Reconnaissance automatique de caractères arabes

Travail réalisé par : NGUYEN-DUC Cédric AHALLAT Yassine
Deep learning - CentraleSupélec
Avril 2018

1 Abstract

La reconnaissance d'images est une des applications les plus répandues de l'apprentissage profond. Elle consiste à pouvoir affecter à une image en entrée, un label pour la classifier. Pour cela, plusieurs approches sont possibles et plusieurs types de réseaux de neurones peuvent être utilisés. Nous avons abordé dans le cadre de ce projet la problématique de reconnaissance de caractères arabes saisis à la main. Pour cela, nous avons choisi de recourir à deux solutions concurrentes : un Multi Layer Perceptron et un Réseau Convolutif. Nous avons donc pu évaluer et comparer les performances de chacun de ces deux réseaux appliqués à notre problématique. Nous décrivons dans ce document l'approche adoptée pour les deux solutions, la justification de nos choix de paramètres ainsi que les résultats obtenus suite à la cross validation. Nous avons finalement obtenu une précision de 81.42% pour le MLP après huit crossvalidations, et 94.94% pour la solution basée sur un CNN.

2 Introduction

Depuis quelques années, le domaine du Deep Learning connaît un essor considérable, et ce, notamment grâce à l'augmentation de la puissance de calcul disponible sur machine. C'est un domaine qui a atteint des performances révolutionnaires dans des domaines tels que la reconnaissance de langage, d'image ou le big data en général. La reconnaissance de caractères s'applique dans de nombreux cas comme l'automatisation de tâches bureautiques, la lecture d'adresse postale ou de manuscrits. Par ailleurs, la langue arabe est parlée par plus de 260 millions de personnes et a une place importante dans beaucoup de cultures différentes. L'arabe a de nombreuses variantes à l'oral mais dispose d'une unique version standard à l'écrit, et elle est notamment utilisée dans tous les cadres officiels. L'automatisation de la reconnaissance de cette langue à l'écrit aurait donc de grands bénéfices. L'une des caractéristiques des lettres arabes est que leur forme varie en fonction de leur position dans le mot, en effet, pour certaines lettres, l'écriture est différente selon si la lettre est au début, au milieu ou à la fin du mot. L'alphabet arabe contient 28 caractères comme montrés dans le tableau ci-dessous :

خ	ح	ج	ث	ت	ب	ا
kha	haa	jiim	thaa	taa	baa	alif
ص	ش	س	ز	ر	ذ	د
saad	shiin	siin	zaay	raa	thaal	daal
ق	ف	غ	ع	ظ	ط	ض
qaaf	faa	ghayn	ayn	thaa	taa	daad
ي	و	ه	ن	م	ل	ك
yaa	waaw	ha	nuun	miim	laam	kaaf

FIGURE 1 – Alphabet arabe

3 Etat de l'art

Plusieurs méthodes ont été mises en place pour la reconnaissance de chiffres manuscrits. Tout d'abord on peut citer le CNN couplé avec SVM (Support Vector Machine) de Xiao-Xiao Niu , Ching Y. Suen en 2012. Leurs expériences ont été réalisées avec la base de données MNIST et ont permis d'atteindre 94.40% de reconnaissance avec succès. Miguel D. Tissera et Mark D. McDonnell ont proposé une solution basée sur de l'ELM (Extreme Machine Learning) pour classifier des chiffres manuscrits latins qui atteint 99.19%.
En ce qui concerne les lettres arabes, on peut citer Mohamed Elleuch, Najiba Tagougui, Monji Kherallah et leurs travaux sur un DBNN (Deep Belief Neural Network), avec une erreur de classification de seulement 2.1% sur la base de données HACDB. En 2014, Marwa Rashad et Noura A. Semary ont eux étudié l'efficacité d'un KNN (K - Nearest Neighbours) et d'un RFT (Random Forest Tree) et ont déterminé que le RFT était meilleur de 11%. Des études ont été également menées par Ramzi Haraty et Catherine Ghaddar sur deux MLP, et ont obtenu un résultat de 73% de réussite. En 2013, Iping Supriana et Albadr Nasution ont entre autres travaillé sur un système de classification de lettres arabes par OCR (Optic Characters Recognition) et ont obtenu une précision de 82%. Enfin, on pourrait citer l'étude de Majida Ali Abed et Hamid Ali Abed A lasad dont le EBPANN (Error Back Propagation Artificial Neural Network) a montré une performance de 93.61%.

4 Approche gérérale

Afin de pouvoir obtenir une base de données exploitable pour la reconnaissance de lettres arabes, il y a beaucoup d'étapes chronophages dont la récupération de nombreux échantillons, le scan, la segmentation et la traduction en vecteurs ligne de pixels. Nous avons cependant pu nous procurer un dataset de 13440 images d'entraînement et 3360 images de tests toutes labellisées (28 labels différents) grâce au site Kaggle. Nous avons donc choisi d'entraîner notre MLP ainsi que notre CNN sur ce dataset. Bien que nous ayons recouru à deux solutions différentes, le protocole expérimental reste rigoureusement la même. Après avoir charger les

données à partir de fichiers csv, nous avons découpé l'ensemble d'entraînement en 5 sous-ensembles pour pouvoir effectuer la cross-validation. Nous fixons au début nos hyper paramètres à des valeurs qui nous paraissent réalistes compte tenu des valeurs choisies dans l'état de l'art pour des problèmes semblables. Suite à cela, nous entraînons notre réseau sur les 4/5ème de l'ensemble d'entraînement, puis nous constatons les performances sur les données de validation (le 1/5ème restant). Nous effectuons cette opérations à 5 reprises pour avoir, à chaque itération, un ensemble de validation différent, et par conséquent des résultats qui dépendent moins de la distribution des données. Ensuite, nous changeons un (ou des) hyper paramètres puis nous réitérons le même processus jusqu'à obtention des meilleurs performances.

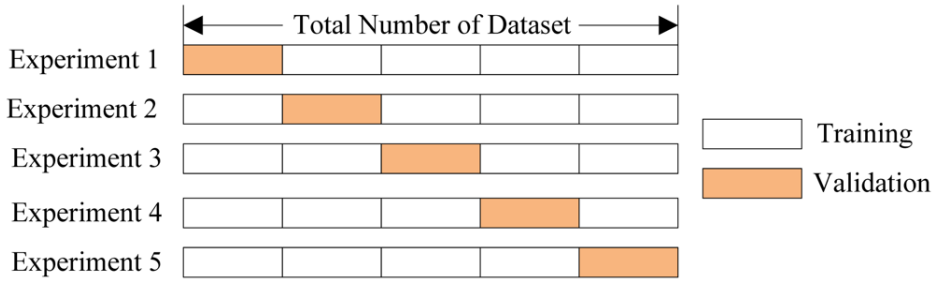

FIGURE 2 – Cross validation

Une fois la meilleure configuration d'hyper paramètre obtenue, nous testons les performances de notre réseau sur les données de test, que ne sont pas intervenues jusque-là.

5 Approche par Multi Layer Perceptron

5.1 Approche expérimentale

Un Multi Layer Perceptron est un réseau neuronal direct sous forme d'empilement de plusieurs couches de perceptrons. Chaque couche est complètement connectée aux autres couches. Le principe est d'initialiser aléatoirement les poids qui lient les couches et de les apprendre en ayant recours à la descente de gradient (normale ou stochastique). L'équation qui régit les entrées et la sortie unique de chaque perceptron est la suivante :

$$output = \sum_{j=1}^{n}(w_{ij} * x_j + \theta_j)$$

FIGURE 3 – Fonction de transfert d'un perceptron

L'objectif est de réduire une fonction de coût obtenue en appliquant le principe de maximum de vraisemblance, ou de log-vraisemblance.
En pratique, nous devons définir l'architecture du réseau et modifier des hyper paramètres pour obtenir les meilleures performances possibles.

