Lab 04: Deep Neural Networks Shyshmarov Alexandre - Pinto da Cunha da Mata Guilherme

2. Digit recognition from raw data

Select a final model (e.g., the one with best performance) for analysis (please, see below the summary of work to know what to include in the report)

```
Pour mon model final j'ai modifié par rapport au model de base :
    La structure de réseau de neuron. J'ai ajouté une deuxième couche cachée, j'ai
changé le nombre de neuron (il est a 30 maintenant) et la fonction d'activation,
c'est la relu maintenant

    J'ai modifié le nombre d'epoch. On est a 10 alors qu'avant, on était a 3
```

1. What is the learning algorithm being used to optimize the weights of the neural networks?

```
RMSprop.
Etant donnée que l'algo n'a pas de parametre il utlise les parametre par défaut :
learning_rate = 0.001 / rho = 0.9 / epsilon = 1e-7 / etc..

La loss fonction est categorical_crossentropy
```

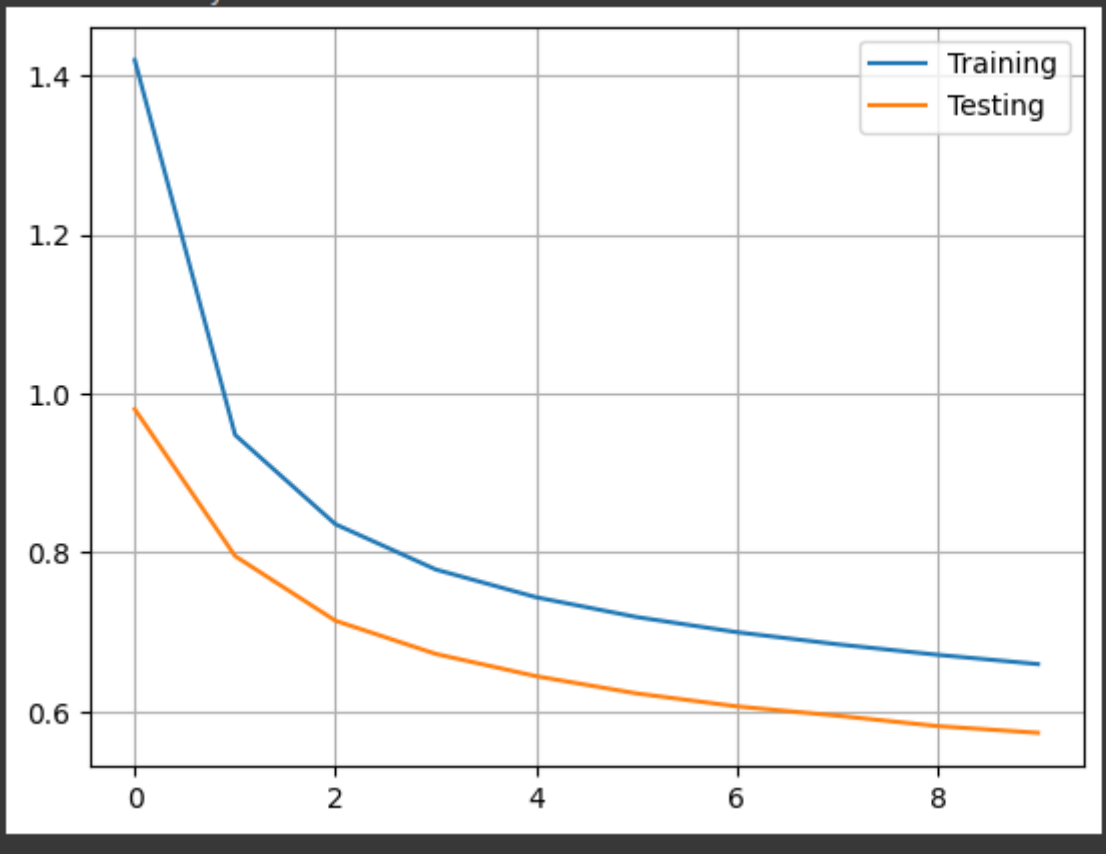
2. For each experiment excepted the last one (shallow network learning from raw data, shallow network learning from features and CNN)
    1. Select a neural network topology and describe the inputs, indicate how many are they, and how many outputs?
    2. Compute the number of weights of each model (e.g., 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 le model que j'ai sélectionner avant :

