MSE | MASTER OF SCIENCE IN ENGINEERING

TSM – Machine Learning

PW 11: Convolutional Neural Networks

4 December 2023

Daniel Ribeiro Cabral – Bastien Veuthey

MSE | Data Science and Computer Science

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

1. Digit recognition from raw data

Le but de ce notebook est d'utiliser les données *raw* du dataset MNIST et d'entraîner plusieurs modèles dits *shallow neural network* (réseau neuronaux peu profond). Pour cela, il faut effectuer de la sélection de modèles MLP en essayant différentes compositions et hyper paramètres. Pour chaque modèle décrit dans ce rapport, plusieurs tests sont effectués et le meilleur résultat est retourné.

Le dataset est séparé en trois sets différents : train (50'000 samples), validation et test (10'000 samples chacun). Il est précisé qu'il ne faut utiliser le test set seulement lorsque le modèle final aura été sélectionné grâce aux différents essais et approuvé grâce au validation set.

Le modèle de départ (**Sequential** pour un simple shallow neural) consiste en :

- **Epochs :** 10
- **Bacth-Size :** 128
- **Learning-Rate :** 0,001
- **Hidden neuron :** 300
- **Optimizer :** RMSprop
- **Validation set accuracy :** 97,7 %

Avec cette précision initiale élevée, notre point de départ (ou baseline) est déjà de haute qualité. Notre objectif est d'expérimenter avec différents modèles pour en tirer des conclusions concrètes. Malgré la haute performance de base, nous avons décidé de décrire trois modèles et de présenter le meilleur que nous avons trouvé.

Premier Modèle : Augmentation des Epochs

Avant de toucher au nombre d'hidden neurons, il est intéressant d'explorer les nombres d'epochs et d'observer l'impact sur les résultats. Ici, nous avons décidé d'augmenter les epochs à 100 (soit 10x celui du baseline).

- **Précision sur le validation set :** 98,18 %

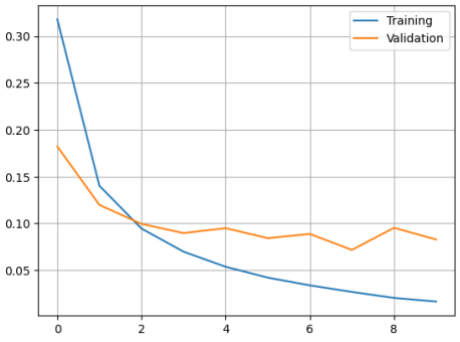


Figure 1 : Raw – baseline

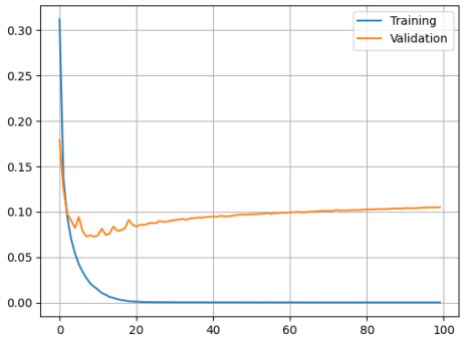


Figure 2 : Raw - premier modèle

La précision a augmenté (moins d'un pourcent) rendant le premier modèle plus performant. La convergence est beaucoup plus visible et palpable. Le nombre d'epochs peut être choisi vers 60 ainsi.

Deuxième Modèle : Modification de l'Architecture avec Dropout

Notre deuxième modèle a intégré une couche de dropout afin d'améliorer la généralisation et de prévenir le overfitting. Le dropout (0.5) désactive aléatoirement des neurones pendant l'entraînement, ce qui force le réseau à apprendre des caractéristiques plus robustes qui ne dépendent pas d'un petit nombre de neurones.

- **Précision sur le validation set :** 98,36 %

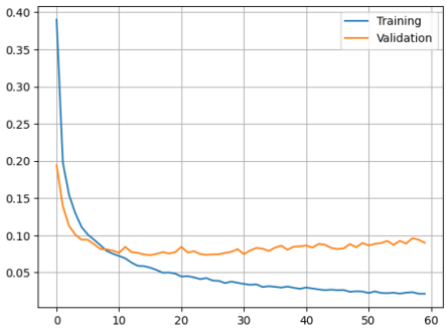


Figure 3 : Raw - deuxième modèle

Troisième modèle : hidden neuron

Le but ici est de modifier le nombre d'hidden neuron de la première couche du modèle et d'établir une dernière observation avant de désigner le modèle final.