5.2 Expérimentations sur les hyperparamétres

Nous disposons de plusieurs hyperparamètres sur lesquels nous pouvons jouer pour obtenir les meilleures performances. Nous allons dans cette partie présenter différentes configurations d'hyper paramètres ainsi que leurs performances dans la cross validation.

Le seul hyperparamètre que nous avons chosi de ne pas modifier est la fonction de coût Softmax. En effet, nous sommes devant un problème de classification où il faut choisir une solution parmi 28 en sortie. La fonction Softmax est conçue de telle façon à maximiser la probabilité d'une sortie par rapport aux autres et est donc la plus adaptée à notre problématique.

$$Softmax = \frac{e^{z_j}}{\sum e^{z_i}}$$

FIGURE 4 – Softmax

5.2.1 Première cross validation : 2 couches cachées de 256 neurones / couche, batch de taille 50, 90 epochs, Non linéarité Relu, fonction de coût softmax.

```
Validation accuracy for the first validation set : 0.502976
Validation accuracy for the second validation set : 0.738839
Validation accuracy for the third validation set : 0.640625
Validation accuracy for the fourth validation set : 0.729539
Validation accuracy for the fifth validation set : 0.686384
```

Nous pouvons approcher les performances de cette combinaison d'hyperparamètres à une moyenne de **65,9%** de précision
Nous avons ensuite fait le choix de doubler le nombre de neurones par couche, sans augmenter la profondeur du réseau, voici donc la nouvelle configuration et les résultats associés :

5.2.2 Deuxième cross validation : 2 couches cachées de 512 neurones / couche, batch de taille 50, 90 epochs, Non linéarité Relu, fonction de coût softmax.

```
Validation accuracy for the first validation set : 0.517857
Validation accuracy for the second validation set : 0.751116
Validation accuracy for the third validation set : 0.650298
Validation accuracy for the fourth validation set : 0.753348
Validation accuracy for the fifth validation set : 0.688616
```

Les performances ont augmentés pour atteindre **67.1%.** L'augmentation du nombre de neurones a eu donc un impact positif sur le réseau.
Nous avons voulu tester l'influence de la profondeur du réseau sur les performances. Ce qui nous a mené au test avec la configuration suivante :

5.2.3 Troisième cross validation : 3 couches cachées de 256 neurones / couche, batch de taille 50, 90 epochs, Non linéarité Relu, fonction de coût softmax.

```
Validation accuracy for the first validation set : 0.481399
Validation accuracy for the second validation set : 0.713542
Validation accuracy for the third validation set : 0.622396
Validation accuracy for the fourth validation set : 0.704241
Validation accuracy for the fifth validation set : 0.651042
```

Il n'y a pas d'amélioration de performances par rapport aux conditions initiales, au contraire, nous remarquons une légère baisse des performances à **63.4%** de précision.

Nous avons ensuite testé l'influence de la taille du batch sur le réseau, nous avons donc doublé la taille du batch, et gardé les même autres hyperparamètres que pour la première configuration.

5.2.4 Quatrième cross validation : 2 couches cachées de 256 neurones / couche, batch de taille 100, 90 epochs, Non linéarité Relu, fonction de coût softmax.

```
Validation accuracy for the first validation set : 0.494048
Validation accuracy for the second validation set : 0.736607
Validation accuracy for the third validation set : 0.615327
Validation accuracy for the fourth validation set : 0.735491
Validation accuracy for the fifth validation set : 0.688988
```

Nous constatons que doubler la taille du batch n'a pas eu une grande influence sur les performances **(65,4%)**. Nous aurions dû changer encore plus la taille du batch pour pouvoir clairement lire l'effet de cet hyperparamètre sur les performances.
Nous avons voulu capitaliser sur les bons résultats obtenus en doublant le nombre de neurones par couche, et ajouter à cela un plus de profondeur au réseau. Nous avons donc testé les performances sur la combinaison de hyperparamètres suivante :

5.2.5 Cinquième cross validation : 4 couches cachées de 512 neurones / couche, batch de taille 50, 90 epochs, Non linéarité Relu, fonction de coût softmax.

```
Validation accuracy for the first validation set : 0.468006
Validation accuracy for the second validation set : 0.713914
Validation accuracy for the third validation set : 0.633185
Validation accuracy for the fourth validation set : 0.707589
Validation accuracy for the fifth validation set : 0.636161
```

Les performances avoisinent les **63%** de précision. Nous constatons que l'augmentation de la profondeur du réseau n'est pas toujours bénéfique. Cela dépend du nombre de neurones dans chaque couche.

Pour le test suivant, nous avons décidé de garder la meilleur configuration obtenue jusqu'à présent (512 neurones / couche avec 2 couches cachées), mais en modifiant cette fois la non linéarité utilisée. Nous avons donc fait le choix de recourir à la fonction d'activation Sigmoid.

5.2.6 Sixième cross validation : 2 couches cachées de 512 neurones / couche, batch de taille 50, 90 epochs, Non linéarité Sigmoid, fonction de coût softmax.

```
Validation accuracy for the first validation set : 0.473958
Validation accuracy for the second validation set : 0.685268
Validation accuracy for the third validation set : 0.592634
Validation accuracy for the fourth validation set : 0.709821
Validation accuracy for the fifth validation set : 0.613095
```

Le changement de la fonction d'activation nous a mené à un résultat de **61.46%** de précision, ce qui n'est pas satisfaisant par rapport aux résultats antécédents. Nous avons donc fait le choix de modifier encore une fois cet hyperparamètre, et de choisir la tangente hyperbolique comme fonction d'activation. Voici donc les nouvelles conditions de cross-validation :

5.2.7 Septième cross validation : 2 couches cachées de 512 neurones / couche, batch de taille 50, 90 epochs, Non linéarité Tangente hyperbolique, fonction de coût softmax.

```
Validation accuracy for the first validation set : 0.495164
Validation accuracy for the second validation set : 0.716518
Validation accuracy for the third validation set : 0.615327
Validation accuracy for the fourth validation set : 0.708705
Validation accuracy for the fifth validation set : 0.648438
```

Le choix de la tangente hyperbolique n'améliore pas spécialement les performances.
Nous allons maintenant quadrupler le nombre de neurones, et revenir à la fonction d'activation Relu, et observer les résultats.

5.2.8 Huitième cross validation : 2 couches cachées de 1024 neurones / couche, batch de taille 50, 90 epochs, Non linéarité Relu, fonction de coût softmax.

```
Validation accuracy for the first validation set : 0.521577
Validation accuracy for the second validation set : 0.752976
Validation accuracy for the third validation set : 0.653646
Validation accuracy for the fourth validation set : 0.756324
Validation accuracy for the fifth validation set : 0.701637
```

Les performances ont atteint une moyenne de **67.7%**.
C'est le meilleur résultat que l'on ait obtenu jusqu'à présent, après huit cross validations.
Bien que nous aurions pu sans doute améliorer encore plus ces performances en variant d'avantage les hyperparamètres, nous avons décidé d'en rester là et voir les résultats finaux sur l'ensemble de test avec cette même dernière configuration d'hyperparamètres.

5.3 Test

En dernier lieu, nous testons les performances de notre MLP sur les 3360 images de test, qui étaient mises à l'écart jusque là. Nous avons gardé la configuration de la dernière cross validation. Nous avons donc finalement obtenu une précision de **81.42%**.

```
Test accuracy: 0.814286
```

Nous avons jugé ce résultat satisfaisant compte tenu de la présence de 28 classes différentes en sortie, mais aussi en comparaison avec les résultats obtenus dans l'état de l'art actuel.

