itérations (ou Sprints) de quinze jours environ. Chaque itération se déroulant sur deux semaines, il fallait au début de chaque itération planifier le sprint en y intégrant les remarques du tuteur sur la dernière

itération et également ses nouvelles exigences. Grâce à la planification adaptative et itérative, cela nous a permis de prendre en compte les remarques et les exigences du commanditaire. La durée des itérations nous a permis de développer des fonctionnalités qui avaient de la valeur pour le tuteur à chaque livraison.

Pour planifier les itérations, nous avons utilisé un outil permettant la gestion et la répartition des tâches, Trello. À chaque itération, il aurait fallu revaloriser et estimer les stories. Pour cela, l'utilisation d'un planning poker aurait été envisagé. Le planning poker, permet de donner des points d'effort et des points de valeurs pour pouvoir prioriser les tâches.

Après chaque itération, une rétrospective générale de fin de sprint était mise en place. Durant cette session, nous utilisions des post-it pour mettre en évidence ce qui avait bien marché, ce qui avait moins bien marché et les possibles améliorations. Le scrum master se chargeait de faire le point sur la situation. Ainsi, nous pouvions régler les problèmes dans un premier temps, puis nous féliciter du travail accompli pour permettre de rester sur un point positif afin de ne pas nous décourager. Enfin, il fallait penser aux possibles améliorations pour respecter le principe de l'amélioration continue. En effet, l'amélioration continue est une des clés de la réussite d'un projet.

La méthode Scrum permet un feed-back régulier et rapide entre l'équipe et le commanditaire. Ainsi ceci permet d'éviter l'effet tunnel et de pouvoir s'adapter aux changements et aux contraintes. Durant la totalité du projet, nous avons réussi à mettre en place une communication entre l'équipe et le commanditaire. Cela nous a permis de travailler sans stress ni préjudice, car nous savions exactement ce que notre tuteur attendait de nous, nous n'avons pas eux de fonctionnalités refusées donc aucun travail à rattrapé sur le sprint suivant.

Entre membres de l'équipe, il faut également communiquer et favoriser les dialogues en face à face. Nous avons eu quelques difficultés au début du projet pour assimiler ce nouveau domaine d'étude, mais grâce à notre cohésion, nous sommes parvenus à nous entre aidé pour développé les fonctionnalités demandées.

Nous pouvons donc en retenir que respecter les principes agiles est important pour le fonctionnement d'un projet agile. Il faut donc bien communiquer avec le tuteur pour bien connaître son projet, lui livrer de la valeur tout au long du projet et rapidement pour corriger les éventuelles erreurs. De plus, il faut faire preuve d'auto organisation pour gérer les problèmes (confinement) et les résoudre. Enfin, il faut pouvoir s'adapter aux changements et les accueillir positivement.

6.1.1 Diagramme de Gantt

Bien que le GANTT, figure 28, ne fasse pas partie des méthodes agiles, nous avons trouvé judicieux d'en créer un afin de mettre en avant et de manière plus lisible, les grandes étapes de notre projet. Cela nous a donc permis de donner les objectifs pour chaque sprint. Nous avons considéré que chaque réunion avec notre tuteur était l'accomplissement d'un sprint, permettant ainsi de réaliser des sprints tout au long du projet, d'une durée allant d'une à deux semaines, comme nous pouvons le constater sur la capture suivante.