Il y a 784 inputs ce qui correspond à une image 28x28. Il y a 30 neurones sur la
couche input + les biais de chaque neurone.
On est à 23550 paramètres. La seconde couche a 30 neurones et a une fonction
d'activation relu. Avec 30 entré par neurones
et 30 neurones au total on est à 900 paramètres + 30 biais. La couche output a 10
neurones qui correspond aux 10 chiffres
qu'on veut prédire. Et a softmax comme fonction d'activation. Chaque neurone sur
cette couche a 30 entrées ce qui fait 300
paramètres + 10 biais.

Le total : Input = 30 * 784 + 30 / Hidden = 30 * 30 + 30 / Output = 30 * 10 + 10
```

3. Test at least three different meaningful cases

Quand je change le nombre d'input à 3, le model confond :
5 avec 0 / 5 avec 8 / 8 avec 2 / 8 avec 5 / 9 avec 7

```
Test score: 0.6670774221420288
Test accuracy: 0.7871000170707703
```



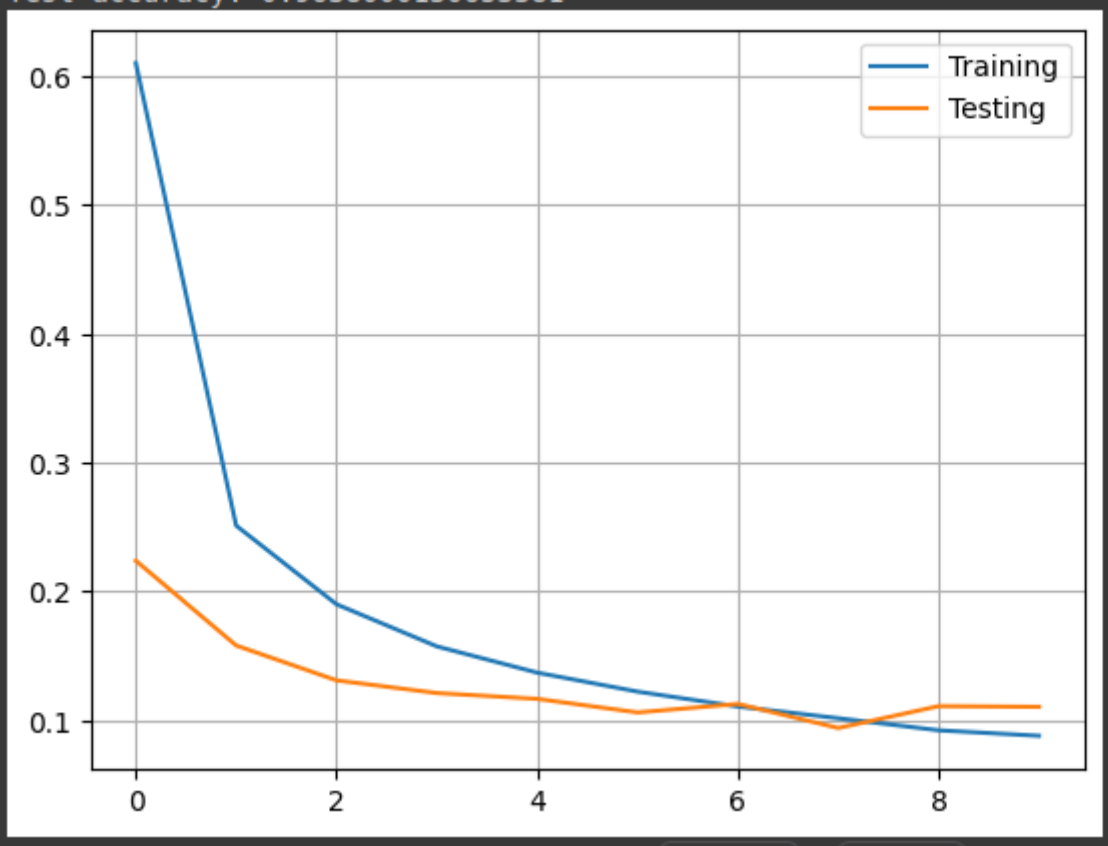
La courbe est pas trop mal, on peut également voir qu'on a une loss qui est plutôt grande. Ce qui est assez mauvais pour
un modèle.