6 Approche par un Convolutional Neural Network

6.1 Approche expérimentale

Notre objectif est maintenant de trouver la meilleure configuration de paramètres pour un CNN. Pour obtenir des résultats exploitables, nous avons mis en place le même mode opératoire pour chaque validation avec le même dataset.Tous les tests suivants ont été réalisés sur le même processeur. La seule différence étant le paramètre que l'on a modifié.

Comme nous l'avons fait pour le MLP, nous avons entraîné notre CNN et nous y avons appliqué la cross validation. On évalue ensuite l'efficacité du modèle. Nous répétons alors l'expérience

avec un des paramètres modifié dans le but de se rapprocher d'une architecture optimale. Nous avons ajouté ici le temps de calcul afin de pouvoir étudier la performance mesurée pour un temps de calcul donné (en secondes). Nous sommes partis d'une architecture classique qui nous paraissait adaptée à la tâche voulue, à savoir :

- une entrée de 32*32 car notre entrée est un ensemble d'image 32*32 (1024 pixels)
- une couche de convolution de 80 filtres 3*3
- une couche de convolution de 64 filtres 3*3
- 3 couches fully connected pour obtenir une classification de 1 parmi 28 (car 28 lettres dans l'alphabet)

La convolution permet de repérer des features dans l'image. Cela peut être des features de bas niveau comme des contours partiels, mais aussi de haut niveau comme la lettres en entier. Nous avons rajouté du max pooling à notre architecture afin de réduire les dimensions de l'image traitée, ce qui a pour effet de diminuer le nombre de paramètres, et donc le temps de calcul. De plus, il permet d'isoler les features dont le réseau est sûr, et d'éliminer d'éventuels "bruits" qui pourrait fausser le résultat. Voici, ci-dessous, une représentation schématique d'un max pooling :

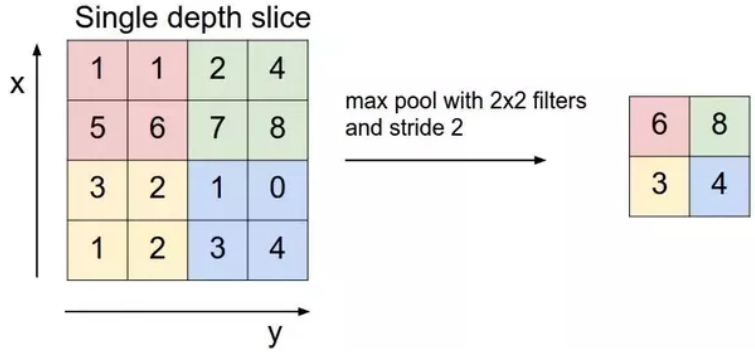

FIGURE 5 – Max Pooling

On voit sur ce schéma que seulement la plus grande valeur est retenue par échantillon de la taille du filtre choisi. Ici le stride est de 2, donc le filtre se déplacera de 2 pixels en 2 pixels. On obtient donc une sortie 2*2 à partir d'une entrée de 4*4. Nous avons aussi choisi d'utiliser la technique de régularisation dropout pour éviter le phénomène d'overfitting en supprimant temporairement certains neurones du réseau choisis aléatoirement. Enfin, en ce qui concerne la fonction d'optimisation, nous avons choisis la descente de gradient stochastique, qui est l'une des plus utilisée.

6.2 Expérimentation sur les hyperparamétres

On présentera dans cette partie, les différentes validations de paramètres. On commence tout d'abord par valider notre CNN de départ.

6.2.1 Cross validation 1 :

- Input : 32*32
- CL1 : 80 3*3 filtres
- Max Pooling : 2*2
- CL2 : 64 3*3 filtres
- Max Pooling : 2*2

— fully connected : 1024 neurons
— dropout : 0.8
— fully connected : 512 neurons
— dropout : 0.8
— fully connected : 28 neurons
— optimisation : Stochastic Gradient Descent