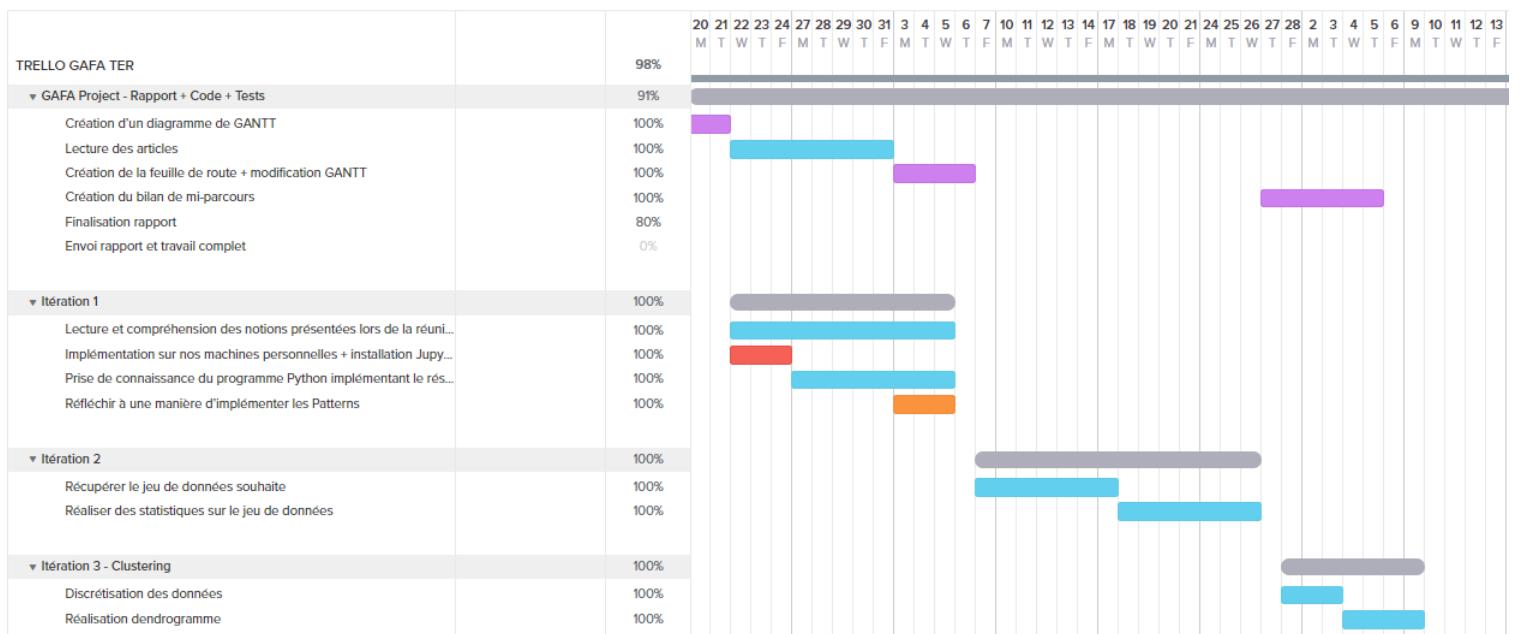


FIGURE 28 – Diagramme de Gantt

Au fil des semaines, notre travail s'est découpé en plusieurs itérations qui sont visibles à l'aide du Gantt. Nous nous sommes fixé des objectifs lors de ces itérations et si certaines fonctionnalités n'ont pas été implémentées, nous discutions lors des réunions de quelles solutions nous comptions appliquer. Ces différents sprints nous ont permis de réaliser une méthode dite "agile" qui a été entièrement détaillée dans la partie précédente.

6.1.2 Rôles dans l'équipe

Dès le début de notre projet, nous avons appliqué les méthodes agile. Notre tuteur nous a demandé de respecter des sprints de deux ou trois semaines étant donné les congés scolaires. Nous nous sommes donc répartis les rôles suivants, Product Owner, Scrum Master et développeur.

Attributions de rôles :

Adrien DIDER, Développeur ;
Loïc MARTIN, Scrum Master ;
Pierre PORTAL, Développeur ;
Aurélien TROUCHE, Product Owner.

Nous avons fait le choix de travailler ensemble sur toutes les parties, car le sujet des réseaux de neurones était nouveau pour nous tous et chacun avait besoin de comprendre les différentes parties du code qui pouvait parfois s'avérer très abstrait. De plus, le Product Owner devait s'occuper de la relation tuteur-équipe. Le Scrum Master quant à lui s'occupait de la gestion de l'équipe. Il devait faire en sorte que son équipe se sente le plus impliquée possible dans le projet. Il avait pour tâche de planifier les différentes réunions, la création de la feuille de réunion, de maintenir à jour le Trello ainsi que de s'assurer de la bonne progression de son équipe.

6.1.3 Itérations

Comme expliqué dans la partie précédente, nous avons travaillé en respectant des sprints. Le travail à effectuer pour le sprint suivant était décidé lors des réunions en fonction de l'accomplissement du sprint précédent.

De plus, il était demandé de réaliser une fiche de rendu de réunion suite à chaque réunion réalisée :

Sprint 0 : Réalisation d'un gantt, Répartition des rôles, Découverte du sujet.

Sprint 1 : Découverte de MNIST, Réalisation fonction d'activation, Ingénierie des données (statistiques/discrimination).

Sprint 2 : Discréétisation, Levenshtein, Clustering, Algorithmes hiérarchique/-densité, Matrice des distances.

Sprint 3 : Discréétisation des données et clusterisation.

Sprint 4 : Utilisation des méthodes predict.

Sprint 5 : Obtention de la vérité terrain, affichage de la clusterisation et comparaison des résultats obtenus.

Sprint 6 : Sauvegarde des données dans les différents CSV et développement de l'application web.

Sprint 7 : Développement de l'application web et rédaction rapport.