- **Précision sur le validation set :** 98,58 %

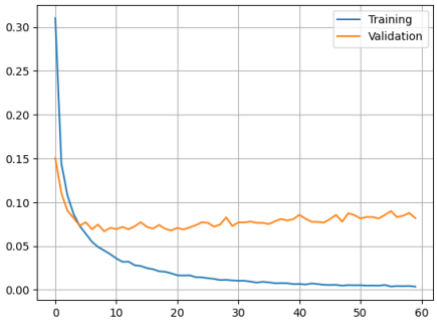


Figure 4 : Raw - troisième modèle

L'évolution de la précision est augmentée d'environ 0.2% avec un nombre de neurones cachés à 1000.

Modèle Final : Réponses aux questions

Toutes les questions répondus ci-dessous sont basés sur le modèle final trouvé et décrit à la fin du troisième modèle.

1. What is the learning algorithm being used to train the neural networks?

L'algorithme utilisé est RMSprop (Root Mean Square Propagation). Il s'agit d'un algorithme d'optimisation du taux d'apprentissage adaptatif et est une extension de la descente de gradient. Il ajuste le taux d'apprentissage pour chaque poids du modèle individuellement, en fonction des amplitudes récentes des gradients pour ce poids. Cela permet d'atténuer les problèmes de disparition et d'explosion des gradients qui peuvent survenir avec la descente de gradient de base.

2. What are the parameters (arguments) being used by that algorithm?

L'optimiseur RMSprop est utilisé avec ces valeurs par défaut (learning rate à 0.001 et momentum à 0 par exemple). La taille du batch pour l'entraînement est de 128, et le modèle est entraîné pendant 60 epochs. Il utilise une couche dense ReLU de 1000 neurones.

3. Model Complexity: Select a neural network and describe the topology, describe the inputs and output, compute the number of weights and explain how do you get the number of weights.

- **Topologie** : Le modèle est un simple MLP (feed-forward network) avec une couche cachée dense ReLU et une couche output qui utilise la fonction d'activation softmax.
- **Input** : Il consiste de 784 neurones (28x28 pixels images reconduits en vecteurs 1D).
- **Output** : La sortie est une couche dense avec **n_classes** unités, correspondant au nombre de classes pour la classification de notre dataset.
- **Nombres de weights** : Le nombre de poids dans ce modèle peut être calculé comme suit :
  - Dans la couche cachée, chacun des 1000 neurones est connecté à chacun des 784 neurones d'entrée. Il y a donc 784 × 1000 = 784'000 poids. En outre, il y a 1000 termes de biais (un pour chaque neurone), soit un total de 784'000 + 1000 = 785'000 paramètres dans la couche cachée.
  - Dans la couche de sortie, chacun des neurones (10) est connecté à chacun des 1000 neurones de la couche cachée. Il y a donc 1000 × 10 de poids plus les termes de biais pour chaque classe, cela donne un total de 1000 × 10 + 10 paramètres dans la couche de sortie.

Le nombre total de paramètres dans le réseau est donc de 785'000 + 1000 × 10 + 10 = **795'010**. La couche de dropout n'ajoute aucun poids au modèle.

4. Final Results

Ces résultats sont ceux du troisième modèle (regroupant tous les ajustements aux niveaux des paramètres) avec le test set enfin exécuté et prédit.

- **Epochs:** 60
- **Optimizer:** RMSprop
- **Cost function:** categorical crossentropy
- **Batch size:** 128

Cela donne une précision de **98,44%** et un score de 0.084.

```
array([[ 975,    1,    1,    0,    0,    0,    1,    1,    1,    0],
       [   0, 1127,    3,    1,    0,    0,    1,    1,    2,    0],
       [   2,    1, 1017,    3,    1,    0,    0,    5,    3,    0],
       [   1,    0,    3,  995,    0,    4,    0,    2,    3,    2],
       [   0,    0,    3,    1,  962,    0,    3,    2,    1,   10],
       [   2,    0,    0,    9,    1,  874,    1,    1,    3,    1],
       [   6,    2,    0,    1,    2,    7,  940,    0,    0,    0],
       [   1,    1,    6,    1,    0,    0,    0, 1013,    2,    4],
       [   3,    1,    2,    5,    2,    1,    1,    3,  953,    3],
       [   2,    3,    0,    5,    7,    0,    0,    3,    1,  988]])
```

Figure 5 : Raw - matrice confusion

La diagonale est visible et transcrit une bonne performance du modèle qui n'a pas de problèmes a trouvé les bons digits. Il y a quelques erreurs mais jamais plus de 10. La plus grosse confusion semble se faire entre « 9 » et « 4 » + « 3 » et « 5 » sans pour autant paraître élévé.