```
array([[ 760,    1,    3,    0,    8,    4,   52,    1,  147,    4],
       [   0, 1082,   16,   10,    0,    1,    3,    2,   19,    2],
       [  11,   44,  776,   48,    5,    2,   72,   12,   57,    5],
       [   0,   51,   58,  786,    3,   51,    4,   17,   36,    4],
       [   7,    1,    0,    1,  800,    6,   13,   11,   24,  119],
       [  15,    5,   11,   76,   44,  602,   10,    4,  105,   20],
       [  63,    2,    9,    0,    3,    4,  800,    4,   62,   11],
       [   0,   31,    3,    1,   17,    0,   15,  880,   11,   70],
       [ 101,   15,   17,   24,   26,  109,   24,    8,  624,   26],
       [   6,    4,    0,    3,  117,   16,    7,   83,   12,  761]])
```

La matrice de confusion reflète ce qui a été dit plus haut. Certains nombres ont du mal a été reconnu par notre modèle.

Quand j'augment le nombre de couche caché à 3 couches , le model confond :
3 avec 9 / 8 avec 2 / 8 avec 3 / 9 avec 7

```
Test score: 0.1258256733417511
Test accuracy: 0.9638000130653381
```



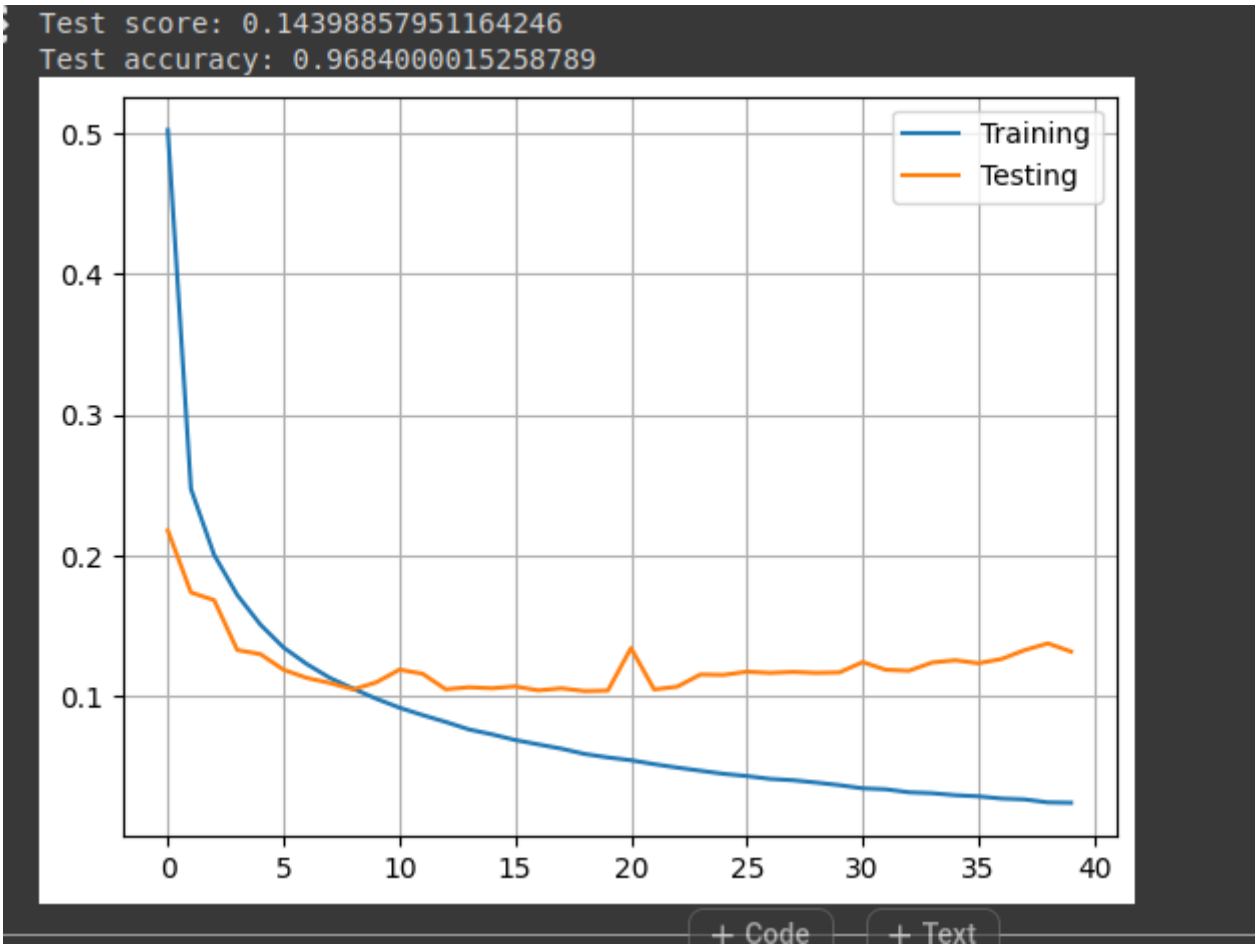
On peut voir que le modèle est meilleur que celui avant, mais on voit que partir de 7 epoch on commence à voir de l'overfitting.
Au niveau de la loss, elle est bien mieux qu'avant et l'accuracy est presque à 1.