6.2 Outils

6.2.1 Gestion des tâches - Trello

Pour avoir une vue d'ensemble du projet, nous avons utilisé Trello pour organiser les tâches. Cet outil en ligne permet de nombreuses fonctionnalités telles que :

- La création de tâches avec des objectifs à réaliser.
- L'affectation des membres sur les différentes tâches
- La mise en place de balises de couleurs pour identifier le domaine de la tâche
- L'indication de l'avancement des tâches

De plus, il est possible de créer différentes catégories pour classer l'ensemble des tâches :

- **IDEA** différentes idées à discuter ou présenter au tuteur
- **TO DO** le travail à réaliser
- **DOING** les tâches en cours de traitement par le/les membre(s) assigné(s)
- **DONE** ensemble des tâches réalisées

Sur la figure 29, nous pouvons observer la carte doing lors du début du projet. Le travail demandé était de lire des articles, mais aussi de créer le diagramme de GANTT. De plus, pour commencer le projet, il était demandé de se renseigner sur le jeu de données et de commencer la réalisation de statistiques sur ce dernier.

La réalisation et le suivi de ses différentes cartes ont été attribués au scrum master. Il discutait avec les différents membres pour déterminer qui ferait quelle tâche, et devait en cas de soucis être présent pour trouver une solution arrangeant l'équipe.

Cet outil a permis à l'équipe de se sentir bien plus impliquée dans le projet, car ces derniers pouvaient en temps réel se rendre compte du travail accompli tout en visualisant les objectifs posés. Ils avaient une vision élargie sur le projet.

