Paris-Saclay university
Master 2 Data science

Apprentissage semi-supervisé par prédiction de rotations d'images

Realized by : DataSaiyentist

Supervised by :
Dr. Blaise Hanczar

University year : 2022 / 2023

Contents

List of Figures

1 Abstract

In this project, we will use different deep learning models for the classification of handwritten digit images from the MNIST dataset. However, we force ourselves to work with a reduced version of MNIST dataset (only 100 labels are available).

We tested different approaches to tackle the low-labeled issue, including preprocessing, feature extraction and data augmentation. But our main focus will be a self-supervised model named **RotNet** (based on Gildaris et al. [1]). The model consists of a convolutional neural network that is first trained to recognize image orientations with all unlabeled images. Then, thanks to transfer learning, that model is trained to recognize digits with the labeled ones.

Our results show that our RotNet is effective for classifying handwritten digits, achieving an honorable test accuracy. Our work was done using PyTorch and is published on :
https://github.com/DataSaiyentist/RotNet

In this report, we explained our models and analyzes their performances on the test data.

Key words: *PyTorch, RotNet, MNIST, handwritten digits, image classification, orientation recognition, data augmentation.*

2 Introduction

2.1 Présentation du projet

La classification d'images est une tâche cruciale dans le domaine de l'intelligence artificielle et du traitement d'images. Les réseaux de neurones profonds, en particulier les réseaux de neurones convolutifs (CNN), ont montré des performances exceptionnelles dans ce domaine. En effet, le Deep Learning est un outil puissant et à la mode qui permet de résoudre des tâches complexes comme nous le verrons dans ce projet même.

Dans ce rapport, nous étudierons l'application de plusieurs modèles de réseaux de neurones, tels que des perceptrons multicouche (MLP), des CNNs et notre RotNet, pour résoudre le problème de la classification d'images sur la base de données MNIST. Nous présenterons également les différentes étapes de mise en œuvre et d'évaluation de ces modèles.

2.2 Objectifs

L'objectif de ce projet est de développer des modèles de reconnaissance de chiffres manuscrits basés sur des modèles de deep learning et d'évaluer ses performances en utilisant la base de données MNIST. Néanmoins, nous sommes contraints concernant les données labellisées :
En fait, **nous disposons de seulement 100 labels** du jeu de données MNIST.
Il faut savoir que ce problème peut survenir dans la réalité, car l'annotation manuelle de données peut être fastidieuse, coûteuse et peu pratique.

Dans ce projet, nous avons certes implémenté naïvement quelques modèles de Deep Learning sans prendre en compte ce problème. Mais nous avons également implémenté une méthode semi-supervisée appelée **RotNet**. Effectivement, considérer du semi-supervisé permet de contourner notre problème en apprenant des représentations haut niveau sans avoir besoin d'un grand nombre de données labellisées. Ainsi, cette démarche permet de réduire le temps et les coûts associés à l'annotation manuelle des données.

Nos modèles sont construit avec PyTorch et sont entièrement disponible sur :
https://github.com/DataSaiyentist/RotNet

3 État de l'art

3.1 Présentation de la base de données MNIST

La base de données MNIST (Modified National Institute of Standards and Technology) est une collection de 70 000 images en niveaux de gris de chiffres manuscrits, réparties en 60 000 images d'entraînement et 10 000 images de test. Chaque image est de taille 28x28 pixels.

MNIST est largement utilisée comme base de référence pour évaluer les performances des algorithmes de classification d'images (en effet, reconnaître les chiffres consiste à classifier les images par rapport à 10 classes, chacune correspondant à un chiffre) et a largement contribué à l'avènement du Deep Learning [2] grâce au premier développement des CNNs.

Figure 1: Aperçu du jeu de données MNIST

3.2 MLP - Perceptron multicouche

Le perceptron multicouche est un modèle de réseau de neurones interconnectés. Il est simple et pratique à utiliser pour résoudre des tâches classiques de prédiction : Il est composé de plusieurs couches de neurones, chacune étant connectée à toutes les unités de la couche précédente et suivante. Bien que les MLPs soient moins performants que les autres modèles implémentés dans ce projet (moins adapté pour la classification d'images), ils peuvent toujours être utilisés, avec parcimonie, comme point de comparaison pour évaluer les performances des autres modèles.

3.3 CNN - Réseaux de neurones convolutifs

Les réseaux de neurones convolutifs sont des modèles de réseaux de neurones profonds spécialement conçus pour traiter des données structurées en plusieurs dimensions, telles que les images. Ils sont composés de plusieurs couches, notamment des couches de convolution, des couches d'activation non linéaire et des couches de pooling.
Les CNN ont démontré des performances exceptionnelles dans diverses tâches de vision par ordinateur, telles que la classification d'images, la segmentation d'images et la détection d'objets.

3.4 Présentation du modèle RotNet

RotNet est une méthode d'apprentissage semi-supervisée qui consiste à **entraîner un CNN à reconnaître la rotation appliquée à l'image d'entrée, puis (grâce au transfer learning) à l'entraîner sur une autre tâche comme la classification d'images.**

Plus précisément, pour entraîner le réseau, un ensemble de rotations est appliqué à toutes les images (non labellisées du jeu de données). On se retrouve alors avec quatre fois plus d'images et les labels de ces images sont alors les angles de rotation appliqués. Ensuite, le modèle est construit pour prédire ses angles de rotation (nous prendrons 0°, 90°, 180° et 270°). L'hypothèse sous-jacente de cette étape est que pour qu'un CNN soit en mesure de reconnaître la rotation qui a été appliquée à une image, il doit avoir compris les caractéristiques sémantiques des objets dans l'image, telles que leur emplacement, leur type, leur posture, etc.