```
array([[ 961,    0,    0,    2,    0,    4,    9,    1,    1,    2],
       [   0, 1109,    1,    2,    0,    1,    3,    0,   19,    0],
       [   4,    1,  974,    4,    6,    3,    2,    8,   29,    1],
       [   1,    0,    1,  974,    1,    4,    0,    5,   21,    3],
       [   1,    0,    3,    0,  931,    2,    8,    2,    4,   31],
       [   1,    1,    0,    7,    1,  865,    4,    0,    9,    4],
       [   9,    2,    1,    1,    2,    6,  932,    0,    5,    0],
       [   1,    3,   11,    7,    3,    0,    0,  969,    8,   26],
       [   0,    0,    1,    5,    3,    5,    2,    1,  956,    1],
       [   4,    4,    0,   12,    6,    5,    1,    2,    8,  967]])
```

La matrice de confusion est presque parfaite, si ce ne sont quelques chiffres qui posent toujours problème.

Quand j'augmente le nombre d'epoch a 40 , le model confond :
1 avec 6 / 2 avec 8 / 3 avec 5 / 9 avec 4

```
Test score: 0.14398857951164246
Test accuracy: 0.9684000015258789
```



On a très clairement un gros problème d'overfitting dès le début.

```
array([[ 966,    0,    2,    2,    1,    2,    3,    1,    2,    1],
       [   0, 1123,    4,    2,    0,    1,    3,    1,    0,    1],
       [   3,    4, 1004,    5,    4,    0,    3,    5,    4,    0],
       [   0,    1,    6,  978,    1,    8,    0,    5,    6,    5],
       [   1,    0,    6,    1,  948,    0,    6,    3,    3,   14],
       [   4,    1,    0,   11,    3,  854,    8,    2,    5,    4],
       [  11,    3,    0,    1,    7,    8,  924,    1,    3,    0],
       [   0,    3,    8,    3,    2,    0,    0, 1004,    2,    6],
       [   5,    3,   11,    6,    5,    9,    2,    6,  925,    2],
       [   0,    4,    0,    3,   10,    5,    1,   16,   12,  958]])
```

Malgré l'overfitting vu juste avant, la matrice est plutôt pas mal. Mais si, on test le modèle sur une autre dataset,
on aurait sûrement des résultats un peu moins bien.

3. Digit recognition from features of the input data

1. What is the learning algorithm being used to optimize the weights of the neural networks?