2. Digit recognition from features of the input data

Comme le notebook précédent, le dataset utilisé est MNIST (digit entre 0 et 9). Cependant, cette fois la méthode HOG feature est appliquée pour reconnaître les features des images plutôt que les inputs raw (28x28) directement.

Les mêmes sets et leur répartition sont similaires à l'exercice 1.

HOG (Hi

Le modèle de départ (**Sequential** pour un simple shallow neural) consiste en :

- **Orientations** : 8
- **Pixels per cell** : 4
- **Epochs :** 5
- **Bacth-Size :** 128
- **Learning-Rate :** 0,001
- **Hidden neuron :** 300
- **Optimizer :** RMSprop
- **Validation set accuracy :** 97,36 %

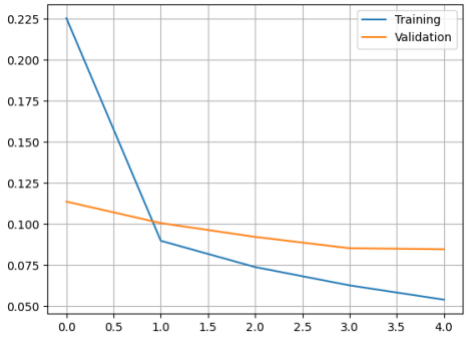


Figure 6 : HOG - baseline

Cette baseline a une précision élevée mais le graph nous indique qu'il est possible de faire mieux. Il serait au moins intéressant d'essayer d'atteindre une convergence, ce qui n'est pas le cas avec 5 epochs.

Premier Modèle : Augmentation des Epochs

De la même manière que le notebook précédent, l'augmentation devrait permettre d'améliorer les performances du modèle.

- **Précision sur le validation set :** 97,99 %

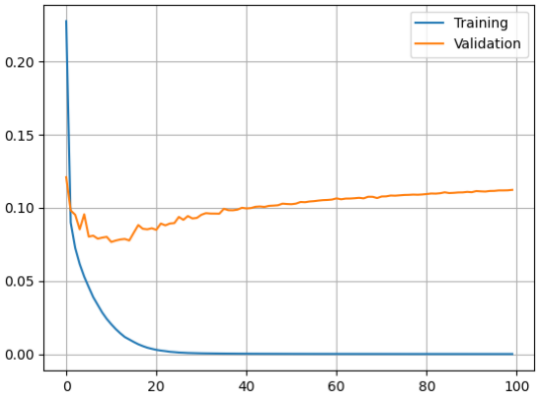


Figure 7 : HOG - premier modèle

Pour un nombre d'epoch à 100, la convergence est tout de suite plus visible par rapport au modèle précédent (aux alentours de 80). Nous nous apercevons cependant que la courbe du validation set a tendance a continuellement accroître ce qui pourrait laisser présager à de l'overfitting.

Deuxième Modèle : Modification de l'Architecture avec Dropout

Les solutions possibles seraient d'arrêter l'évaluation beaucoup plus tôt (15 epoch par exemple) mais cela nous donne la même précision que le premier modèle (97.99%). L'autre serait d'utiliser du dropout (0.5).

- **Précision sur le validation set :** 98,18 %

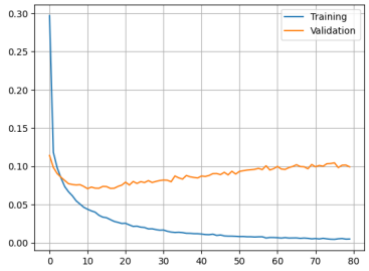


Figure 8 : HOG - deuxième modèle

La courbe du validation set se stabilise mieux vers le 0.1 qu'auparavant.

Troisième Modèle : Manipulation des métriques HOG

Le calcul du HOG est fortement dépendant de deux métriques : le nombre de directions de gradients et le nombre de pixels par cellule. La direction (en degrés ou radians) est l'angle à laquelle une intensité de pixels change le plus rapidement. Avec une direction par défaut de 5, cela implique un « bin » de 5 angles dont chaque pixel doit pouvoir être classé.