Enfin, il suffit de reprendre ce modèle pré-entraîné et de l'entraîner sur une autre tâche (dans notre cas, la reconnaissance des chiffres) grâce aux données labellisées dont nous disposons.

Selon l'article [1] présentant ce modèle, le RotNet a été testé sur différentes bases de données d'images et a montré des performances équivalentes à celles des modèles traditionnels. Et dans le cas où le modèle dispose de peu de données labellisées, le RotNet démontre de meilleurs résultats que d'autres modèles de Deep Learning dans les mêmes conditions.

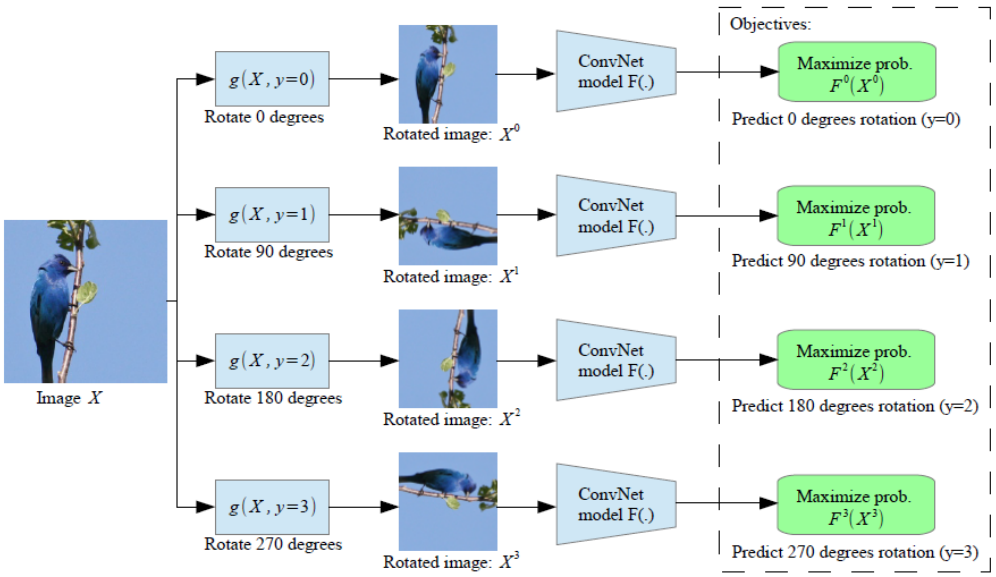

Figure 2: Feature learning grâce à la prédiction de la rotation

4 Traitement des données

4.1 Qualité des images

Pour rappel, les images du jeu de données MNIST sont en niveaux de gris. Ainsi, pour améliorer la qualité des données, on normalise de telle manière à ce que les pixels soient exclusivement en noir ou en blanc (ie. prennent les valeurs 0 ou 1).
Cette normalisation peut être faite automatiquement lors de l'importation du jeu de données (regardez la fonction `load_mnist(cleared = True)` dans `data.py`).

Figure 3: Exemple de normalisation

4.2 Extraction des données

Au cas où notre baseline, un CNN, n'arrive pas à s'entraîner pour la classification des chiffres (en raison de l'accès limité aux labels), nous aimerions entraîner une seconde baeline, ie. un MLP avec des variables plus puissantes pour compenser le problème. Une astuce bien connue consiste à extraire des caractéristiques de l'écriture, à savoir :

- la projection des pixels sur l'axe X
- la projection des pixels sur l'axe Y
- le profil gauche de l'écriture (ie. pour chaque ligne, on considère la coordonnée de la première transition de couleurs/ lecture de gauche à droite)
- le profil droit de l'écriture (ie. pour chaque ligne, on considère la coordonnée de la première transition de couleurs/ lecture de droite à gauche)

Pour chaque image, on se retrouve alors avec 4 vecteurs de taille 28. Si besoin, regardez la fonction `extract_mnist` dans `data.py` pour plus de détails.

4.3 Augmentation des données

Les réseaux de neurones convolutifs peuvent faire des choses remarquables s'il y a suffisamment de données. Cependant, trouver la quantité correcte de données d'apprentissage pour pouvoir entraîner des modèles de Deep Learning est une question difficile en raison de notre disposition (uniquement 100 images labellisées).

Pour rendre le réseau robuste, la méthode de data augmentation peut être considérée. Dans le cas du jeu de données MNIST, on peut alors s'amuser à appliquer plusieurs transformations au hasard pour augmenter considérablement le nombre de données labellisées (par rotation, translation, dilation, rétrécissement ou déformation/ cf. la fonction `extend_mnist` dans `data.py`).

Figure 4: Exemple d'augmentation des données (à gauche l'image originale)

Pour d'autres exemples, veuillez consulter le dossier `/save/dataAugmentation_examples`.

Notez tout de même que l'augmentation des données est une extension de notre travail. L'objectif principal reste toujours de regarder les performances du RotNet et de vérifier si ceux-ci sont satisfaisants ou non lorsque nous disposons de peu de labels.

5 Modélisation

Dans cette section, nous allons vous présenter l'ensemble de nos modèles, leur contexte d'apprentissage et leurs performances respectives. D'ailleurs, concernant cette section, nous effectuerons toujours un train-test split de 20% sur train : le nouveau train sera ainsi utilisé pour entraîner le modèle, le jeu de données de validation (résultant des 20% restant, pris aléatoirement) permettront au besoin d'ajuster le modèle et le test sera seulement utilisé pour évaluer le modèle.

L'ensemble de nos résultats sont disponibles dans le dossier `/notebooks`. Vous pouvez également accéder à tous nos modèles pré-entraînés dans le dossier `/save/models`.

