Labo 4 - Learning with Artificial Neural Networks

Loïc Brasey Bastien Pillonel

May 9, 2023

Question 1:

What is the learning algorithm being used to optimize the weights of the neural networks? What are the parameters (arguments) being used by that algorithm? What cost function is being used ? please, give the equation(s)

L'algorithm utilisé est le RMSprop de la librairie Keras.

Les paramètres utilisés sont :

```
tf.keras.optimizers.RMSprop(
    learning_rate=0.001,
    rho=0.9,
    momentum=0.0,
    epsilon=1e-07,
    centered=False,
    weight_decay=None,
    clipnorm=None,
    clipvalue=None,
    global_clipnorm=None,
    use_ema=False,
    ema_momentum=0.99,
    ema_overwrite_frequency=100,
    jit_compile=True,
    name="RMSprop",
    **kwargs
)
```

L'equation utilisé est :

$v_{dw} = \beta * v_{dw} + (1 - \beta) * dw^2$
$v_{db} = \beta * v_{dw} + (1 - \beta) * db^2$
$W = W - a * \frac{dw}{\sqrt{v_{dw}} + \epsilon}$
$b = b - a * \frac{db}{\sqrt{v_{db}} + \epsilon}$

Selon article de site web towardsdatascience

Et la loss function utilisé est la categorical_crossentropy.

Question 2

Model complexity: for each experiment (shallow network learning from raw data, shallow network learning from features, CNN, and Fashion MNIST), select a neural network topology and describe the inputs, indicate how many are they, and how many outputs. Compute the number of weights of each model (e.g., how many weights between the input and the hidden layer, how many weights between each pair of layers, biases, etc..) and explain how do you get to the total number of weights.

Pour commencer nous avons garder les modèles de base de chaque notebook.

Shallow network learning from raw data

Nbr d'entrés : 784
Nbr de sorties : 10 classes
Nbr de couches cachée : 1
Nbr de neurones cachés : 300
Nbr de poids dans couche caché : 784 * 300 + 300 = 235 500 Nbr de poids à la sortie : 300 * 10 + 10 = 3010

Nbr de poids total : 235 500 + 3010 = 238 510

Shallow network learning from features

Nbr d'entrés : 392
Nbr de sorties : 10 classes
Nbr de couches cachée : 1
Nbr de neurones cachés : 200
Nbr de poids dans couche caché : 392 * 200 + 200 = 78'600 Nbr de poids à la sortie : 200 * 10 + 10 = 2'010

Nbr de poids total :78'600 + 2'010 = 80'610

CNN

Nbr de poids Layer 1 : (Conv2D) 9 * 25 + 9 = 234 Nbr de poids Layer 2 : (Conv2D) 9^2 * 25 + 9 = 2034
Nbr de poids Layer 3 : (Conv2D) 16 * 9^2 + 16 = 1312 Nbr de poids Layer 4 : (Dense) 25 * 144 + 25 = 3625
Nbr de poids Layer 5 : (Dense) 10 * 25 + 10 = 260

Nbr de poids total : 234 + 2034 + 1312 + 3625 +260 = 7465

Fashion MNIST

Question 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

Oui les réseaux profonds ont plus de capacité car ils sont plus efficaces en terme de nombre de poids. Comme vu au point numéro 2, les 2 réseaux shallow ont beaucoup plus de poid que le réseau convolutionnel. => Moins de poid permet de traiter plus rapidement une époch.