```
RMSprop c’est une variante de la descente de gradient stochastique.
```

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

```
Comme on a aucune valeurs explicite, on utilise les valeurs par défaut.
Learning_rate : 0.001
Rho : 0.9
Momentum : 0.0
Epsilon : 1e-7
Centered : false
Use_ema :false
Ema_momentum : 0.99
Ema_overwrite_frquency : None

https://keras.io/api/optimizers/rmsprop/
```

2. What loss function is being used ?

```
categorical_crossentropy
```

3. Please, give the equation(s)

```
RMSprop:
```

$$E[g^2](t) = \beta E[g^2](t-1) + (1-\beta)(\frac{\partial c}{\partial w})^2$$

$$w_{ij}(t) = w_{ij}(t-1) - \frac{\eta}{\sqrt{E[g^2]}} \frac{\partial c}{\partial w_{ij}}$$

```
categorical_crossentropy :
CE = - ∑p dp log(yp)
```

1. For each experiment excepted the last one (shallow network learning from raw data, shallow network learning from features and CNN)

1. Select a neural network topology and describe the inputs, indicate how many are they, and how many outputs?

```
La topologie du réseau neuronal est un réseau peu profond (shallow network) avec
une couche cachée de 2 neurones et une couche de sortie de 10 neurones
correspondant aux classes de sortie (les chiffres de 0 à 9).

Pour notre modèle on a décidé de mettre 100 neurones.

Le nombre de sorties est égal au nombre de classes dans le problème de
classification. Comme il s'agit de la base de données MNIST, qui contient des
chiffres de 0 à 9, il y a 10 classes au total
```

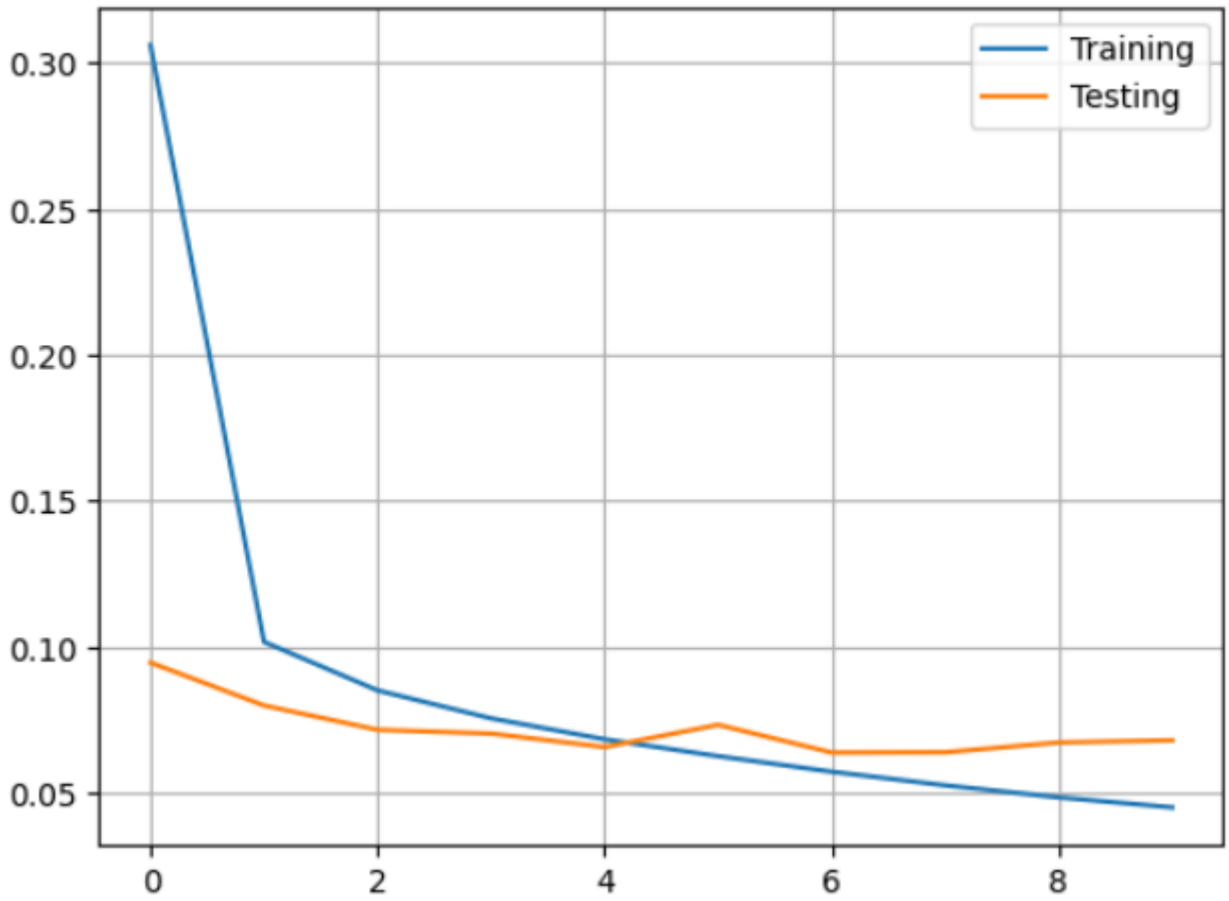
2. Compute the number of weights of each model (e.g., how many weights between the input and the hidden layer, how many weights between each pair of layers, biases, etc..) and explain how do you get to the total number of weights.