5.1 MLP avec données extraites

Notre premier modèle est un MLP entraîné sur les 100 images labellisées. Mais comme dit précédemment (voir 4.2), on lui donne plutôt en entrée les variables extraites (le jeu de données correspondant est alors de taille 100x112, puisque 112 = 28 x 4).

Son architecture se trouve dans le fichier `model.py` mais le voici visuellement :

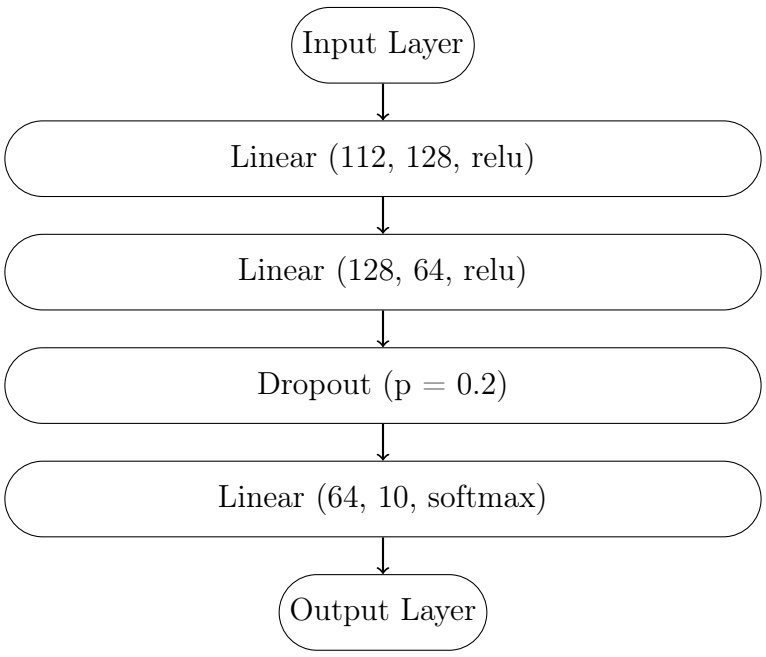

Figure 5: Architecture de notre MLPC

Ce modèle est entraîné en utilisant l'optimiseur `Adam` avec un pas d'apprentissage de 0.01 ainsi qu'une fonction de perte de type `CrossEntropy`. L'entraînement est effectué sur un nombre de lots fixe (80 pour les données d'entraînement soit l'entièreté des données et 20 pour les données de validation). Enfin, ce processus est répété sur 20 epochs jusqu'à ce que le modèle atteigne une précision satisfaisante sur l'ensemble de validation tout en évitant le surapprentissage (notez que la plupart du temps, nous considérerons les mêmes configurations pour l'entraînement, mis à part pour le nombre de lots puisque les données en entrée varient d'un modèle à un autre).

5.1.1 Courbe d'apprentissage

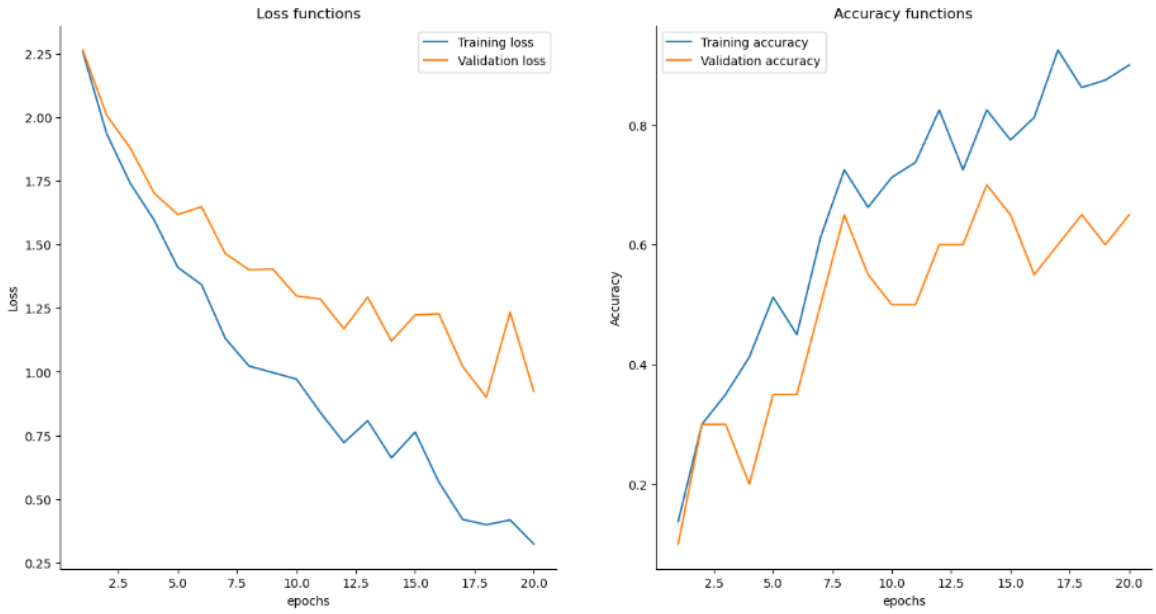

Figure 6: Courbe d'apprentissage du MLP

Les courbes d'apprentissage permettent de surveiller lors de l'entraînement l'évolution des performances sur l'ensemble d'entraînement et de validation. C'est un outil intuitif qui permet entre autres à identifier des problèmes de surapprentissages ou de sous-apprentissage.

Dans notre cas, le nombre d'epochs semble idéal, car les précisions semblent converger vers 0.9 pour train et 0.65 pour validation. D'ailleurs, il n'y a pas de surapprentissage en vue (car les fonctions de pertes semblent bien décroitre pour les deux jeux de données).