Question 4

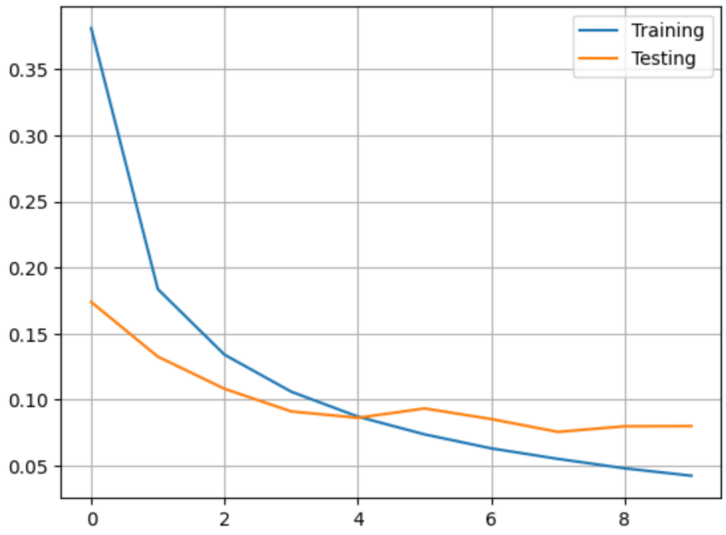
Test every notebook for at least three different meaningful cases (e.g., for the MLP exploiting raw data, test different models varying the number of hidden neurons, for the feature-based model, test pix_p_cell 4 and 7, and number of orientations or number of hidden neurons, for the CNN, try different number of neurons in the feed-forward part) describe the model and present the performance of the system (e.g., plot of the evolution of the error, final evaluation scores and confusion matrices). Comment the differences in results. Are there particular digits that are frequently confused?

Shallow network learning from raw data

Pour ce notebook nous avons principalement essayé de faire varier le nombre de neurones dans la couches cachée.

Résultat avec 100 neurones:

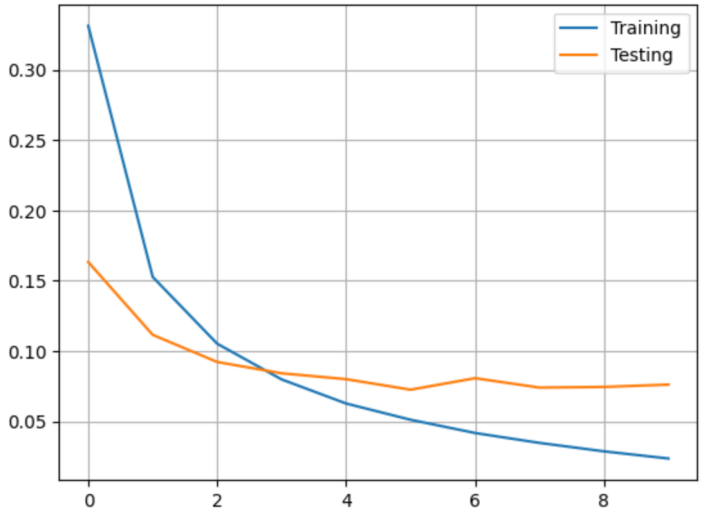
```
Test score: 0.08035963028669357
Test accuracy: 0.9768999814987183
```




```
313/313 [==============================] - 1s 2ms/step
array([[ 969,    0,    1,    2,    1,    1,    2,    1,    1,    2],
       [   0, 1125,    4,    0,    0,    1,    2,    0,    3,    0],
       [   4,    4, 1005,    1,    2,    0,    3,    7,    6,    0],
       [   0,    0,    6,  983,    0,    5,    0,    5,    2,    9],
       [   0,    0,    5,    0,  958,    0,    3,    1,    1,   14],
       [   3,    0,    0,    9,    2,  864,    4,    2,    5,    3],
       [   7,    3,    0,    0,    1,    3,  943,    0,    1,    0],
       [   1,    6,    7,    4,    1,    0,    0,  999,    1,    9],
       [   5,    0,    4,    5,    4,    2,    6,    2,  940,    6],
       [   2,    5,    0,    5,    9,    1,    0,    4,    0,  983]])
```

Résultat avec 200 neurones:

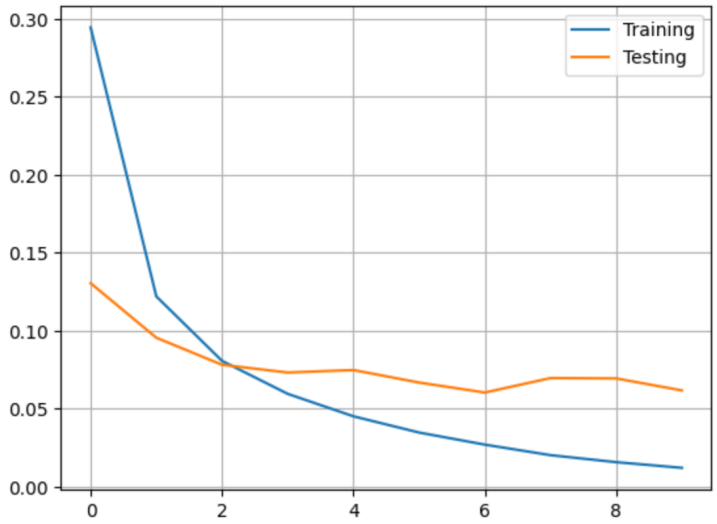
```
Test score: 0.06774899363517761
Test accuracy: 0.9793000221252441
```



```
313/313 [==============================] - 1s 1ms/step
array([[ 968,    0,    3,    0,    1,    0,    4,    1,    2,    1],
       [   0, 1130,    1,    0,    0,    1,    2,    0,    1,    0],
       [   3,    1, 1012,    3,    4,    0,    2,    5,    2,    0],
       [   0,    0,    4,  993,    1,    4,    0,    3,    3,    2],
       [   0,    1,    1,    1,  967,    0,    6,    1,    0,    5],
       [   2,    1,    0,    9,    1,  865,    8,    2,    3,    1],
       [   0,    3,    1,    1,    4,    3,  946,    0,    0,    0],
       [   0,    8,    8,    1,    0,    0,    0, 1009,    1,    1],
       [   5,    2,    7,    7,    4,    4,    4,    4,  935,    2],
       [   1,    7,    1,    6,    9,    2,    2,    8,    5,  968]])
```

Résultat avec 400 neurones:

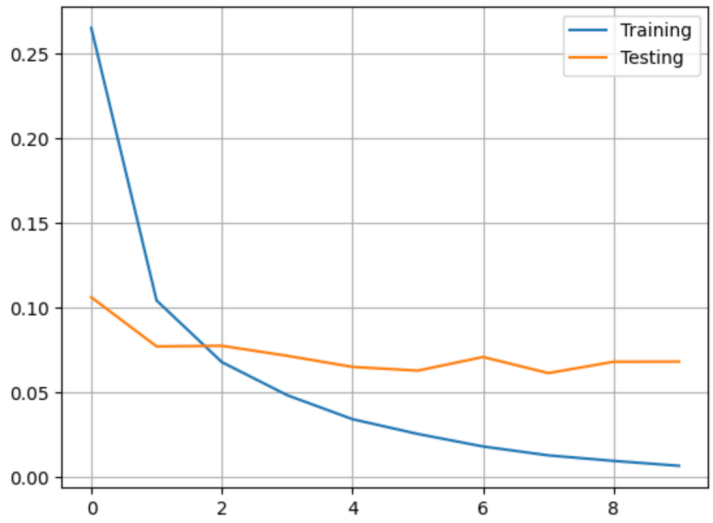
```
Test score: 0.06080867350101471
Test accuracy: 0.9817000031471252
```




```
313/313 [==============================] - 1s 2ms/step
array([[ 971,    1,    1,    0,    1,    0,    3,    1,    2,    0],
       [   0, 1124,    3,    1,    0,    1,    2,    2,    2,    0],
       [   3,    1, 1012,    0,    3,    0,    1,    8,    3,    1],
       [   0,    0,    4,  993,    0,    2,    0,    3,    3,    5],
       [   1,    0,    2,    1,  960,    0,    3,    4,    0,   11],
       [   2,    0,    0,   11,    0,  871,    3,    0,    2,    3],
       [   5,    2,    3,    1,    4,    2,  938,    1,    2,    0],
       [   0,    2,    7,    2,    0,    0,    0, 1013,    1,    3],
       [   3,    0,    3,    5,    4,    2,    1,    4,  949,    3],
       [   2,    2,    0,    5,    4,    2,    0,    8,    0,  986]])
```