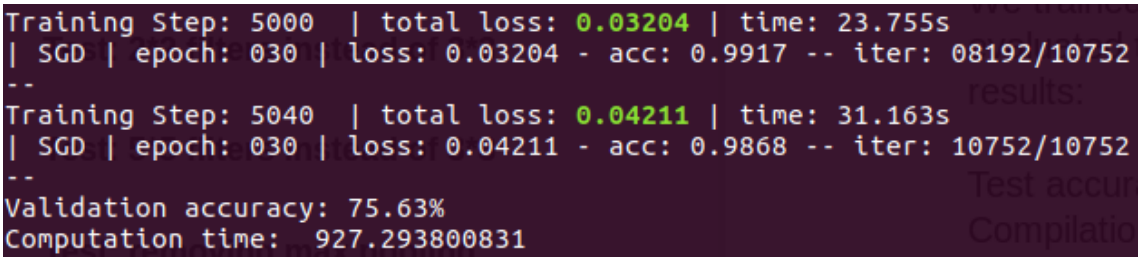

FIGURE 6 – 1$^{\text{ére}}$ validation : Score de validation : 75.63%

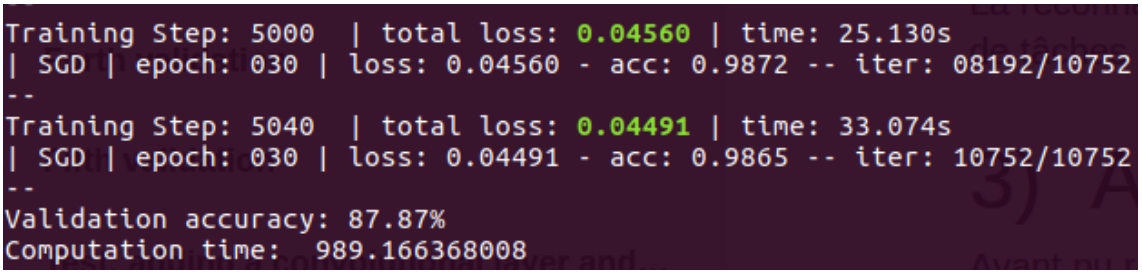

FIGURE 7 – 2$^{\text{ème}}$ validation : Score de validation : 87.87%

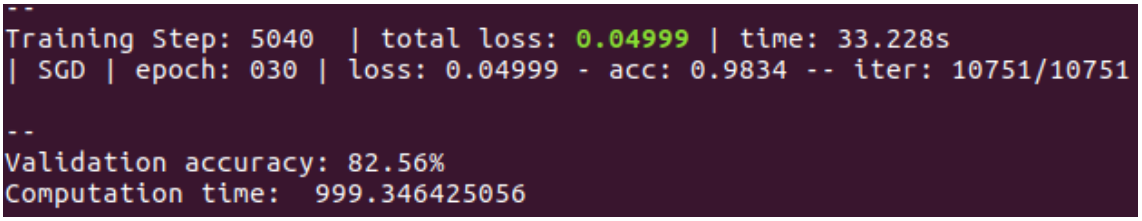

FIGURE 8 – 3$^{\text{ème}}$ validation : Score de validation : 82.56%

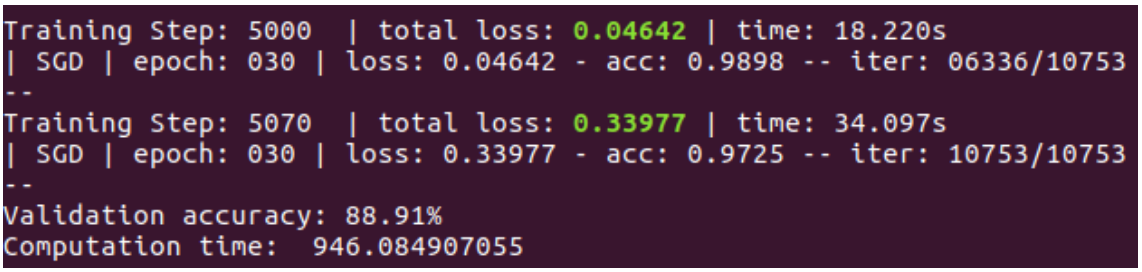

FIGURE 9 – 4ème validation : Score de validation : 88.91%

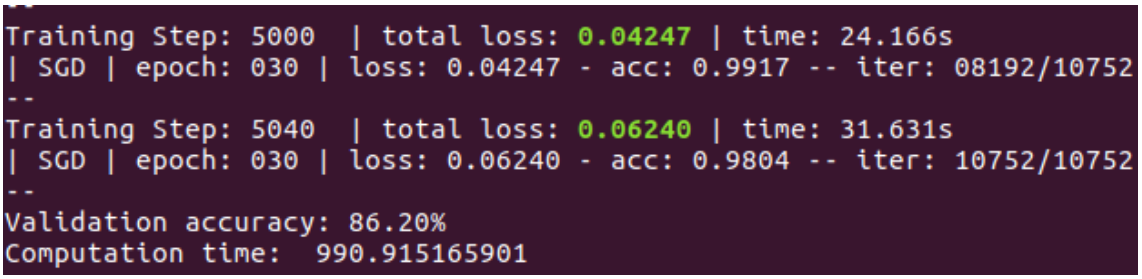

FIGURE 10 – 5ème validation : Score de validation : 86.20%

A l'issue de cette cross validation, on calcule le taux de réussite total :

$$\frac{75.63 + 87.87 + 82.56 + 88.91 + 86.20}{5} = 84,234\%$$

Nous obtenons un résultat de **84,234%** sur le modèle de base. Augmentons maintenant le nombre de filtres de la 1ére couche de convolution et observons le résultat.

6.2.2 Cross validation 2 :

- Input : 32*32
- CL1 : **110** 3*3 filtres
- Max Pooling : 2*2
- CL2 : 64 3*3 filtres
- Max Pooling : 2*2
- fully connected : 1024 neurons
- dropout : 0.8
- fully connected : 512 neurons
- dropout : 0.8
- fully connected : 28 neurons
- optimisation : Stochastic Gradient Descent

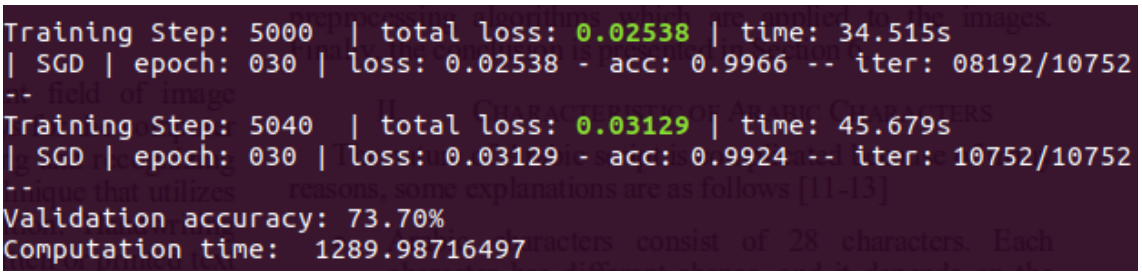

FIGURE 11 – 1ére validation : Score de validation : 73.70%

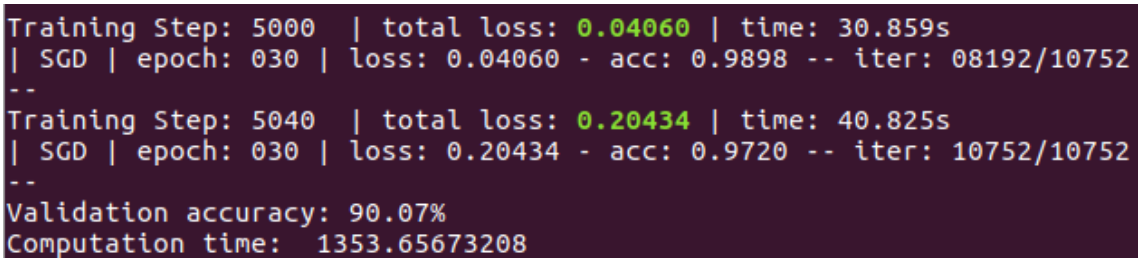

FIGURE 12 – 2ème validation : Score de validation : 90.07%

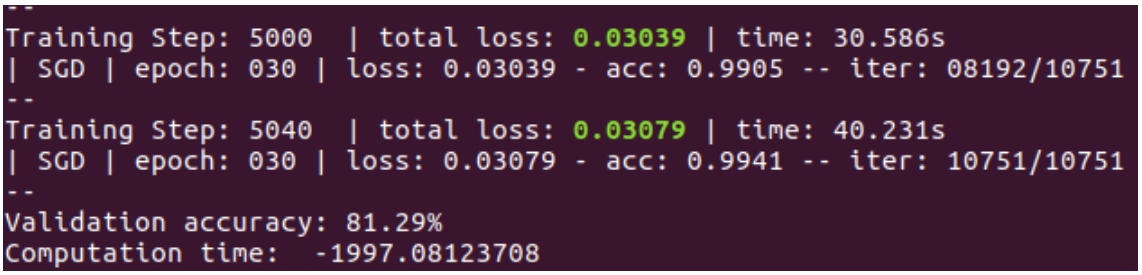

FIGURE 13 – 3ème validation : Score de validation : 81.29%

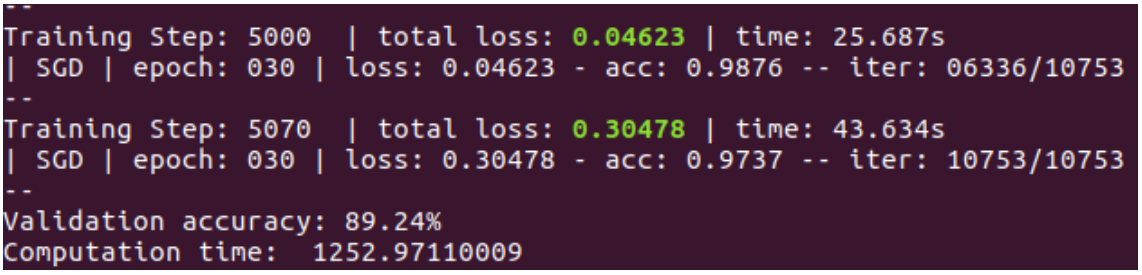

FIGURE 14 – 4ème validation : Score de validation : 89.24%

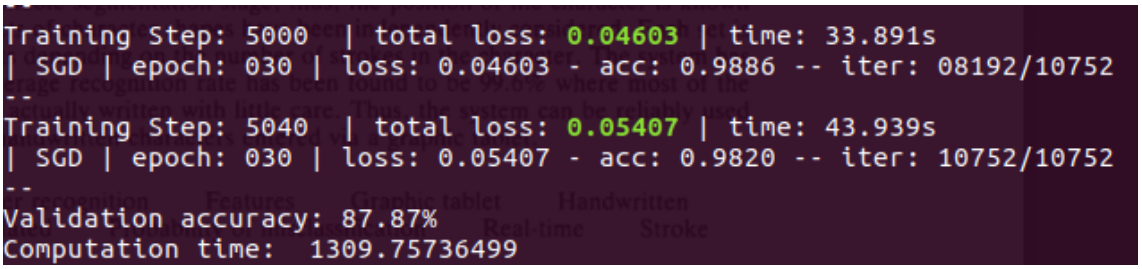

FIGURE 15 – 5ème validation : Score de validation : 87.87%

Nous obtenons à présent un score total de 84,434%. Cette configuration est donc moins efficace que la précédente. Nous avons émis l'hypothèse que cela était dû à la duplication de features et que si nous voulions obtenir de meilleures performances, il nous faudrait plus de données d'apprentissage. Par ailleurs, on met, ici, en évidence que le temps de calcul est fortement affecté par le nombre de filtre (300 ms de plus environ). Observons maintenant si l'augmentation du filtre de la première convolution a un effet bénéfique sur le modèle.