5.1.2 Evaluation sur test

On introduit un autre outil de performance qu'est la matrice de confusion, une représentation tabulaire des résultats. Elle permet de comparer les prédictions du modèle avec les labels réelles des données de test. Et pour le problème de reconnaissance des chiffres manuscrits de MNIST, la matrice de confusion afficherait les résultats d'une classification avec 10 classes.

En plus de cela, nous disposerons d'un rapport de classification calculant différentes métriques pour chaque classe de notre problème (cf. la fonction `evaluate_model` dans `utils.py`). Il nous permettra en particulier d'identifier les classes pour lesquelles le modèle a des difficultés à prédire correctement (nous orientant ainsi vers des pistes d'améliorations).

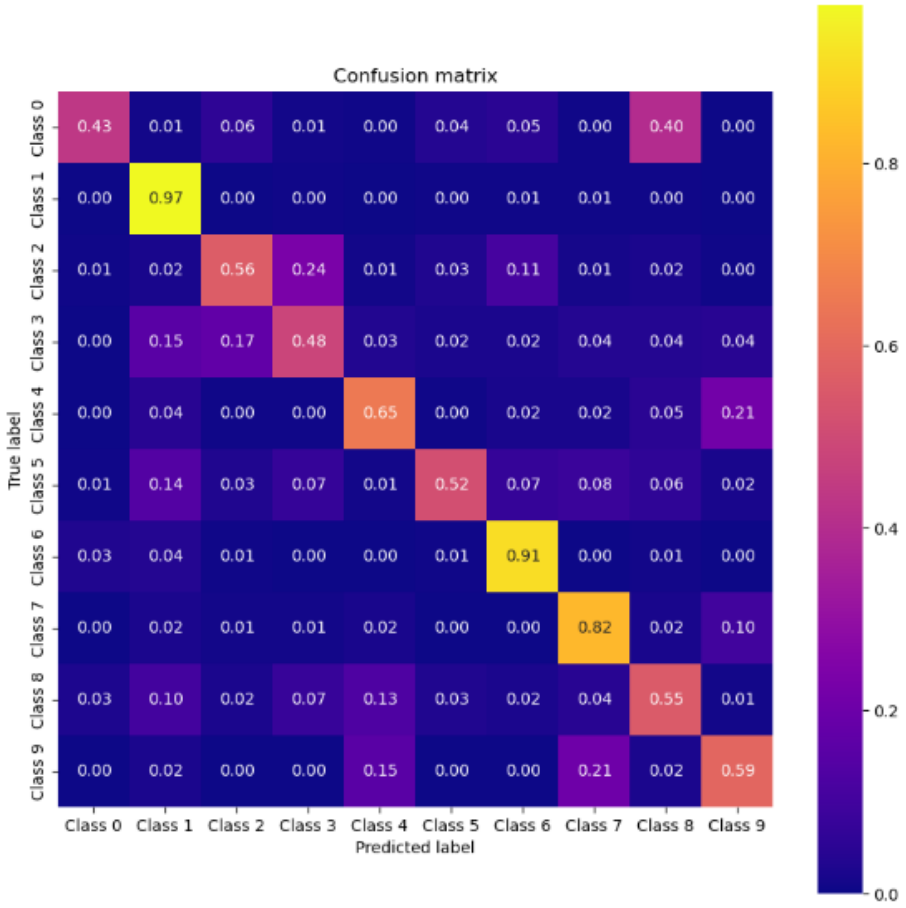

Figure 7: Matrice de confusion du MLP

Comme on peut le voir, la matrice de confusion est généralement présentée sous la forme d'un tableau avec les classes réelles représentées sur les lignes et les classes prédites représentées sur les colonnes. Les éléments diagonaux du tableau représentent le taux de prédictions correctes pour chaque classe, tandis que les éléments non diagonaux représentent les erreurs de classification. À présent, si on se lance sur son analyse, alors il semble que le modèle a pu réaliser de très bonnes performances pour certains chiffres (notamment le 1 et le 6), mais au contraire celui-ci a parfois un peu de mal avec certains chiffres (comme le 0 ou le 3).

Métriques	0	1	2	3	4	5	6	7	8	9
Précision	0.86	0.68	0.66	0.54	0.64	0.77	0.76	0.68	0.47	0.61
Rappel	0.43	0.97	0.56	0.48	0.65	0.52	0.91	0.82	0.55	0.59
F1-score	0.58	0.80	0.61	0.51	0.65	0.62	0.83	0.74	0.51	0.60
Support	980	1135	1032	1010	982	892	958	1028	974	1009

avec accuracy = 0.66

Figure 8: Rapport de classification du MLP

L'évaluation sur l'ensemble test est assez satisfaisante d'après la matrice de confusion et le rapport de classification. Mais nous ne souhaitons pas nous en arrêter là sachant que les CNNs sont des modèles mieux adaptés pour prendre en entrée des images.

5.2 CNN

Notre seconde baseline est un CNN entraîné sur 100 images labelisées (chacune de dimension 1x28x28). Nous avons construit une architecture peu profonde pour avoir une meilleure vitesse d'entraînement comparé à notre MLP. En effet, nous avons remarqué que l'entraînement était quasiment instantané comparé au MLP : avec une NVIDIA GeForce RTX 3060, cela a pris quelques microsecondes contre 1 min et 39 sec avec le MLP).

Le CNN est d'ailleurs entraîné avec les mêmes paramètres que le modèle précédent (cf. voir les remarques de la page 7) afin que le modèle atteigne des résultats acceptables.