Résultat avec 800 neurones:

```
Test score: 0.06336165219545364
Test accuracy: 0.9817000031471252
```




```
313/313 [==============================] - 1s 2ms/step
array([[ 969,    1,    2,    0,    0,    0,    3,    1,    4,    0],
       [   0, 1126,    2,    1,    0,    0,    2,    2,    2,    0],
       [   2,    1, 1010,    4,    3,    0,    1,    6,    5,    0],
       [   0,    0,    3,  991,    0,    4,    0,    2,    5,    5],
       [   0,    1,    3,    0,  970,    0,    3,    1,    1,    3],
       [   2,    0,    0,    6,    1,  874,    3,    1,    3,    2],
       [   3,    3,    0,    1,    2,    3,  946,    0,    0,    0],
       [   0,    4,    6,    1,    1,    0,    0, 1010,    3,    3],
       [   2,    0,    3,    3,    3,    1,    2,    4,  955,    1],
       [   3,    7,    0,    5,   16,    3,    0,    7,    2,  966]])
```

Interprétation des résultats:

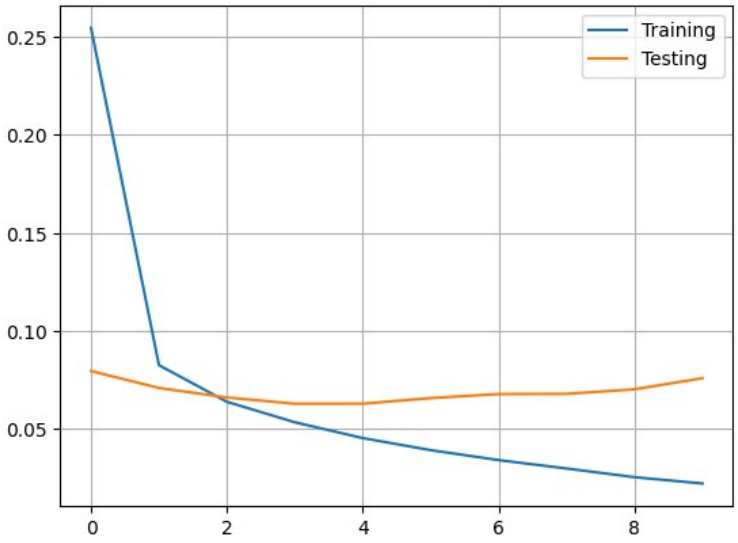
A partir de 400 neurones, on commence à observer un overfitting du modèle. Encore plus présent pour une couche cachée possèdant 800 neurones (l'erreur de la courbe de test commence à remonter après plusieurs epochs).

Pour ce cas-ci 200 neurones semble être un bon nombre de neurones.

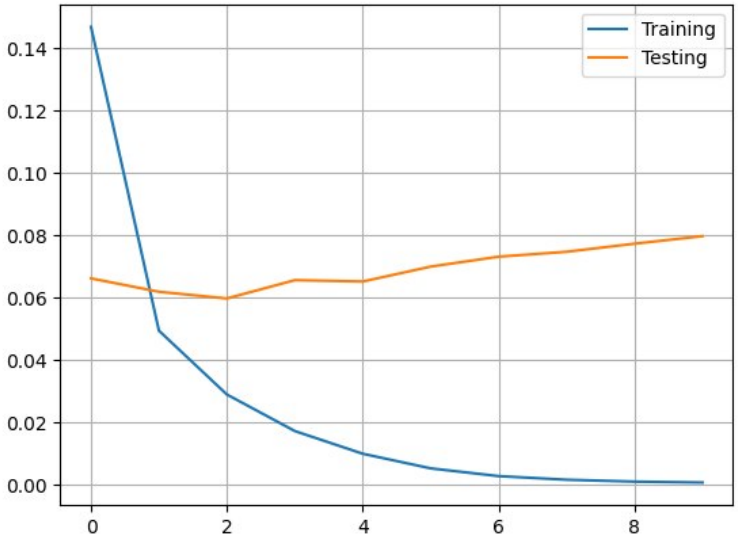
Shallow network learning from features

Ici l'augmentation des neurones n'améliore pas l'entraînement du modèle passé les 200 neurones. Nous avons ensuite choisit aléatoirement diverse valeur de pixel_p_cell et d'orientation.