6.2.3 Cross validation 3 :

- Input : 32*32
- CL1 : 110 **5*5** filtres
- Max Pooling : 2*2
- CL2 : 64 3*3 filtres
- Max Pooling : 2*2
- fully connected : 1024 neurons
- dropout : 0.8
- fully connected : 512 neurons
- dropout : 0.8
- fully connected : 28 neurons
- optimisation : Stochastic Gradient Descent

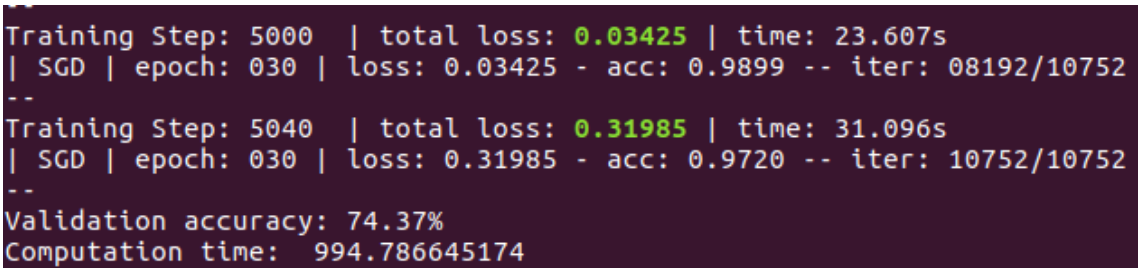

FIGURE 16 – 1ére validation : Score de validation : 74.37%

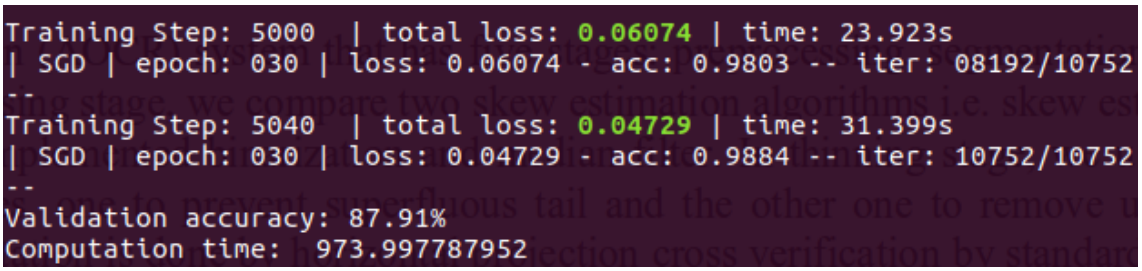

FIGURE 17 – 2ème validation : Score de validation : 87.91%

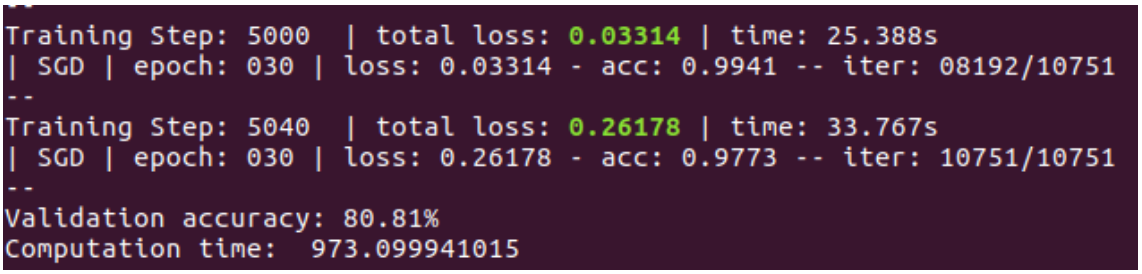

FIGURE 18 – 3ème validation : Score de validation : 80.81%

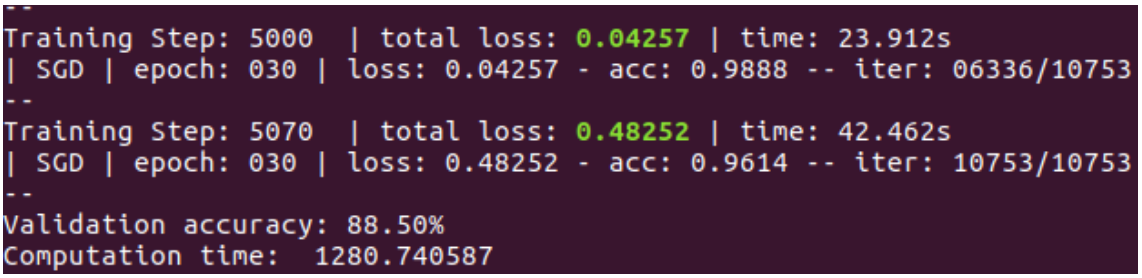

FIGURE 19 – 4ème validation : Score de validation : 88.50%

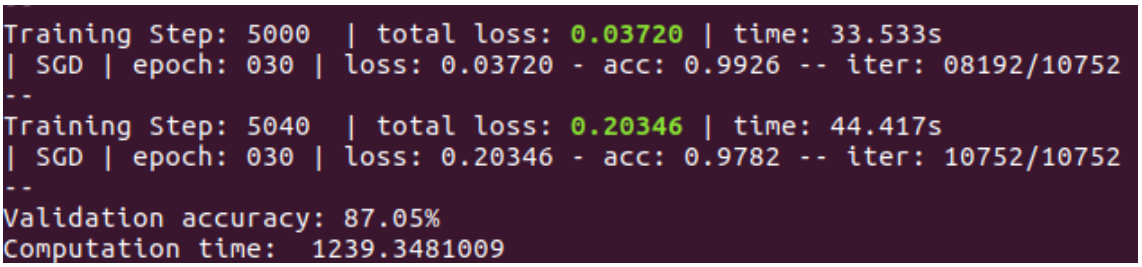

FIGURE 20 – 5ème validation : Score de validation : 87.05%

La cross validation a atteint **85.236%** en moyenne, notre architecture de base (avec filtres

3*3) reste donc plus efficace. Dans la cross validation suivante, nous ajoutons une couche de convolution à notre système ainsi que du max pooling.