Voici à la page suivante son architecture, qui est notamment accessible dans `model.py` :

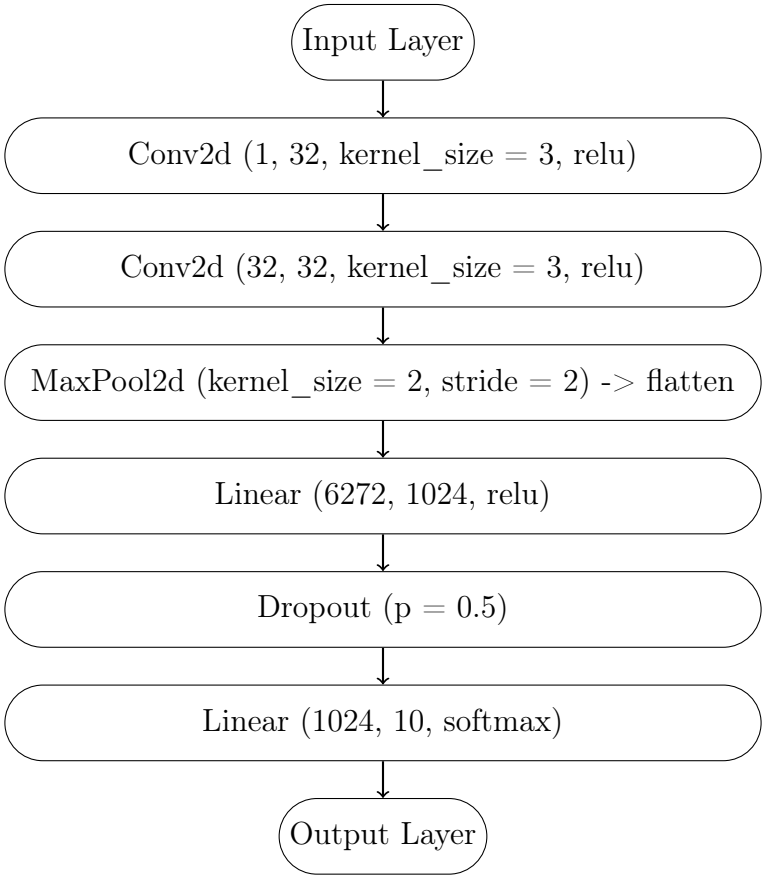

Figure 9: Architecture de notre CNN

5.2.1 Courbe d'apprentissage

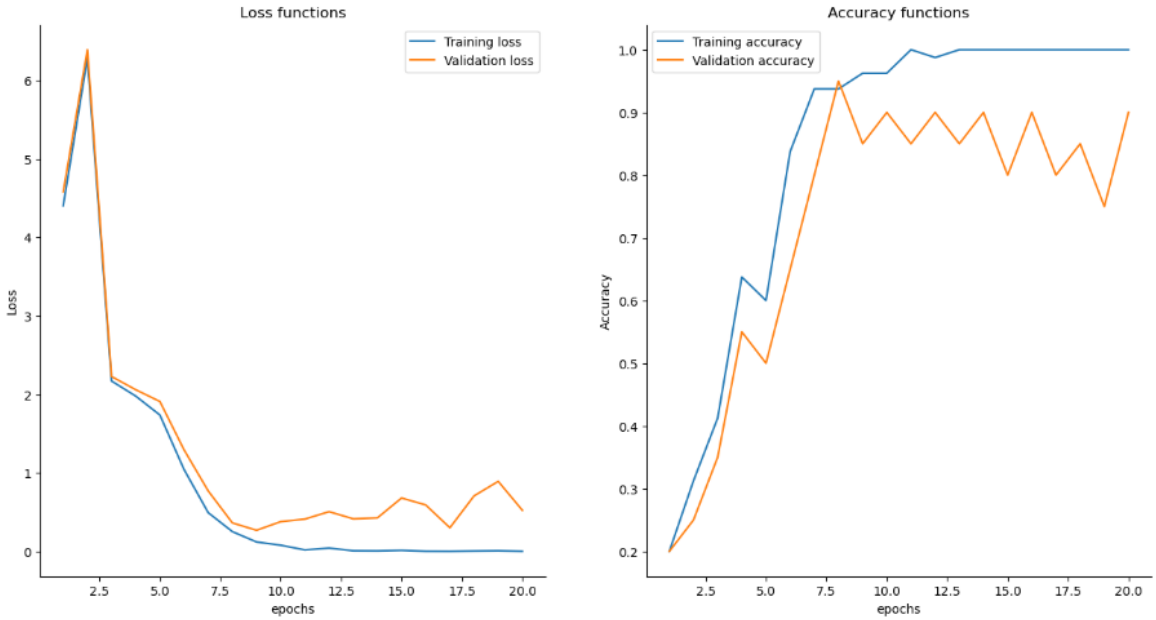

Figure 10: Courbe d'apprentissage du CNN

Les résultats sont satisfaisants (il semble mieux apprendre que le MLP en termes de performances : accuracy de 0.9 contre 0.65 sur les données de validation). Notons tout de même un certain début de surapprentissage vers la fin de l'entraînement, puisque l'accuracy sur validation semble commencer à décroître (et la fonction de perte à croître). C'est la raison pour laquelle, nous n'avons pas cherché à entraîner sur davantage d'epochs.

À présent, si on compare ses résultats plus en détail par rapport au MLP, alors il semble évident que ce modèle est meilleur. D'une part, il est plus rapide à entraîner (temps d'entraînement moins important et prend en entrée directement les images/ contrairement au MLP qui nécessite le calcul des nouvelles variables). D'autre part, si on regarde la fonction de perte, elle semble converger rapidement vers 0 pour train (et l'accuracy augmente plus rapidement).