```
Comme on a une image de 28x28, le nombre d’orientation qui vaut 8 et pix_p_cell 4.
On a donc que le nombre de caractéristiques extraites par image à partir du
descripteur HOG vaut height * width * n_orientations / (pix_p_cell * pix_p_cell).
En remplaçant les valeurs on obtient 28 * 28 * 8 / (4 * 4) = 392. Le poids de la
première couche qui a 2 neurones vaut donc 786, car chaque caractéristique est
connectée à deux neurones ce qui fait 2 * 392 connexions donc 784. Comme chaque
neurone a un biais on ajoute donc 2 à 784, ce qui nous fait un poids de 786.

Le poids pour la deuxième couche est égal au nombre de neurones de la première
couche * le nombre de neurones de la deuxième. Ce qui nous donne 2 * 10, donc 20.
Il faut ensuite ajouter le biais pour chaque neurone de la deuxième couche, donc
10. Ce qui nous fait 20 + 10 donc 30 est le poids de la deuxième couche.

Le poids total est l’addition des deux couches donc 816 car 786 + 30 = 816.

Notre modèle nous avons utilisé 100 neurones dans la couche cachée, l'idée des
calculs reste la même.
```

3. Test at least three different meaningful cases

100 neurones, 8 orientations, 4 pix_p_cell sur 10 epochs