6.2.4 Cross validation 4 :

- Input : 32*32
- CL1 : 80 3*3 filtres
- Max Pooling : 2*2
- **CL2 : 64 3*3 filtres**
- **Max Pooling : 2*2**
- fully connected : 1024 neurons
- dropout : 0.8
- fully connected : 512 neurons
- dropout : 0.8
- fully connected : 28 neurons
- optimisation : Stochastic Gradient Descent

```
--
Training Step: 5000  | total loss: 0.03722 | time: 27.668s
| SGD | epoch: 030 | loss: 0.03722 - acc: 0.9909 -- iter: 08192/10752
--
Training Step: 5040  | total loss: 0.52653 | time: 36.297s
| SGD | epoch: 030 | loss: 0.52653 - acc: 0.9652 -- iter: 10752/10752
--
Validation accuracy: 79.13%
Computation time:  1070.88827801
```

FIGURE 21 – 1ére validation : Score de validation : 79.13%

```
--
Training Step: 5000  | total loss: 0.05635 | time: 26.775s
| SGD | epoch: 030 | loss: 0.05635 - acc: 0.9872 -- iter: 08192/10752
--
Training Step: 5040  | total loss: 0.05815 | time: 35.415s
| SGD | epoch: 030 | loss: 0.05815 - acc: 0.9799 -- iter: 10752/10752
--
Validation accuracy: 89.62%
Computation time:  1041.54037499
```

FIGURE 22 – 2ème validation : Score de validation : 89.62%

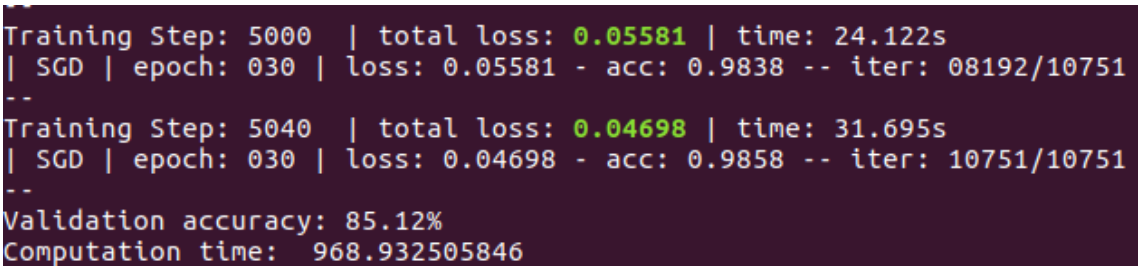

FIGURE 23 – 3ème validation : Score de validation : 85.12%

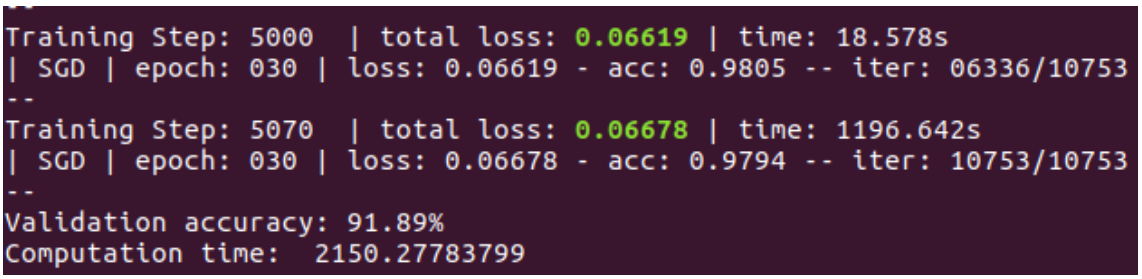

FIGURE 24 – 4ème validation : Score de validation : 91.89%

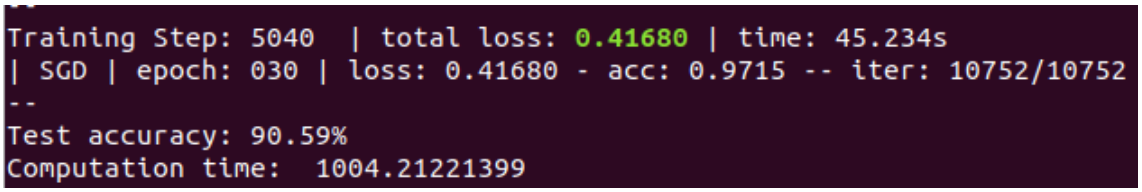

FIGURE 25 – 5ème validation : Score de validation : 90.59%

Nous obtenons donc 87.27% de précision en moyenne. On en conclut qu'ajouter une couche de convolution améliore les performances du système d'environ 3%. En effet, notre réseau effectuant plus de traitements, il propose une classification plus efficace. Nous pouvons cependant noter que cela augmente le temps de calcul.Tentons maintenant d'augmenter la taille du filtre sur ce nouveau réseau.

6.2.5 Cross validation 5 :

- Input : 32*32
- CL1 : 80 **5*5** filtres
- Max Pooling : 2*2
- CL2 : 64 3*3 filtres
- Max Pooling : 2*2
- **CL3 : 64 3*3 filtres**
- **Max Pooling : 2*2**
- fully connected : 1024 neurons

— dropout : 0.8
— fully connected : 512 neurons
— dropout : 0.8
— fully connected : 28 neurons
— optimisation : Stochastic Gradient Descent

```
--
Training Step: 5000  | total loss: 0.03722 | time: 27.668s
| SGD | epoch: 030 | loss: 0.03722 - acc: 0.9909 -- iter: 08192/10752
--
Training Step: 5040  | total loss: 0.52653 | time: 36.297s
| SGD | epoch: 030 | loss: 0.52653 - acc: 0.9652 -- iter: 10752/10752
--
Validation accuracy: 79.13%
Computation time:  1070.88827801
```

FIGURE 26 – 1ére validation : Score de validation : 79.13%

```
--
Training Step: 5000  | total loss: 0.05635 | time: 26.775s
| SGD | epoch: 030 | loss: 0.05635 - acc: 0.9872 -- iter: 08192/10752
--
Training Step: 5040  | total loss: 0.05815 | time: 35.415s
| SGD | epoch: 030 | loss: 0.05815 - acc: 0.9799 -- iter: 10752/10752
--
Validation accuracy: 89.62%
Computation time:  1041.54037499
```

FIGURE 27 – 2ème validation : Score de validation : 89.62%

```
--
Training Step: 5000  | total loss: 0.05581 | time: 24.122s
| SGD | epoch: 030 | loss: 0.05581 - acc: 0.9838 -- iter: 08192/10751
--
Training Step: 5040  | total loss: 0.04698 | time: 31.695s
| SGD | epoch: 030 | loss: 0.04698 - acc: 0.9858 -- iter: 10751/10751
--
Validation accuracy: 85.12%
Computation time:  968.932505846
```

FIGURE 28 – 3ème validation : Score de validation : 85.12%

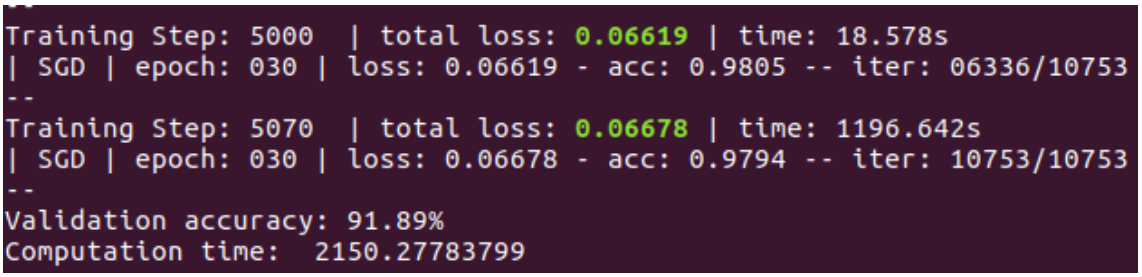

FIGURE 29 – 4ème validation : Score de validation : 91.89%

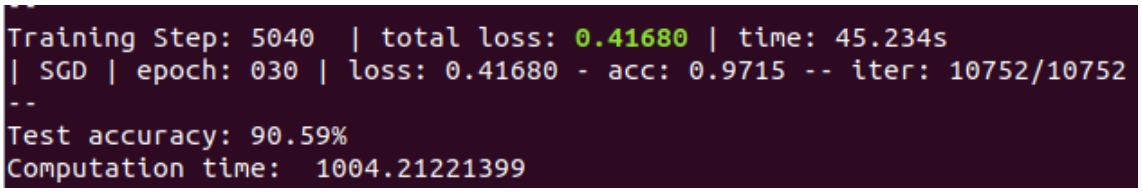

FIGURE 30 – 5ème validation : Score de validation : 90.59%

Le résultat moyen de **86.556%** indique que l'augmentation de la taille du filtre et l'ajout d'une couche de convolution n'améliore pas les performances. Nous allons ensuite changer la fonction d'activation de Relu à sigmoid.