5.2.2 Evaluation sur test

Maintenant, nous allons regarder ce que notre CNN donne sur les données test :

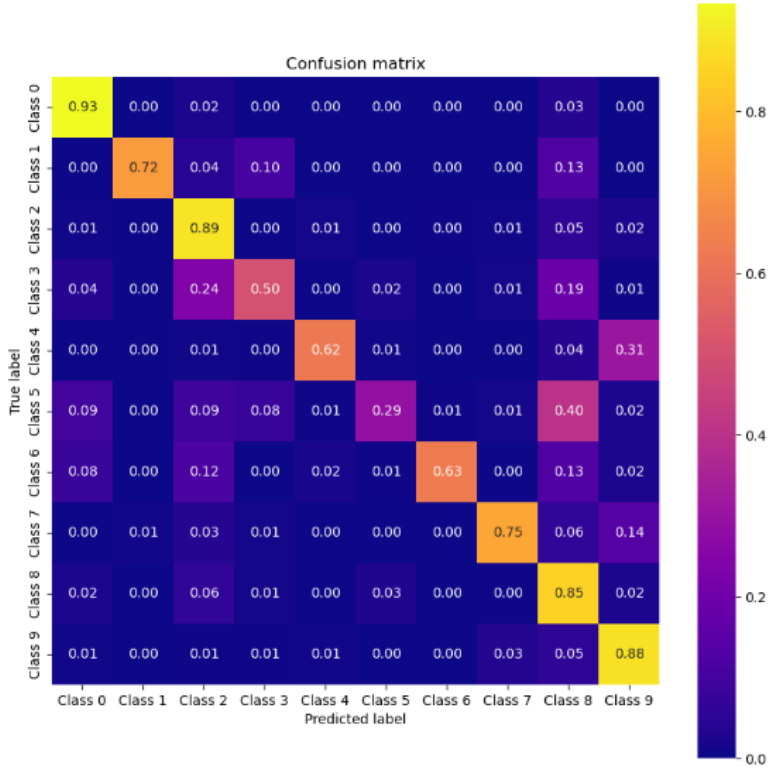

Figure 11: Matrice de confusion du CNN

Métriques	0	1	2	3	4	5	6	7	8	9
Précision	0.79	0.99	0.60	0.69	0.93	0.80	0.97	0.91	0.44	0.62
Rappel	0.93	0.72	0.89	0.50	0.62	0.29	0.63	0.75	0.85	0.88
F1-score	0.86	0.83	0.72	0.58	0.75	0.43	0.76	0.82	0.58	0.73
Support	980	1135	1032	1010	982	892	958	1028	974	1009

avec accuracy = 0.71

Figure 12: Rapport de classification du CNN

Notre CNN, en plus d'avoir de meilleures performances (accuracy de 0.71), semblent avoir de meilleures propriétés (il arrive à mieux prédire la plupart des classes). Mais tout ceci se fait au détriment de la bonne reconnaissance du chiffre 5, car seul 29% des images de cette classe sont bien reconnues (le reste étant en majorité prédit comme un 8).

Conséquemment, choisir parmi les deux baselines nécessiterait au préalable de consulter le cahier des charges (prendre le plus rapide des deux, celui qui classifie bien certains chiffres, le moins coûteux en mémoire, etc.) Pour notre part, nous aurions choisi le CNN. Et si besoin, pour régler son problème de classification, nous aurions fusionné les 5 et les 8 (l'astuce aurait alors été de mettre un autre modèle derrière le CNN pour distinguer ces deux chiffres).

5.3 RotNet

L'architecture de notre RotNet reste en soi un CNN. Cependant, pour que notre modèle apprenne de bonnes caractéristiques sémantiques, RotNet devra être plus profond que notre baseline (nécessitant plus de couches de convolution) comme vous pouvez le voir plus bas :