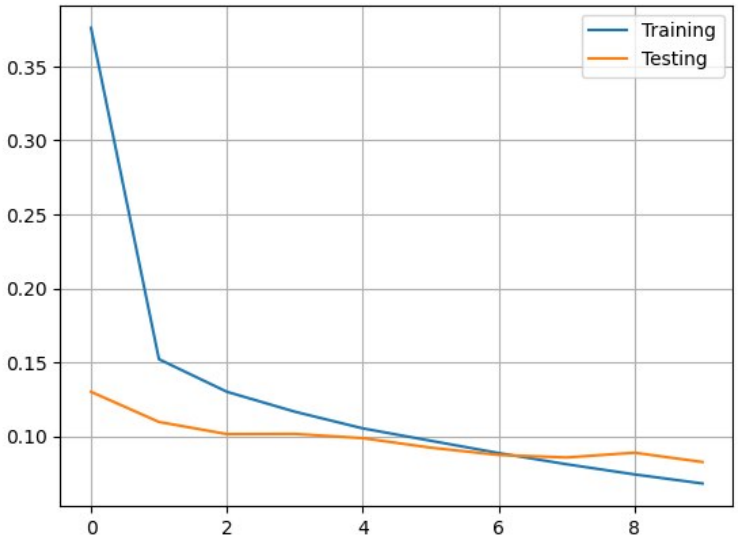
Résultat avec 4 pixel_p_cell, 20 d'orientation et 100 neurones:



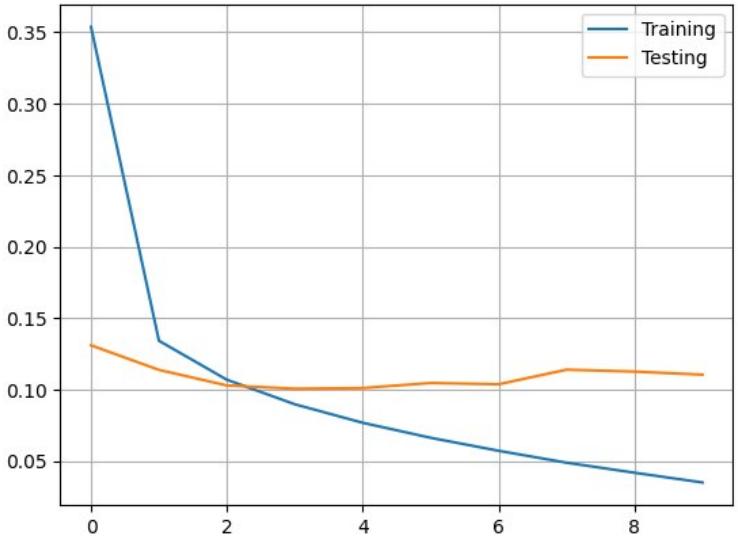
Résultat avec 2 pixel_p_cell, 12 d'orientation et 200 neurones:



Résultat avec 7 pixel_p_cell, 21 d'orientation et 200 neurones:



Résultat avec 7 pixel_p_cell, 64 d'orientation et 200 neurones:



Interprétation des résultats: Après plusieurs essaies, on voit que la configuration 7 pixel par cell, avec une orientation à 21 et un nombre de neurones à 200 obtient les meilleures résultats avec une courbe de test assez proche de la courbe de training.