6.2.6 Cross validation 6 :

- Input : 32*32
- CL1 : 80 5*5 filtres - **activation sigmoid**
- Max Pooling : 2*2
- CL2 : 64 3*3 filtres - **activation sigmoid**
- Max Pooling : 2*2
- CL3 : 64 3*3 filtres - **activation sigmoid**
- Max Pooling : 2*2
- fully connected : 1024 neurons - **activation sigmoid**
- dropout : 0.8
- fully connected : 512 neurons - **activation sigmoid**
- dropout : 0.8
- fully connected : 28 neurons
- optimisation : Stochastic Gradient Descent

FIGURE 31 – 1ére validation : Score de validation : 3.57%

On peut voir dés la première validation que la fonction d'activation Sigmoid n'est pas du tout adaptée à notre système. Passons donc à l'essai suivant, ou l'on va étudier l'impact de la fonction d'optimisation Momentum.

6.2.7 Cross validation 7 :

- Input : 32*32
- CL1 : 80 5*5 filtres
- Max Pooling : 2*2
- CL2 : 64 3*3 filtres
- Max Pooling : 2*2
- CL3 : 64 3*3 filtres
- Max Pooling : 2*2
- fully connected : 1024 neurons
- dropout : 0.8
- fully connected : 512 neurons
- dropout : 0.8
- fully connected : 28 neurons
- optimisation : **momentum**

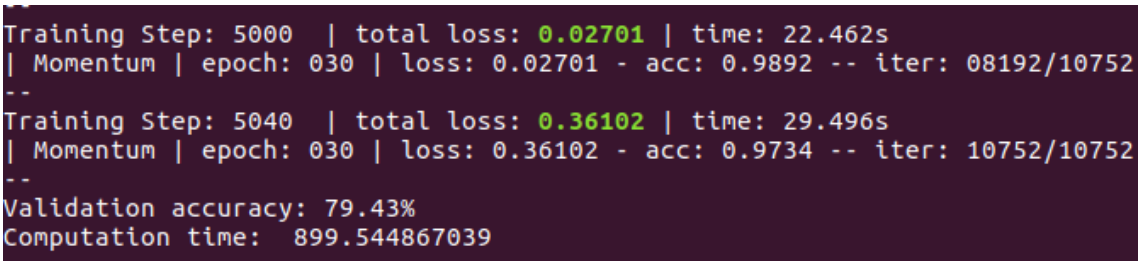

FIGURE 32 – 1ére validation : Score de validation : 79.43%

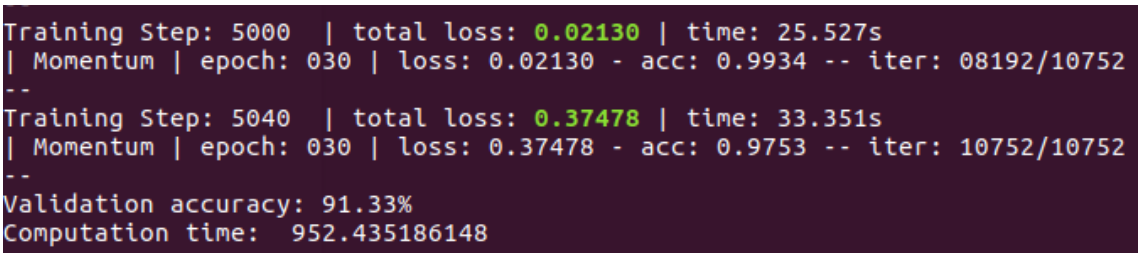

FIGURE 33 – 2ème validation : Score de validation : 91.33%

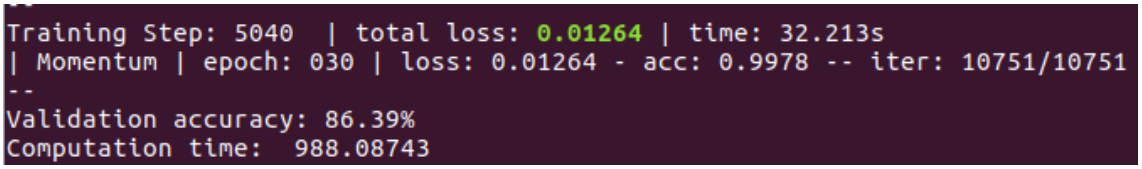

FIGURE 34 – 3ème validation : Score de validation : 86.39%

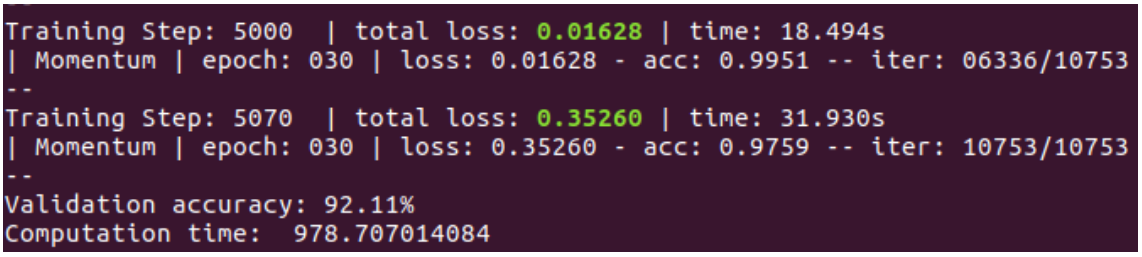

FIGURE 35 – 4ème validation : Score de validation : 92.11%

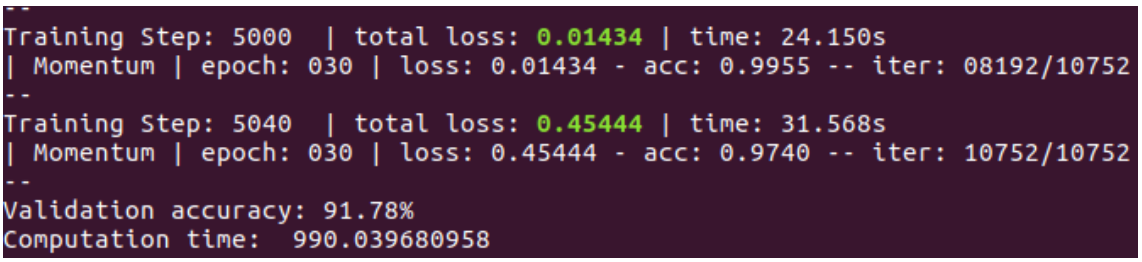

FIGURE 36 – 5ème validation : Score de validation : 91.78%

Le score global obtenu est **88.208%**. Nous remarquons donc que nous avons amélioré une nouvelle fois la précision de notre réseau de 0.7%. Étudions maintenant l'influence des multiples couches fully connected en en retirant.