Le but pour ce modèle est de trouver une combinaison de ces deux paramètres qui améliorent la précision par défaut. À savoir que le détail de la description HOG est dépendant de ces nombres, plus il y a d'orientations et moins de pixels par cellules, plus elle sera détaillée.

- **Précision sur le validation set :** 98,36 %

Pour les valeurs de *n_orientations* à 8 et *pix_p_cell* à 2, la précision s'est encore améliorée.

Quatrième modèle : hidden neuron

Avec 1000 neurones cachés, la précision augmente à :

- **Précision sur le validation set :** 98,5 %

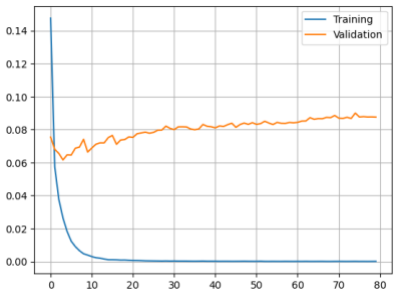


Figure 9 : HOG - quatrième modèle

Modèle Final : Réponses aux questions

Toutes les questions répondus ci-dessous sont basés sur le modèle final trouvé et décrit à la fin du quatrième modèle.

5. What is the learning algorithm being used to train the neural networks?

Il s'agit du même que pour le modèle des données raw de MNIST, RMSprop.

6. What are the parameters (arguments) being used by that algorithm?

La différence comparée au modèle précédent est le nombre d'epochs légérement supérieurs, à 80, et les nouveaux paramètres qui sont propres aux méthodes de features descriptor comme HOG avec le nombre de direction à 10 et le nombre de pixels par cellules à 2.

7. Model Complexity: Select a neural network and describe the topology, describe the inputs and output, compute the number of weights and explain how do you get the number of weights.

- **Topologie** : Le modèle est un simple MLP (feed-forward network) avec une couche cachée dense ReLU et une couche output qui utilise la fonction d'activation softmax.
- **Input** : Il consiste à la taille du vecteur HOG (hog_size) qui est de 1568.
- **Output** : La sortie est une couche dense avec **n_classes** unités, correspondant au nombre de classes pour la classification de notre dataset.
- **Nombres de weights** : Le calcul se fait exactement de la même manière que le modèle des raw inputs avec à la place des 784 inputs, 1568. Le résultat est donné par le calcul suivant : 1568 × 1000 + 1000 + 1000 × 10 + 10 = **1'579'010** paramètres.

8. Final Results

Ces résultats sont ceux du troisième modèle (regroupant tous les ajustements aux niveaux des paramètres) avec le test set enfin exécuté et prédit.

- **Orientations:** 8
- **Pixels per cell:** 2
- **Epochs:** 80
- **Optimizer:** RMSprop
- **Cost function:** categorical crossentropy
- **Batch size:** 128

Cela donne une précision de **98,58%** et un score de 0.069.

```
array([[ 974,    0,    0,    0,    0,    0,    4,    1,    0,    1],
       [   2, 1129,    1,    1,    0,    1,    1,    0,    0,    0],
       [   3,    2, 1018,    0,    1,    0,    2,    4,    2,    0],
       [   0,    0,    0, 1000,    0,    5,    0,    2,    2,    1],
       [   1,    2,    1,    0,  966,    0,    1,    0,    2,    9],
       [   1,    0,    0,    7,    0,  881,    1,    0,    2,    0],
       [   5,    3,    0,    0,    3,    3,  942,    0,    2,    0],
       [   1,    1,    4,    0,    2,    0,    0, 1013,    2,    5],
       [   4,    1,    5,    3,    1,    0,    1,    3,  952,    4],
       [   3,    4,    1,    2,    4,    4,    0,    7,    0,  984]])
```

Figure 10 : HOG - matrice confusion

3. Convolutional neural network digit recognition

Dans cette section, nous abordons la reconnaissance de chiffres à l'aide des CNN comme vue dans la théorie. Trois modèles ont été sélectionnés, conformément aux directives de l'exercice. Les notebooks fournis comportaient une configuration initiale des hyperparamètres pour l'entraînement sur le dataset MNIST, que nous avons utilisée comme point de départ. Les détails et performances de cette configuration de base sont les suivants :