```
Test score: 0.07152404636144638
Test accuracy: 0.9768999814987183
```

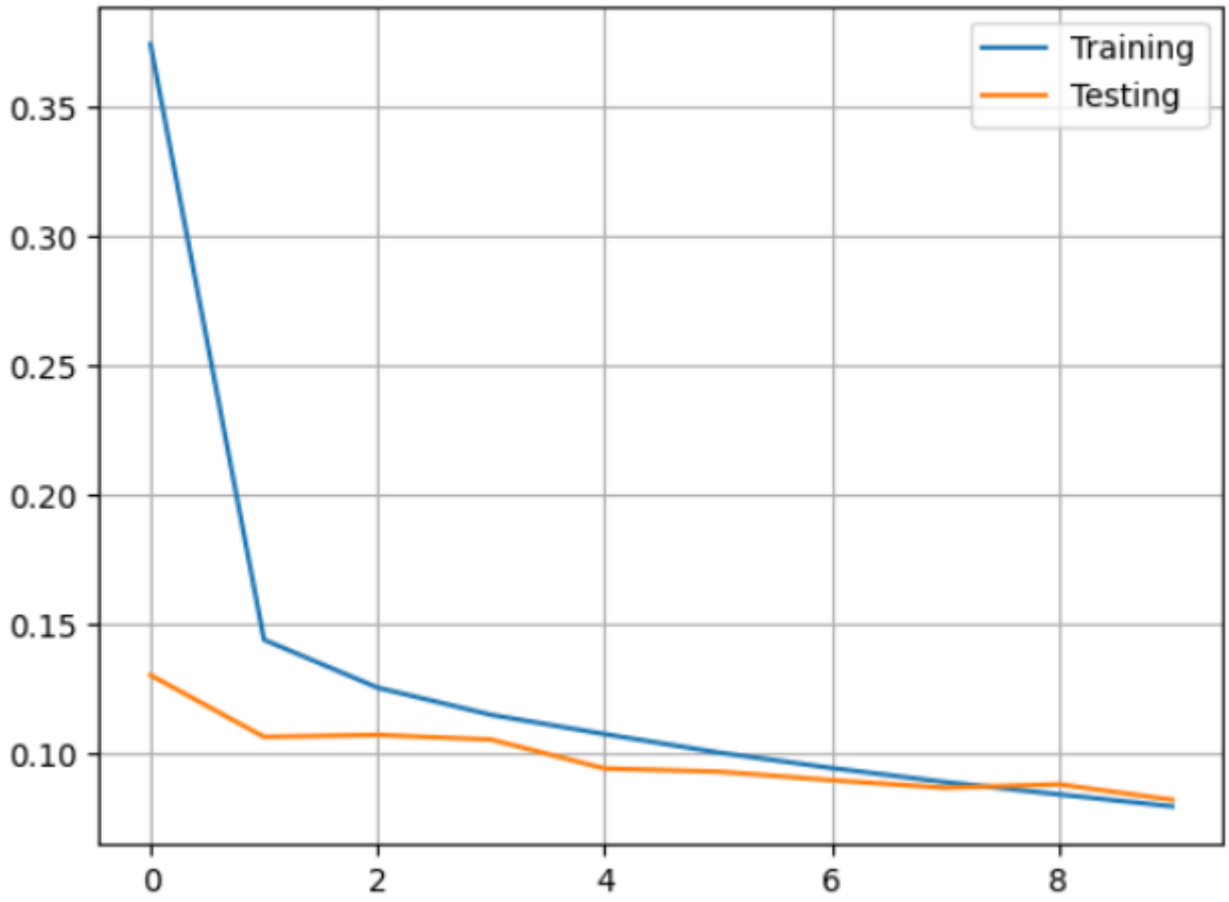



```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[ 974,    0,    2,    0,    0,    1,    0,    1,    1,    1],
       [   2, 1120,    2,    2,    1,    0,    2,    3,    3,    0],
       [   3,    1, 1014,    2,    2,    0,    2,    4,    4,    0],
       [   0,    0,    3,  992,    0,    3,    0,    4,    8,    0],
       [   0,    1,    3,    1,  967,    0,    0,    2,    2,    6],
       [   3,    1,    0,   22,    0,  856,    3,    0,    5,    2],
       [   6,    2,    1,    0,    6,    1,  940,    0,    2,    0],
       [   0,    4,   10,    5,    3,    0,    0,  990,    4,   12],
       [   6,    0,    4,    7,    2,    1,    0,    4,  943,    7],
       [   1,    3,    1,    8,   10,    1,    0,    7,    5,  973]])
```

Cette mesure a servi de point de départ, on a donc utilisé les même valeurs de paramètres pour les autres mesures. On a uniquement modifié un seul paramètre par mesure et celui-ci sera mentionné dans les prochaines mesures. Pour cette première mesure, on peut voir qu'on a de l'overfitting. La courbe de test ne suit pas celle d'entraînement. Quant au loss, il reste quand même assez faible. Si on regarde la matrice on peut voir que la plus grande erreur est pour le chiffre 5 qui est confondu avec le chiffre 3. Autrement avec un peu moin, mais il y a aussi de la confusion avec le chiffre 7 qui est confondu avec le 2. Puis également le 9 qui est confondu avec le 5.

Changement du nombre d'orientations à 4