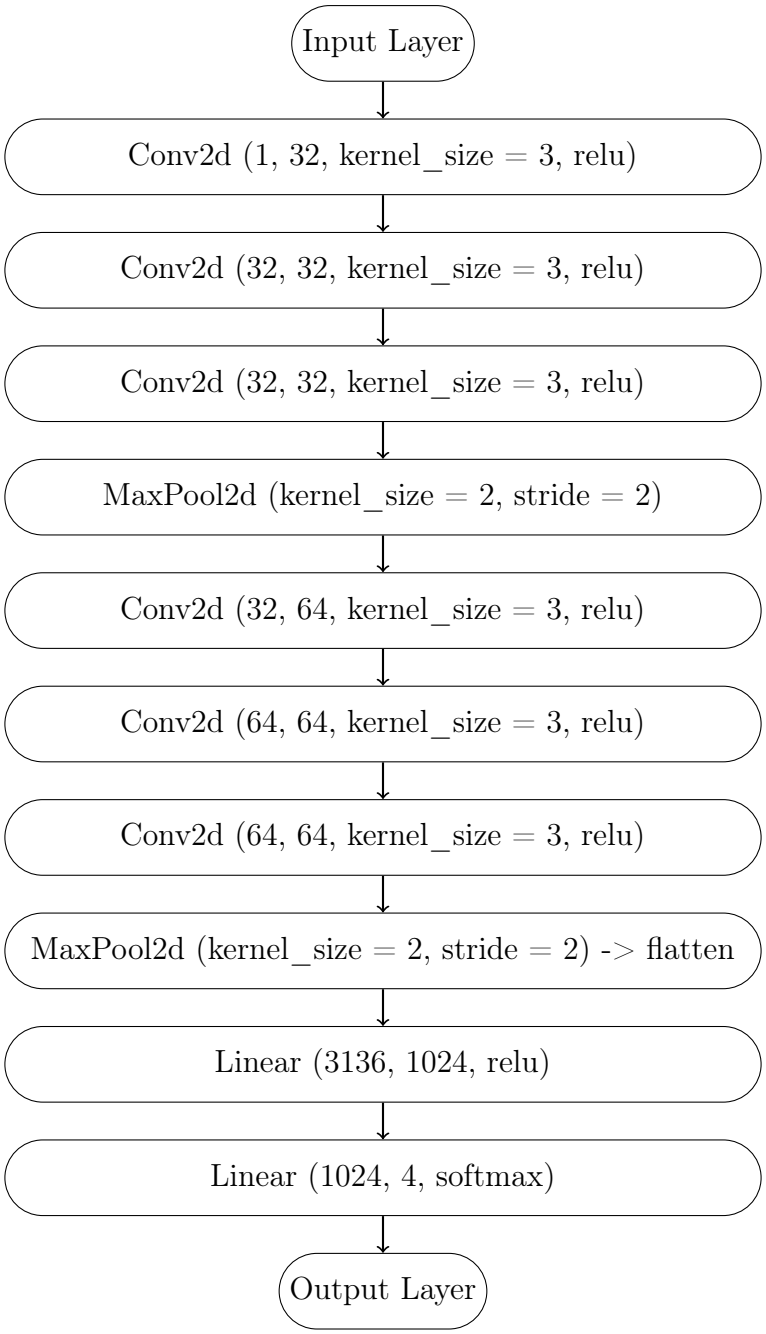

Figure 13: Architecture de notre RotNet

Notez que ce modèle ne comporte que 4 unités pour la sortie. En fait, ceci correspond à la première étape d'entraînement : nous l'entraînerons à prédire l'angle de rotation. Puis dans un second temps, nous fixerons les poids associés au bloc de convolution et nous re-entraînerons le reste sur les 100 images labellisées pour reconnaître les chiffres (soit 10 unités en sortie après).

Avant de poursuivre sur les résultats, nous souhaitions vous indiquer que notre RotNet n'est pas entièrement fidèle à l'article. Effectivement, leur modèle est construit pour classifier le jeu de données CIFAR-10 (dont les images ont des dimensions différentes, ie 32x32x3). Par conséquent, nous vous proposons une version adaptée de RotNet au jeu de données MNIST (avec les paramètres jugés optimaux par nos soins).

5.3.1 Courbe d'apprentissage

Nous analyserons ici exclusivement les résultats de la seconde étape (si nécessaire, regarder la 1ère étape dans `/notebooks/train.ipynb`/ son accuracy est de 0.99 sur test).

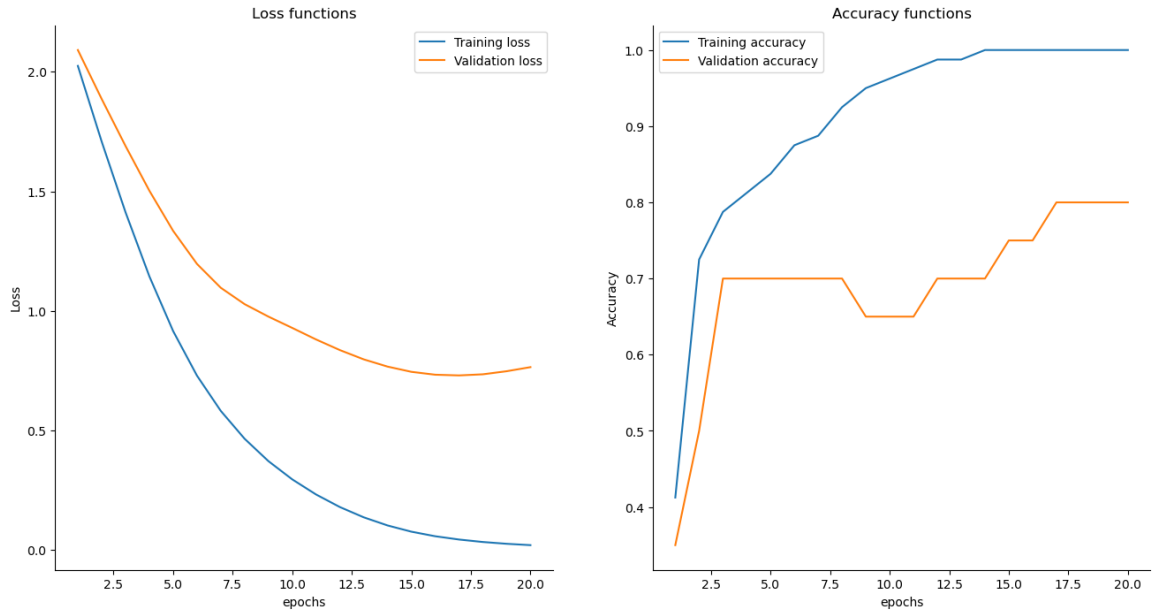

Figure 14: Courbe d'apprentissage du RotNet

Notre modèle semble particulièrement stable par rapport à nos autres modèles (les fonctions sont plus régulières). Mais si on comparait les tendances de performances, alors il semblerait que le CNN est plus avantageux que le RotNet. Pour autant, ces courbes de tendance ne sont pas si fiables parce que rappelons que l'ensemble de validation ne contient que 20 échantillons. Ainsi, nous devrions plutôt nous fier à l'évaluation sur test pour conclure sur ses performances.