- **Epochs :** 5
- **Bacth-Size :** 128
- **Learning-Rate :** 0,001
- **Optimizer :** RMSprop
- **Test set accuracy :** 97,8 %

Premier Modèle : Augmentation des Epochs

Notre première modification a été d'augmenter le nombre d'epochs (200). Cette approche a pour objectif d'améliorer la précision de notre système, bien que cela entraîne un temps d'entraînement plus long. Cependant, une amélioration, même mineure, de la précision peut être significative. Les résultats obtenus sont :

- **Précision sur le test set :** 98,53 %

Cette augmentation notable de la précision est particulièrement impressionnante compte tenu de la haute baseline.

Deuxième Modèle : Modification de l'Architecture avec Dropout

Résultats finaux du deuxième modèle :

- **Précision sur le test set :** 98.59%

Le tableau suivant présente la matrice de confusion sur le jeu de test, illustrant la performance du modèle sur chaque classe de chiffres du dataset MNIST :

```
array([[ 971,    0,    4,    1,    0,    1,    3,    0,    0,    0],
       [   0, 1126,    1,    3,    0,    2,    0,    2,    1,    0],
       [   0,    0, 1029,    1,    0,    0,    0,    1,    1,    0],
       [   0,    0,    1, 1000,    0,    4,    0,    3,    1,    1],
       [   0,    1,    0,    0,  966,    0,    4,    0,    0,   11],
       [   0,    0,    0,    9,    0,  881,    1,    0,    0,    1],
       [   6,    1,    0,    0,    3,    4,  942,    0,    2,    0],
       [   0,    2,    9,    1,    0,    0,    0, 1015,    0,    1],
       [   2,    0,    4,    7,    1,    5,    4,    2,  942,    7],
       [   1,    1,    1,    1,    5,    3,    0,    9,    1,  987]])
```

Figure 11 : CNN - deuxième modèle

Cette matrice montre que le modèle avec dropout maintient une haute précision, avec peu de confusions entre les classes. Les erreurs les plus significatives semblent être entre les classes 2 et 7, et les chiffres 4 et 9, qui sont des erreurs communes dues à la similitude visuelle entre ces chiffres.

Troisième modèle : exploration et modification de l'architecture

Pour le troisième modèle, nous avions comme but de tester un peu tout ce qui était possible afin de voir ce qui était possible avec la librairie tensorflow. Nous avons donc décider d'explorer 3 options :

- **Ajuster les hyperparamètres :** augmenter le nombres filtres, dropouts, ou même changer la taille du noyau
- **Ajouter des techniques de normalisations :** data augmentation, normaliser les données….
- **Potentiellement changer la cost function :** changer l'optimizer, learning rate, cost function…

Voici les différents tests effectués pour l'option **« ajuster les hyperparamètres »** et les résultats obtenus :

Test	Descriptions	Précision sur le test-set[1]
A	Augmentation du nombre de filtres (9 à 32)	98.94%
B	Diminution du nombre de filtres (9 à 6)	98.46%
C	Changement de la Taille de Noyau à 3x3	98.71%
D	Changement de la Taille de Noyau à 7x7	98.97%
E	Réduction du Taux de Dropout à 0.3	98.93%
F	Augmentation du Taux de Dropout à 0.7	98.18%
G	**Mélange des meilleurs paramètres (32 filtres + noyau a 7x7 + dropout à 0.3 + 256 epochs)**	**99.13%**

Ces résultats démontrent clairement que chaque modification des paramètres impacte la précision du modèle. Le Test G, combinant les paramètres les plus efficaces, a montré une amélioration significative de la performance.

Des tests supplémentaires ont été effectués en se basant sur les paramètres du Test G pour évaluer l'impact **des techniques de normalisation** :

Test	Descriptions	Précision sur le test - set
A	Régularisation L1/L2	98.82%
B	Data augmentation	Failed
C	Batch Normalization	99.07%

Ces tests n'ont pas montré d'amélioration significative de la précision par rapport aux résultats obtenus précédemment.

Nous avons également évalué l'impact du choix de l'optimiseur et de la fonction de coût :

[1] **Note importante :** chaque test a été contrôler avec le *validation* set avant de fournir le résultat du test set dans le rapport.