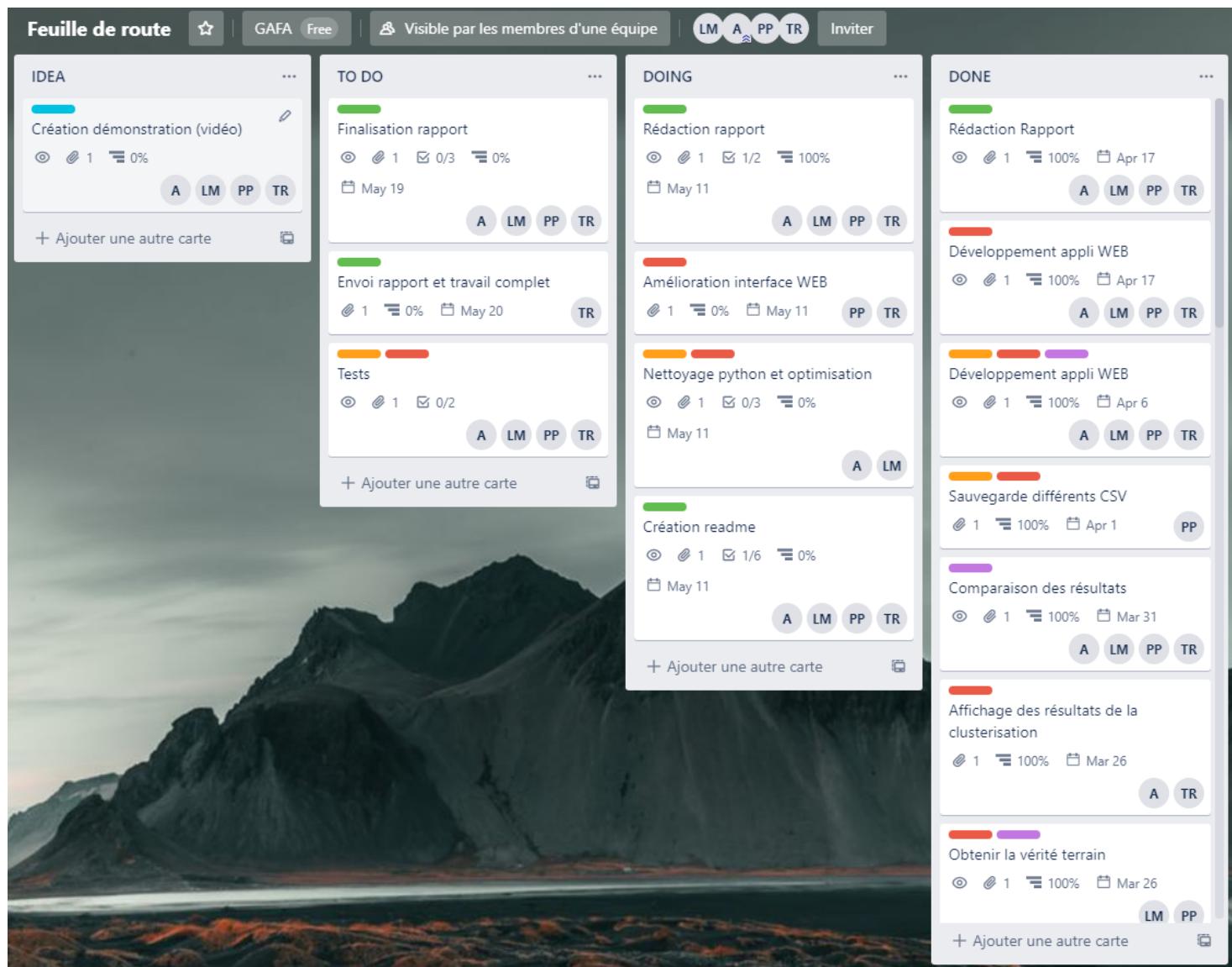


FIGURE 29 – Copié d'écran du Trello

6.2.2 Discord - TeamViewer

La majorité du travail a été effectuée en dehors des enceintes de la faculté des sciences. C'est pourquoi il était primordial d'avoir des outils nous permettant de rester en contact d'où l'utilisations un groupe de discussion Discord ainsi que TeamViewer.

Discord

Discord est un outil principalement dédié aux joueurs de jeux-vidéos permettant la création de salons textuels mais aussi vocaux. Étant déjà utilisateur de cet outil, nous n'avons pas rencontré de difficultés dans l'utilisation de dernier.