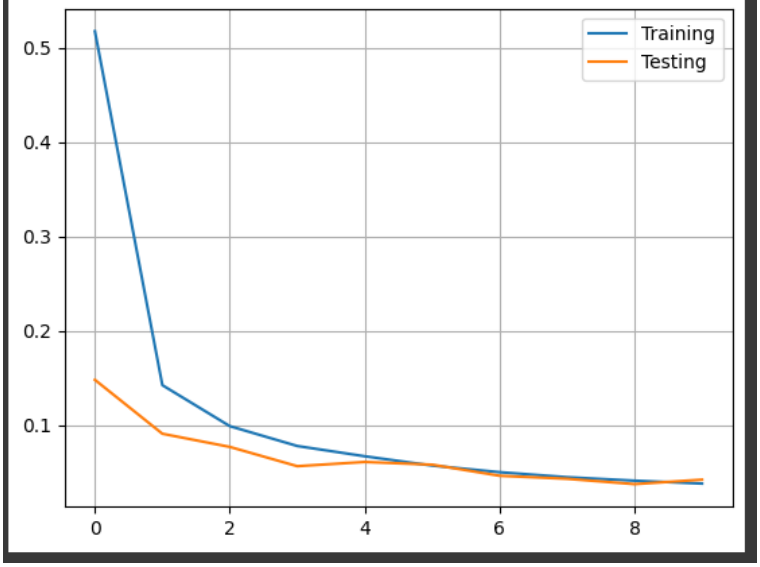
CNN

Pour cette partie il était indiqué dans la donnée de principalement changé le nombre de neurones dans la couches caché. Nous avons tout de même essayer de varier le nombre d'épochs. Ceci n'a juste fait qu'augementer l'overfitting. Puis nous avons essayer de changer les dimensions des filtres de convolution et du pooling mais cela n'a pas eu d'impact positif sur l'entraînement du modèle.

Nous allons donc uniquement nous concentrer sur la variation du nombre de neurones dans la couche caché.

Résultat avec 25 neurones:

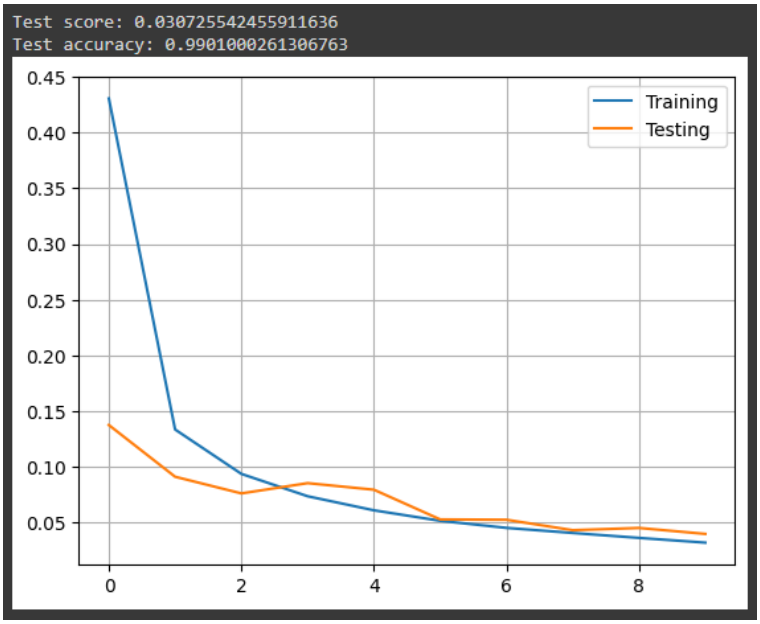
```
Test score: 0.04111729934811592
Test accuracy: 0.9860000014305115
```




```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 969,    0,    0,    0,    0,    1,    7,    1,    2,    0],
       [   0, 1132,    0,    0,    0,    0,    1,    1,    0,    1],
       [   0,    2, 1027,    0,    0,    0,    0,    2,    1,    0],
       [   1,    0,    5,  985,    0,   14,    0,    3,    2,    0],
       [   0,    0,    2,    0,  973,    0,    5,    0,    0,    2],
       [   0,    0,    0,    0,    0,  890,    2,    0,    0,    0],
       [   2,    2,    0,    1,    2,    5,  946,    0,    0,    0],
       [   0,    4,   11,    0,    0,    1,    0, 1005,    1,    6],
       [   2,    0,    5,    2,    1,    8,    4,    2,  946,    4],
       [   2,    3,    0,    0,    7,    3,    2,    4,    1,  987]])
```

Résultat avec 100 neurones:

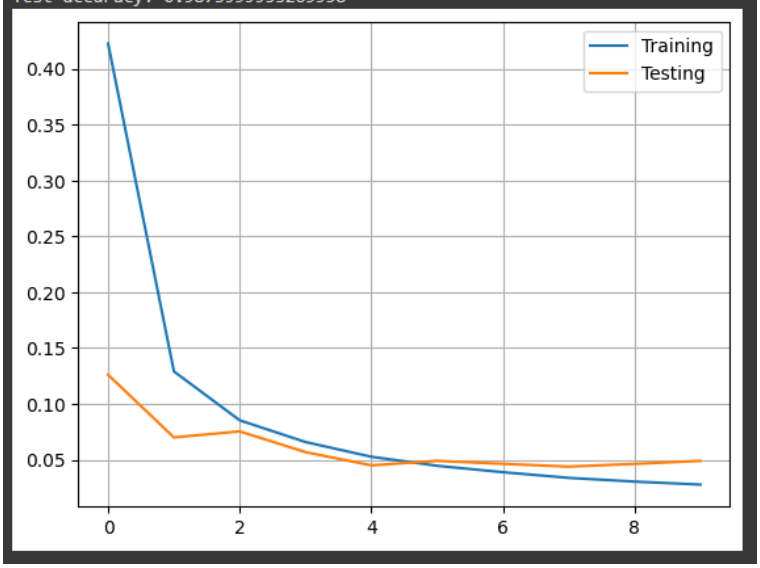
```
Test score: 0.030725542455911636
Test accuracy: 0.9901000261306763
```



```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 977,    0,    0,    0,    0,    0,    1,    1,    1,    0],
       [   0, 1131,    3,    0,    0,    0,    0,    1,    0,    0],
       [   2,    0, 1022,    0,    0,    0,    0,    7,    1,    0],
       [   0,    0,    4,  997,    0,    5,    0,    1,    1,    2],
       [   0,    2,    0,    0,  977,    0,    0,    0,    0,    3],
       [   1,    0,    0,    2,    0,  883,    2,    0,    0,    4],
       [   3,    2,    0,    0,    2,    1,  950,    0,    0,    0],
       [   1,    2,    2,    0,    1,    0,    0, 1016,    1,    5],
       [   6,    0,    6,    1,    0,    0,    1,    2,  953,    5],
       [   1,    2,    0,    0,    6,    1,    0,    4,    0,  995]])
```

Résultat avec 200 neurones:

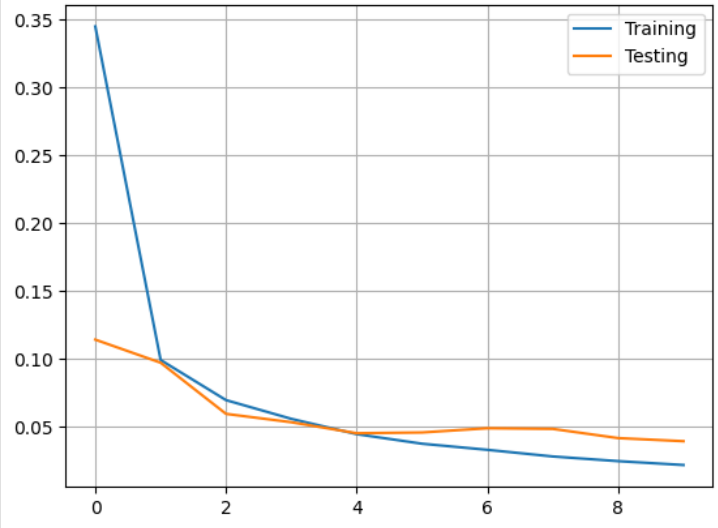
```
Test score: 0.04125058650970459
Test accuracy: 0.9873999953269958
```




```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 975,    0,    2,    0,    0,    0,    1,    1,    1,    0],
       [   0, 1130,    0,    0,    2,    0,    1,    2,    0,    0],
       [   2,    4, 1006,    3,    0,    0,    0,   15,    2,    0],
       [   1,    0,    1,  991,    0,    7,    0,    6,    2,    2],
       [   0,    0,    0,    0,  978,    0,    3,    1,    0,    0],
       [   2,    0,    0,    5,    0,  880,    1,    1,    2,    1],
       [   3,    3,    0,    0,    4,    3,  945,    0,    0,    0],
       [   0,    0,    0,    0,    1,    0,    0, 1026,    0,    1],
       [   3,    0,    0,    1,    1,    2,    1,    3,  960,    3],
       [   0,    0,    2,    2,   13,    2,    0,    7,    0,  983]])
```