5.3.2 Evaluation sur test

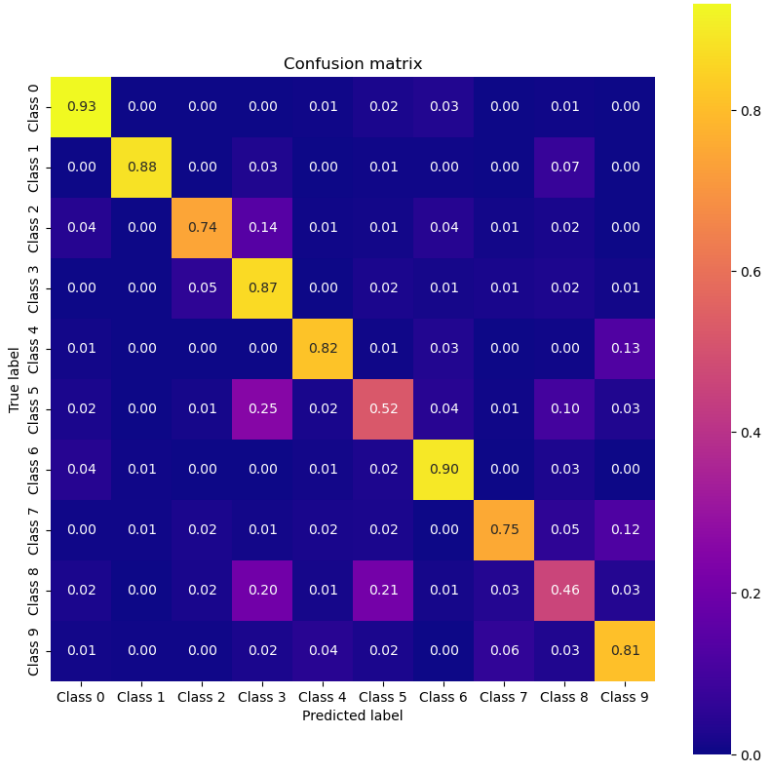

Figure 15: Matrice de confusion du RotNet

Métriques	0	1	2	3	4	5	6	7	8	9
Précision	0.86	0.98	0.88	0.58	0.88	0.59	0.83	0.86	0.58	0.72
Rappel	0.93	0.88	0.74	0.87	0.82	0.52	0.90	0.75	0.46	0.81
F1-score	0.90	0.93	0.80	0.70	0.85	0.55	0.86	0.80	0.51	0.76
Support	980	1135	1032	1010	982	892	958	1028	974	1009

avec accuracy = 0.77

Figure 16: Rapport de classification du RotNet

Par les présents résultats, nous venons de démontrer l'efficacité de l'approche semi-supervisé qui fait mieux que nos baselines. Il est vrai qu'il ne donne pas des résultats significativement plus important que ceux-ci (probablement parce que l'architecture considérée n'est pas la meilleure). Cependant, notre RotNet fournit tout de même des premières pistes d'attaques, de manière semi-supervisée, pour le problème de labellisation.

Si on compare ces évaluations au CNN, alors notre RotNet paraît moins sensible pour distinguer les 5 et les 8. En fait, il se "permet" de faire un peu plus d'erreurs dans toutes les classes pour pouvoir fournir des résultats honorables dans chaque classe.

5.4 BONUS : CNN et RotNet avec augmentation des données

On reprend les mêmes architectures que plus haut. La seule différence est qu'on va augmenter le nombre de données labellisées (à titre d'expérimentation, nous avons appliqué seulement 10 transformations par images, ce qui donne 1100 échantillons).

Sans rentrer dans les détails (cf. les résultats dans `/notebooks/train.ipynb`), nous obtenons de très loin les meilleurs résultats (jusqu'à 0.83 pour le RotNet et 0.85 avec le CNN sur test). Ce n'est pas pour autant qu'il faut délaisser nos anciens modèles. En effet, l'augmentation des données est peut-être mieux adapté à nos données (si on prenait des images plus complexes à classifier, comme CIFAR-10, on n'aurait probablement pas la même conclusion).

Mais dans notre cas, il semblerait que l'augmentation des données, combinées au semi-supervisé, soit une piste d'amélioration envisageable pour faire face à peu de données labellisées.

6 Conclusion

En conclusion, ce projet nous a montré qu'il est possible de construire un modèle de reconnaissance de chiffres manuscrits en disposant de peu de données labellisées.

Nous admettons que certaines de nos architectures et paramétrisations ne sont pas les plus optimales. Pourtant, les résultats semblent très prometteurs et confirment les dires de l'article de référence [1]. Dans l'ensemble, ce projet a montré l'importance de la préparation des données et de la construction d'un modèle efficace pour obtenir des résultats précis pour la reconnaissance d'images. Nous concernant, nous sommes satisfaits de nos résultats. Implémenter RotNet a été une expérience enrichissante pour nous. Cela nous a donné l'opportunité de travailler avec Pytorch pour la première fois, ce qui a été une expérience précieuse. Cela nous a également permis de découvrir le paradigme de l'apprentissage semi-supervisé.

Il reste encore beaucoup à explorer dans le domaine de la reconnaissance d'images. Les perspectives d'amélioration de notre projet pourraient inclure l'utilisation de techniques plus avancées d'augmentation de données (comme les GANs) pour améliorer les performances du modèle, ainsi que l'exploration d'autres modèles semi-supervisé pour apprendre des caractéristiques sémantiques plus poussées qu'avec le RotNet.

References

[1] Spyros Gidaris, Praveer Singh, Nikos Komodakis, "Unsupervised representation learningby predicting image rotations", 2018, ICLR

[2] Fu Jie Huang, Y-Lan Boureau, Yann LeCun, et al. "Unsupervised learning of invariant feature hierarchies with applications to object recognition in Computer Vision and Pattern Recognition", 2007, IEEE Conference on, pp. 1–8.

[3] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research", 2012, IEEE Signal Processing Magazine, vol. 29, no 6, p. 141-142