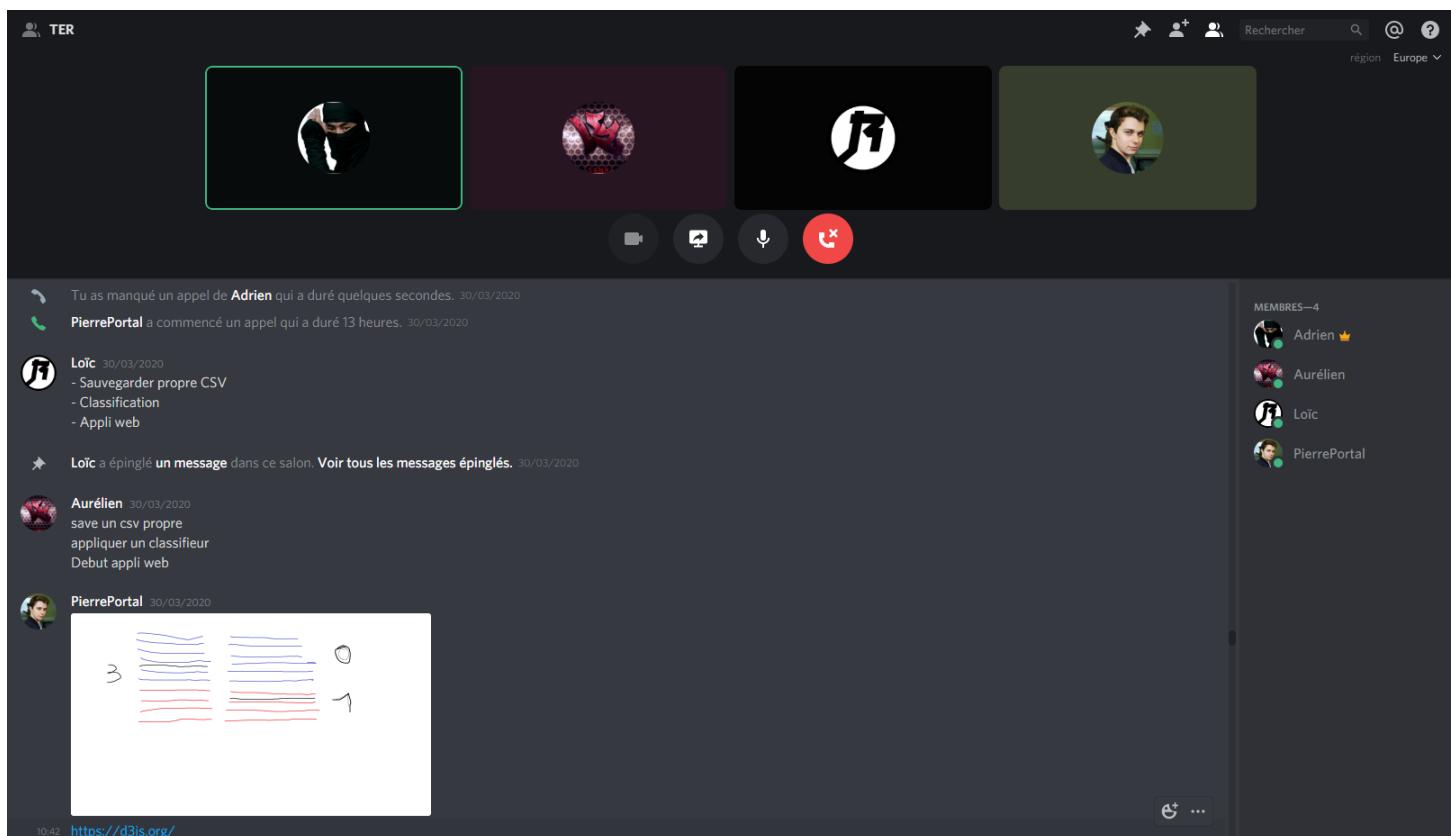


FIGURE 30 – Copie d'écran du groupe de discussion Discord

TeamViewer

Ici, nous faisons un petit aparté sur le fonctionnement de TeamViewer pour que vous puissiez bien comprendre notre façon de travailler hors de l'établissement. TeamViewer est un logiciel gratuit qui permet non seulement de partager un écran à plusieurs personnes, mais également de pouvoir interagir avec le "partageur", c'est-à-dire que la personne qui partage n'est plus la seule à avoir accès à son ordinateur. Les membres du groupe possédant la fibre ont étaient des partageurs pour que le reste du groupe puisse travailler convenablement. C'est pour cela que l'on retrouve souvent les mêmes personnes en commit sur le Github.

The screenshot shows a desktop environment with several windows open. In the foreground, the Spyder IDE is running, displaying a Python script named 'PreTreatment.py'. The script contains code for loading MNIST data, processing it, and creating a neural network model. A tooltip from the Spyder interface is visible, providing information about the 'Aide' (Help) feature. To the right of the IDE, a TeamViewer interface window is open, showing a list of connected sessions. The TeamViewer window has a watermark at the bottom right.

```
40
41 def save_result_layers(filename,X,y,result_layers):
42     f = open(filename, "w")
43     for nb_X in range (len(X)):
44         my_string=""
45         my_string+=str(nb_X)+","
46         for nb_layers in range (len(model.layers)-1):
47             my_string+=">"
48             for i in range (len(result_layers[nb_layers][nb_X])):
49                 my_string+=str(result_layers[nb_layers][nb_X][i])+",
50             my_string+=">."
51         my_string=my_string [:-1]
52         my_string+="\n"
53         f.write(my_string)
54     f.close()
55
56
57 (X_train, y_train), (X_test, y_test) = mnist.load_data()
58 X_train_sample=X_train[0:100]
59 y_train_sample=y_train[0:100]
60
61 X_train=X_train_sample
62 y_train=y_train_sample
63 X_train = X_train.reshape(100, 784)
64 X_train = X_train.astype('float32')
65
66 X_train /= 255
67
68
69
70 X_01=[]
71 X_01.append(1)
72 nb_X=0
73 for i in range(X_train.shape[0]):
74     if (y_train[i]==0 or y_train[i]==1):
75
76         nb_X+=1
77         X_01.append(X_train[i])
78         y_01.append(y_train[i])
79
80
81 train_X=np.asarray(X_01)
82
83 train_y=y_01
84
85 encoder = LabelEncoder()
86 train_y=encoder.fit_transform(train_y)
87
88
89 input_dim = 784
90
91
92 model = Sequential()
93 model.add(Dense(5, input_dim = input_dim , activation = 'relu'))
94 model.add(Dense(1, activation = 'sigmoid'))
95
96 model.compile(loss = 'binary_crossentropy', optimizer='adam' , metrics = ['accuracy'])
97
98 model.fit(train_X, train_y, epochs = 40, batch_size = 32)
```

FIGURE 31 – Partage d'écran via TeamViewer

6.2.3 Github

Github a été utilisé afin d'héberger et versionner notre code en ligne. Le système de commit nous permettait de mettre à jour le code tout en gardant une sauvegarde des anciennes versions en cas de soucis. De plus, ce dernier était un bon moyen de visualiser le code qui avait été écrit. Github permet de nombreuses fonctionnalités telles que l'historique des commits. Nous pouvons observer le nombre de commits effectués par membres, mais aussi la quantité de lignes ajoutées et supprimées. La différence du nombre de commit a été expliquée dans le paragraphe précédent.