Résultat avec 400 neurones:

```
Test score: 0.03265410289168358
Test accuracy: 0.9894000291824341
```




```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[ 974,    0,    1,    0,    0,    0,    3,    1,    1,    0],
       [   0, 1129,    1,    3,    0,    0,    1,    1,    0,    0],
       [   1,    1, 1026,    2,    0,    0,    0,    1,    1,    0],
       [   0,    0,    2, 1005,    0,    1,    0,    0,    2,    0],
       [   0,    0,    1,    0,  975,    0,    0,    0,    0,    6],
       [   0,    0,    0,   12,    0,  877,    1,    0,    2,    0],
       [   3,    1,    0,    1,    1,    3,  947,    0,    2,    0],
       [   0,    2,   11,    2,    0,    0,    0, 1008,    2,    3],
       [   1,    0,    3,    6,    0,    1,    0,    0,  961,    2],
       [   1,    0,    1,    3,    4,    3,    0,    3,    2,  992]])
```

Interprétation des résultats: Ici nous observons un résultat optimal pour 100 neurones => accuracy proche de 1. Pour 200 et 400, on observe de l'overfitting nous choisirons donc 100 neurones.

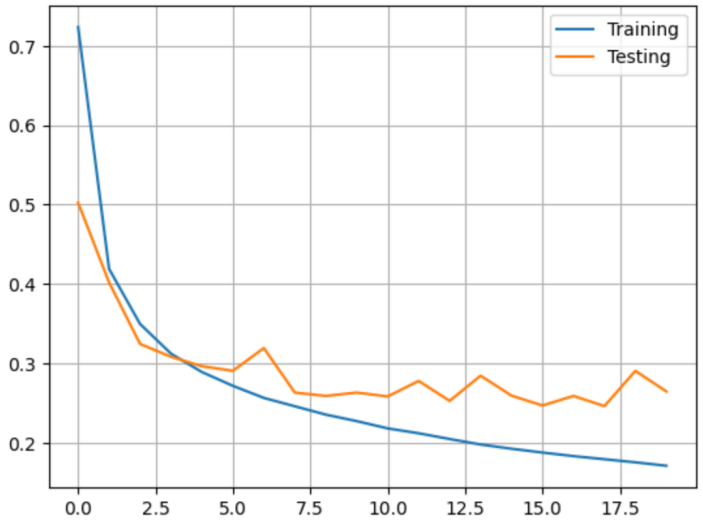
Question 5

Train a CNN to solve the MNIST Fashion problem, present your evolution of the errors during training and perform a test. Present a confusion matrix, accuracy, F-score and discuss your results. Are there particular fashion categories that are frequently confused?

Pour cette dernière partie nous avons augmenter le nombre d'epoch à 20 pour être sûr que le model aie le temps de converger.

Aussi nous avons essayer plusieurs architectures différentes avant de tomber sur celle présente dans le notebook (2 x conv2d + 2 x Dense).

```
Test score: 0.28134334087371826
Test accuracy: 0.9085000157356262
```




```
313/313 [==============================] - 1s 2ms/step
pred.shape =  (10000, 10)
array([[856,   1,  12,  31,   1,   2,  92,   0,   4,   1],
       [  1, 985,   0,  11,   1,   0,   0,   0,   2,   0],
       [ 13,   2, 898,  10,  26,   0,  51,   0,   0,   0],
       [ 14,   6,   7, 944,  11,   0,  17,   0,   1,   0],
       [  0,   1, 110,  44, 794,   0,  48,   0,   3,   0],
       [  0,   0,   0,   0,   0, 970,   0,  23,   0,   7],
       [105,   2,  73,  36,  53,   0, 725,   0,   6,   0],
       [  0,   0,   0,   0,   0,  11,   0, 965,   0,  24],
       [  1,   1,   3,   3,   0,   2,   4,   3, 983,   0],
       [  0,   0,   0,   0,   0,   4,   1,  30,   0, 965]])
```

Interprétation des résultats: Malgré de nombreuse tentative, le model a de la peine à dépasser les 90% d'accuracy.
On peut voir dans la matrice de confusion que il y a 3-4 classes particulièrement difficile à discerner.
Certes avec plus de neurone, on peut réduire cette erreur. Toute fois, elle est loin d'être proportionnel, il faut de plus en plus de neurones pour réduire de moins en moins l'erreur.
Il est sûrement plus intéressant de chercher d'autre méthode que le filtre de convolution pour extraire de nouvelles caractéristiques pour chaque image afin d'aider le model à mieux répartir chaque image.