Test	Descriptions	Précision sur le test - set
A	Optimizer : Adam	98.98%
B	Optimizer : Adagrad	97.11%
C	**Optimizer : SGD**	**99.14%**
D	Optimizer : Adamax	98.66%

Comme on peut le voir, le choix de l'optimiseur est crucial et doit être adapté au type de données et au problème spécifique. Dans nos expériences, l'optimiseur SGD s'est révélé être le plus efficace, offrant la meilleure précision sur le test-set. Il est important de noter que ces résultats peuvent varier en fonction de la nature des données et des spécificités du modèle utilisé. Dans notre, cas il se peut qu'en relançant le code cela ne montre pas le même résultat tout le temps. Malgré ça, le résultat de la précision tourne toujours autour de ces mêmes valeurs avec 0.15 en moins ou en plus.

Modèle Final : Réponses aux questions

Toutes les questions répondus ci-dessous sont basés sur le modèle final trouvé. Une description y sera faite afin de bien comprendre quels paramètres y ont été utilisés.

9. What is the learning algorithm being used to train the neural networks?

Le réseau est entraîné en utilisant une architecture CNN avec plusieurs couches convolutives et de max-pooling, suivies de couches entièrement connectées. L'algorithme d'optimisation utilisé pour l'entraînement est le SGD.

10. What are the parameters (arguments) being used by that algorithm?

L'optimiseur SGD est utilisé avec un momentum de 0.9. La taille du batch pour l'entraînement est de 256, et le modèle est entraîné pendant 30 epochs. Il faut savoir que nous avons plusieurs autres paramètres comme les filtres, noyau etc… et même la longueur de l'architecture peut être complétement changée.

11. Model Complexity: Select a neural network and describe the topology, describe the inputs and output, compute the number of weights and explain how do you get the number of weights

- **Topologie** : Le modèle consiste en trois couches convolutives avec max pooling suivies d'une couche flatten et de deux couches denses. La dernière couche dense utilise une fonction d'activation softmax pour la classification multiclasses (0 à 9 pour le MNIST).
- **Input** : L'input est un tenseur de forme (w, h, 1), ce qui dit que ce sont des images de niveaux de gris avec une certaine hauteur et largeur. Donc notre cas, c'est du 28x28. On a donc l'obligation de faire cela.
- **Output** : La sortie est une couche dense avec **n_classes** unités, correspondant au nombre de classes pour la classification de notre dataset.
- **Nombres de weights** : Pour calculer le nombre de poids, nous devons prendre en compte les poids dans les couches Conv2D (filtres et biais) et les poids dans les couches denses (y compris les biais). Cela peut être calculé en additionnant le produit des dimensions des poids de chaque couche et en ajoutant le nombre de biais pour chaque couche (voir théorie).
    - **Couche l1** : $32 \times (7 \times 7 + 1) = 1600$

- **Couche l2** : 32 × (7 × 7 × 32 + 1) = 50208
- **Couche l3** : 16 × (3 × 3 × 32 + 1) = 4624
- **Couche l4** : 144 × 25 + 25=3625
- **Couche l5** : 25 × 10 + 10=260
- **Total** : 1600 + 50208 + 4624 + 3625 + 260 = 60317

12. Final Results

- **Epochs:** 30
- **Optimizer:** SGD (momentum:0.9)
- **Cost function:** categorical crossentropy
- **Batch size:** 256

Le score de test est de 0.0365, et la précision du test set est d'environ 99.14%.

```
array([[ 973,    0,    3,    0,    0,    2,    0,    1,    1,    0],
       [   0, 1129,    1,    2,    0,    0,    0,    3,    0,    0],
       [   0,    1, 1027,    2,    0,    0,    0,    2,    0,    0],
       [   0,    0,    1, 1006,    0,    2,    0,    1,    0,    0],
       [   0,    0,    2,    0,  975,    0,    1,    1,    0,    3],
       [   0,    0,    0,    5,    0,  885,    1,    1,    0,    0],
       [   6,    2,    1,    1,    1,    7,  940,    0,    0,    0],
       [   0,    2,    3,    1,    0,    0,    0, 1021,    0,    1],
       [   0,    1,    1,    2,    2,    3,    0,    2,  959,    4],
       [   1,    1,    0,    1,    3,    7,    0,    7,    1,  988]])
```

Figure 12 : CNN - matrice de confusion final

La matrice de confusion révèle une performance très bonne avec la majorité des prédictions correctes tombant sur la diagonale principale. Les erreurs de classification sont minimales, mais certaines confusions notables incluent la tendance du modèle à confondre le chiffre "6" avec "0" et "5", et dans une moindre mesure, "1" avec "2" et "7". Ces erreurs sont cependant peu nombreuses par rapport au nombre total de prédictions correctes, ce qui souligne l'efficacité globale du modèle dans la classification des chiffres manuscrits.