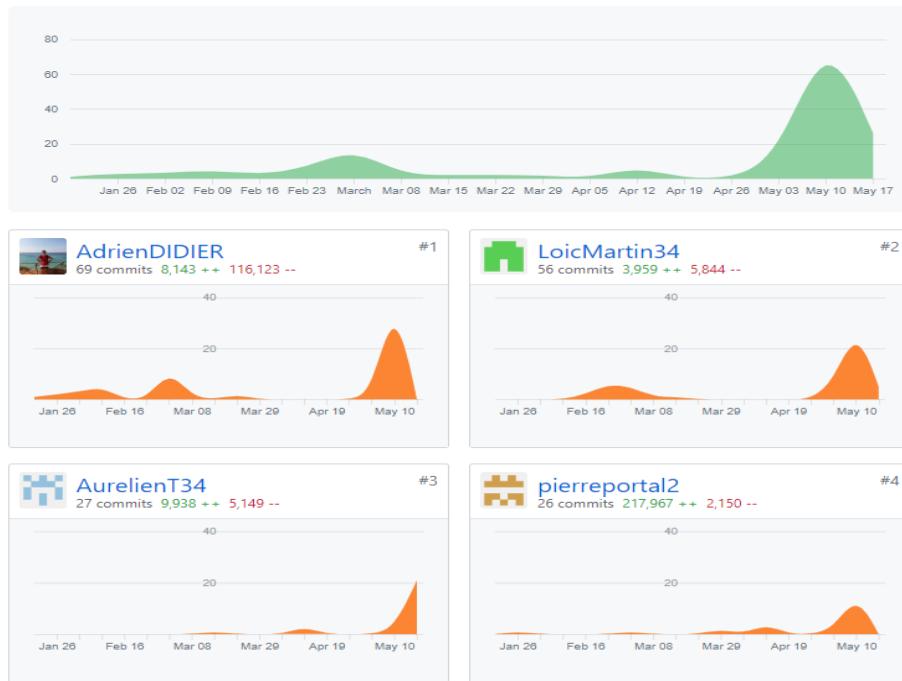


FIGURE 32 – Historique des commits

Ce dernier permet également d'y stocker notre rapport, le fichier readme qui donnera les différentes informations de ce projet telles que le manuel d'utilisation. Nous pouvons aussi y obtenir le lien de la vidéo de présentation.

6.2.4 Anaconda

Anaconda est une distribution libre et open source des langages de programmation Python et R appliquée au développement d'applications dédiées à la science des données et à l'apprentissage automatique, qui vise à simplifier la gestion des paquets et de déploiement.



FIGURE 33 – Logo Anaconda

Pourquoi utiliser Anaconda ? Nous avons choisi d'utiliser Anaconda, car il permet d'installer tout ce qui nous était nécessaire, Jupyter pour les notebooks de cours fournit par notre tuteur, Spyder pour pouvoir coder en python. La fonctionnalité la plus importante pour nous était l'installation de paquets. Anaconda possède une interface de recherche qui permet de savoir quels sont les paquets installés et pouvoir faire une recherche pour installer des paquets manquants.

6.3 COVID-19

Avant le COVID-19, le Product Owner utilisait les mails pour communiquer avec notre tuteur et, durant le confinement, un salon de discussion a été créé avec l'ensemble des projets de Monsieur Poncelet. Ce salon nous met à disposition un canal écrit, mais aussi vocal afin de continuer les réunions de sprint. Nous pouvions également partager notre écran, pour montrer l'avancement du projet.

Nos méthodes de travail ont drastiquement changé dû au confinement. En effet, avant le confinement, le groupe se réunissait sur TeamViewer tous les

vendredis et les lundis pour travailler le projet, car ces deux jours étaient libres sur notre emploi du temps. Durant le confinement, notre charge de travail à la maison a augmenté, nous avons dû réaliser beaucoup plus de travaux pratiques et de projets étant donné que l'année courante est évaluée par le contrôle continu.

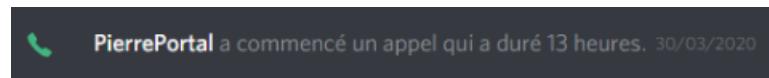


FIGURE 34 – Charge de Travail

Heureusement, nous avions quand même bien avancé sur le projet ce qui nous a permis de terminer le projet avec une entrevue par semaine du groupe sur TeamViewer.

7 Conclusion : synthèse et bilan

L'objectif de ce TER était de comprendre le fonctionnement d'un réseau de neurones et plus particulièrement celui de la boîte noire. Après de nombreuses heures de travail, nous sommes fiers du résultat que nous avons réussi à produire.

7.1 Résultats obtenus

Cette approche met en évidence un certain nombre de choses confère partie analyse des résultats. Nous pouvons déterminer si nous avons trop de couches ou non. L'affichage permet de voir comment se répartie une classe que nous n'avons pas appris.