6.2.8 Cross validation 8 :

— Input : 32*32

— CL1 : 80 5*5 filtres
— Max Pooling : 2*2
— CL2 : 64 3*3 filtres
— Max Pooling : 2*2
— CL3 : 64 3*3 filtres
— Max Pooling : 2*2
— **fully connected : 28 neurons**
— optimisation : momentum

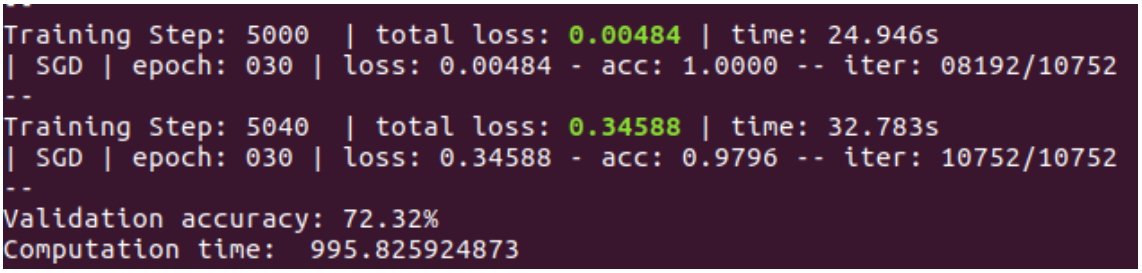

FIGURE 37 – 1ére validation : Score de validation : 72.32%

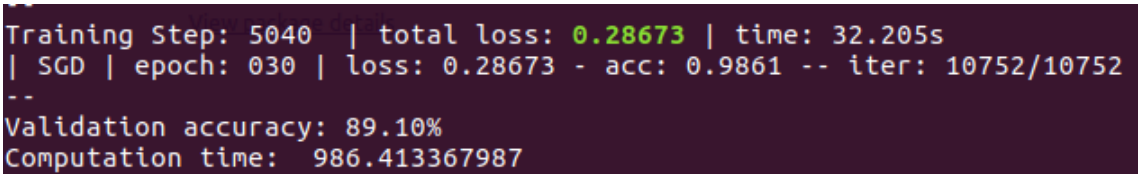

FIGURE 38 – 2ème validation : Score de validation : 89.10%

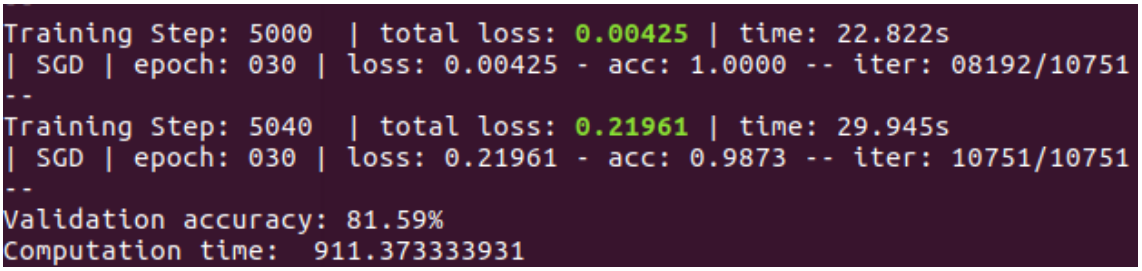

FIGURE 39 – 3ème validation : Score de validation : 81.59%

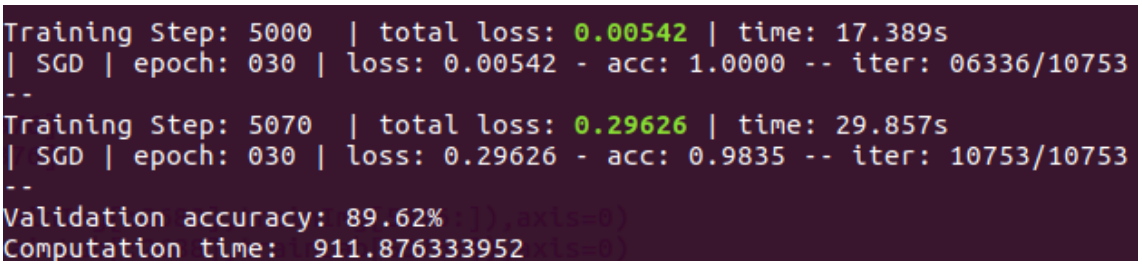

FIGURE 40 – 4ème validation : Score de validation : 89.62%

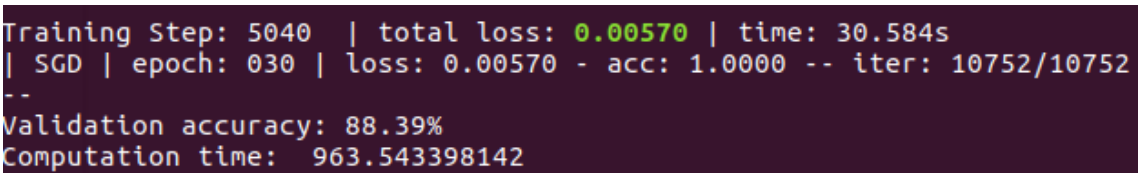

FIGURE 41 – 5ème validation : Score de validation : 88.39%

Cette cross validation retourne **84.204%** de bonne classification. On voit donc bien que retirer les couches fully connected rend le système moins précis, sans pour autant diminuer significativement le temps de calcul, nous allons donc les garder dans notre architecture. En effet, l'ajout de 2 couches de neurones permet d'affiner la reconnaissance du modèle.

Après ces différents "tuning" d'hyperparamétres, nous pouvons enfin passer au test de notre CNN ainsi validé.

6.3 Test

Nous avons maintenant suffisamment de données pour pouvoir conclure sur le choix de l'architecture de CNN suivante :

- Input : 32*32
- CL1 : 80 5*5 filtres
- Max Pooling : 2*2
- CL2 : 64 3*3 filtres
- Max Pooling : 2*2
- CL3 : 64 3*3 filtres
- Max Pooling : 2*2
- fully connected : 1024 neurons
- dropout : 0.8
- fully connected : 512 neurons
- dropout : 0.8
- fully connected : 28 neurons
- optimisation : momentum

L'étape finale consistera au test de ce modèle avec le set de 3360 images de test inédites pour notre CNN :

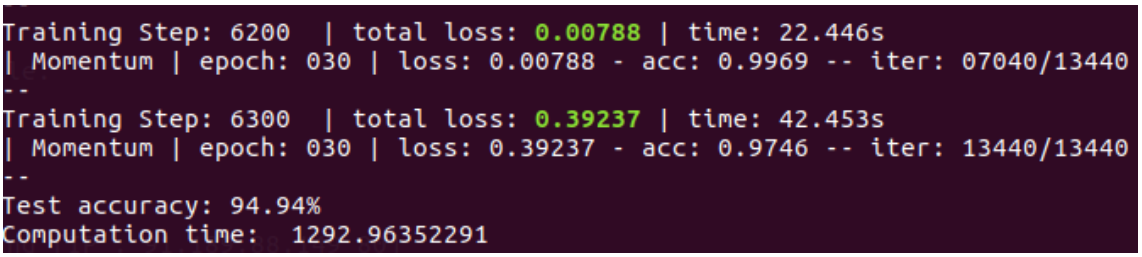

FIGURE 42 – Score de Test : Score de validation : **94.94%**

6.4 Conclusion

On déduit de nos expérimentations que le CNN est un type de réseau plus performant que le MLP pour la détection de caractères arabes avec une différence de **13,52%**.

Ce premier projet de deep learning nous a finalement permis de mobiliser un ensemble de connaissances fondamentales de l'apprentissage profond, et nous a permis de comparer deux approches différentes au problème de reconnaissance de lettres arabes. Cependant, ce sont deux méthodes parmi d'autres, et on pourrait éventuellement penser à utiliser le principe de transfer learning à partir d'un modèle détectant des chiffres. En effet, il est possible que certains chiffres et certains caractères aient des points communs entre eux, et on pourrait reconnaître des lettres arabes en faisant du fine tuning sur le réseau entrainé à reconnaitre des chiffres.

6.5 Références

— figure 5 : https ://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks
— CNN for Handwritten Arabic Digits Recognition based on LeNet-5 by Ahmed El-Sawy, Hazem El-Bakry, Mohamed Loey
— https ://www.kaggle.com/mloey1/ahcd1
— https ://www.sciencedirect.com/science/article/pii/S2212017313003538
— http ://neuralnetworksanddeeplearning.com/chap1.html