Le graphique démontre l'évolution de la loss function durant les 30 epochs de l'entrainement. On observe une décroissance rapide de la loss sur le train set au cours des premières epochs, qui se stabilise ensuite, indiquant que le modèle apprend efficacement les données

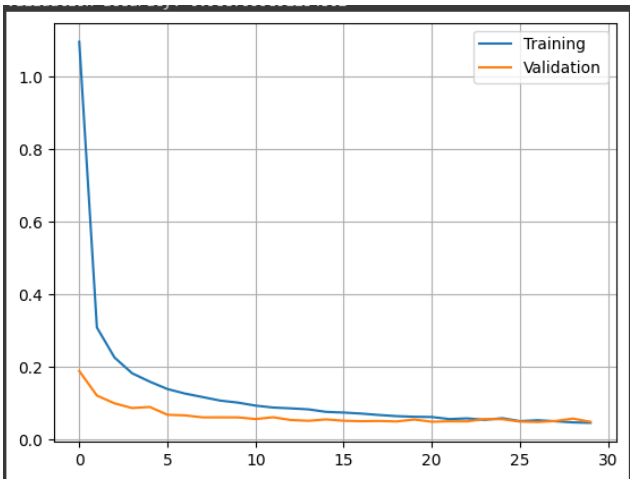


Figure 13 : CNN - modèle final

d'entraînement.

La loss de validation diminue également, mais elle se stabilise à une valeur légèrement supérieure à celle de l'entraînement. Cela indique que le modèle généralise bien sur des données non vues, un signe positif d'un bon équilibre de notre modèle et surtout sans overfitting.

Points communs

Il existe des points communs entre les trois notebooks de ce PW et dont les questions n'ont à être répondues qu'une seule fois.

1. What cost function is being used? please, give the equation(s)

La fonction de coût utilisée est « categorical_crossentropy ». Voici l'équation de cette cost function :

$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

Elle mesure la dissimilarité entre la distribution de probabilité prédite par le modèle et la distribution de probabilité réelle des labels. Plus la valeur de la *categorical crossentropy* est faible, plus les prédictions du modèle sont proches des vrais labels.

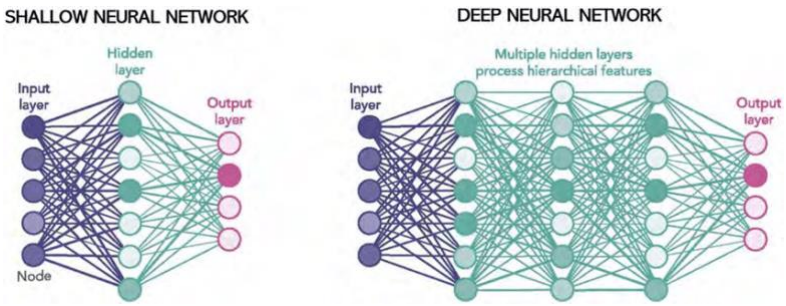
2. How did you create the training, validation and test datasets.

Pour cette question, nous n'avons pas dû faire cette étape car elle était déjà fournie dans les différents notebooks. Nous avons donc utilisé cette version-là.

3. Do the deep neural networks have much more "capacity" (i.e., do they have more weights?) than the shallow ones? Explain with one example.

Les deep neural networks ont généralement une plus grande capacité en raison du fait d'avoir plus de couches, ce qui leur permet de modéliser des fonctions plus complexes et de capturer des abstractions de plus haut niveau. Ils ont tendance à avoir plus de poids que les shallow neural networks.

Pour l'exemple cela est assez logique. Notre algorithme par exemple possède beaucoup de couches (11). Il y aura donc naturellement beaucoup de paramètres à la fin. Cependant, le but des shallow neural networks est d'avoir 1 ou très peu de couches donc elles auront généralement beaucoup de paramètres.



L'image ci-dessus démontre bien cet exemple.