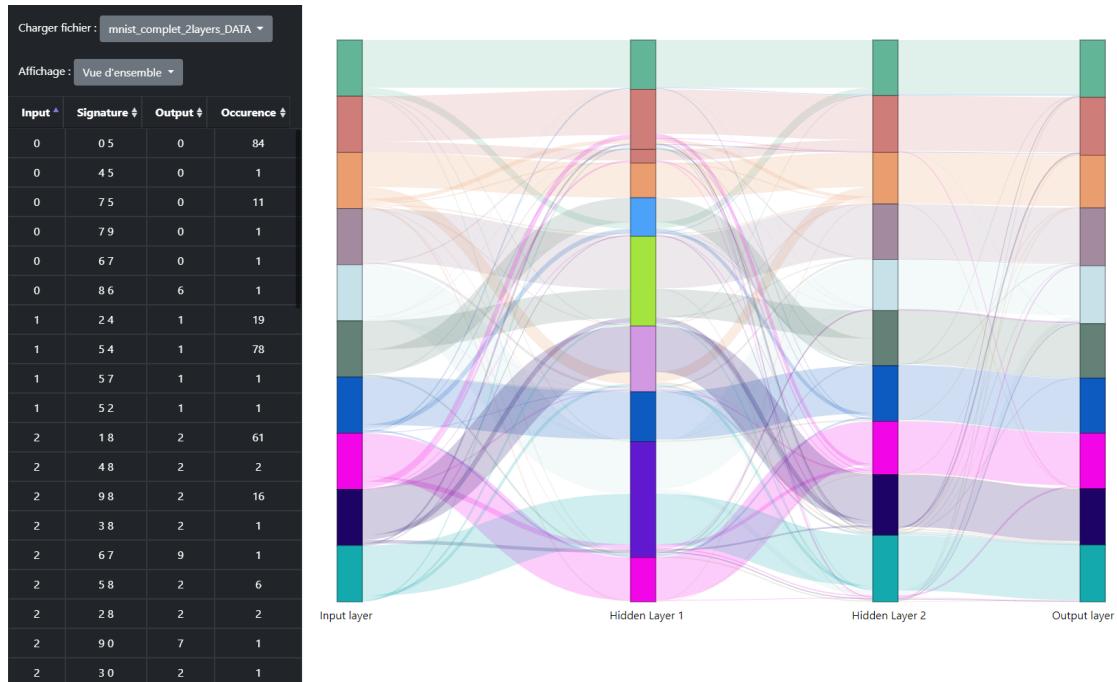


FIGURE 35 – Capture d'écran de l'interface avec l'affichage "vue d'ensemble"

7.2 Perspectives d'évolution du projet

Tout au long de ce projet, nous avons noté des idées et des fonctionnalités qui seraient intéressantes à ajouter.

Nous pourrions considérer considérer des réseaux plus complexes comme par exemple des réseaux de type CNN (e.g. Resnet avec x layers) ou encore intégrant des LSTM ou de l'attention visualisable figure 36.

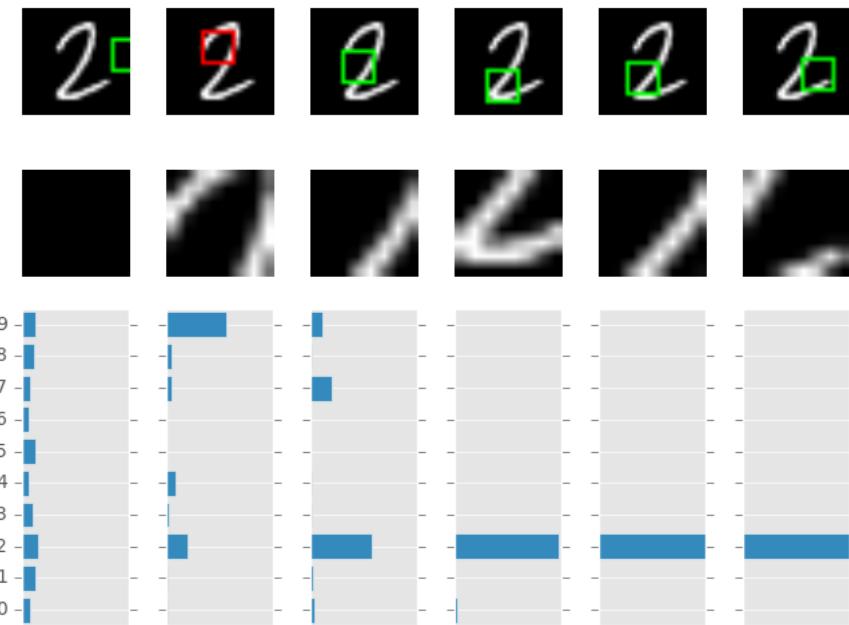


FIGURE 36 – Probabilité obtenu à l'aide du modèle attention

En effet, le modèle attention permet au réseau de ne se focaliser que sur une partie de l'image pour effectuer sa prédiction. Ici, à l'aide de certaines zones, le programme arrive sans difficulté à prédire le deux.

De plus, il serait intéressant d'avoir un programme plus interactif notamment en affichant les résultats des clustering afin de permettre à l'utilisateur de changer le nombre de clusters. Dans ce rapport nous avons utilisé K-means. Il serait intéressant d'utiliser d'autres approches comme dbscan ou encore t-sne.

7.3 Bilan technique et humain

Ce TER nous a énormément plu, en effet, étant étudiants en DECOL et amateurs de réseaux de neurones, le choix de ce TER était évident et intéressant pour nous. Il nous a apporté de multiples aspects sur le point de vue technique comme sur le point de vue humain.

Il nous a permis de découvrir le deep learning et l'apprentissage supervisé. Nous avons pu affiner nos connaissances en python et découvrir différentes librairies qui proposent énormément de fonctionnalités.

Sur le plan humain, nous avons appliqué les méthodes agiles afin de s'organiser au mieux tout au long du projet. Nous connaissant pour la plupart depuis l'IUT, il a été assez simple de connaître les forces et les faiblesses de chacun. Comme précisé précédemment dans le rapport, nous avons dû trouver des moyens pour continuer notre travail et conserver notre productivité en période de confinement.

Un projet d'une telle envergure nous demande toujours de s'investir. Il faut faire face aux multiples obstacles qui se dressent sur le chemin et il est nécessaire d'arriver à se remettre en question afin de faire avancer le projet dans une bonne dynamique. Le fait d'avoir un réel suivi auprès de notre tuteur nous a d'autant plus motivé, sachant que nous n'étions pas seul et pouvions demander de l'aide en cas de besoin et ainsi être guidé.

Nous avons eu la liberté de développer ce qu'il nous semblait nécessaire tout en respectant les consignes du projet et de notre tuteur. Cela nous a permis de laisser libre court à notre imagination et d'implémenter des idées nouvelles sur ce projet.

Nous espérons que ce rapport vous a intéressé et vous a permis également de mieux comprendre le fonctionnement de la boîte noire d'un réseau de neurones. Si ce n'est pas déjà fait, nous vous invitons à découvrir le github contenant le code et une vidéo présentant ce projet.

Github : https://github.com/AdrienL3/TER_RNP_M1

Vidéo : <https://www.youtube.com/watch?v=Jitaq5fWRUA>

Références

- [1] Github :
https://github.com/AdrienL3/TER_RNP_M1
- [2] Documentation RNP Pascal Poncelet :
Cours en ligne (Disponible sur l'ENT de l'Université de Montpellier 2)
Consulté le : 17/01/2020
- [3] Documentation Wikipedia RNA :
https://fr.wikipedia.org/wiki/Reseau_de_neurones_artificiels
Consulté le : 20/01/2020
- [4] Documentation MNIST :
https://en.wikipedia.org/wiki/MNIST_database
Consulté le : 09/02/2020
- [5] Documentation RNA :
<https://www.lebigdata.fr/reseau-de-neurones-artificiels-definition>
Consulté le : 10/02/2020
- [6] Documentation discréétisation + Python :
https://mpra.ub.uni-muenchen.de/76653/1/MPRA_paper_76653.pdf
Consulté le : 12/02/2020
- [7] Documentation Levenshtein :
https://www.python-course.eu/levenshtein_distance.php
Consulté le : 28/02/2020
- [8] Algos hiérarchique/densité et dendrogramme :
http://math.univ-bpclermont.fr/DoWellB/docs/malouche/methodes_classifications_CF_Juin2013.pdf
Consulté le : 29/02/2020
- [9] DBSCAN scikit-learn :
<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>
Consulté le : 04/03/2020
- [10] Clustering scikit-learn
<https://scikit-learn.org/stable/modules/clustering.html>
Consulté le : 11/03/2020
- [11] Documentation D3.js :
<https://d3js.org/>
Consulté le : 20/04/2020
- [12] Documentation sankey :
<https://www.d3-graph-gallery.com/sankey>
Consulté le : 23/04/2020
- [13] Visualizing the Hidden Activity of Artificial Neural Networks (Page 4) :
https://www.researchgate.net/publication/306049229_Visualizing_the_Hidden_Activity_of_Artificial_Neural_Networks
Consulté le : 08/05/2020

[14] Exemple utilisation modèle attention :

<https://modelzoo.co/model/ram>

Consulté le : 15/05/2020

[15] Exemple :

<https://modelzoo.co/model/ram>

Consulté le : 15/05/2020

Annexe

Exemple compte rendu réunion

CR Réunion du 27/02/2020

Lieu : FAC des Sciences BAT 16

Présents :

- Aurélien TROUCHE
- Loïc MARTIN
- Pierre PORTAL
- Adrien DIDIER

Points abordés:

- Discrétisation des données
- Utilisation de l'algorithme Levenshtein
- Clustering
- Explication des algorithmes hiérarchique et densité
- Explication de la matrice des distances

Décision :

- Nous avons décidé de discréteriser nos données
- Nous devons réaliser des clusters (Obtenir un dendrogramme)
- Nous avons opté pour MNIST
- Nous devons réfléchir sur l'interface graphique

Documents/Codes échangés :

- Codes discréterisation et Levenshtein

Date et heure de la prochaine réunion : 09/03/2020 - 11h

Lieu prochaine réunion : BAT 16

FIGURE 37 – Compte rendu réunion