```
Test score: 0.08369038254022598
Test accuracy: 0.9718000292778015
```

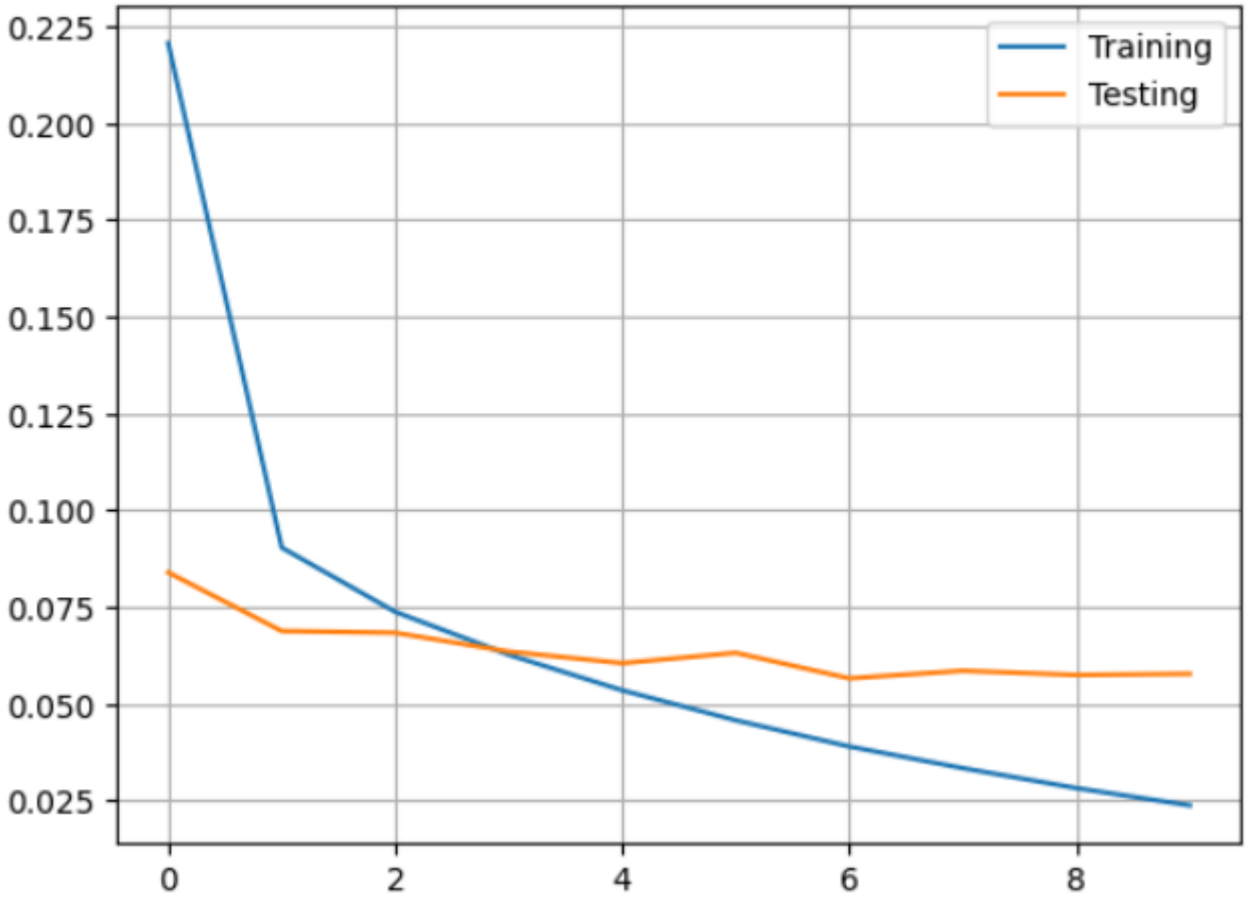



```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[ 964,    2,    3,    0,    0,    3,    4,    2,    2,    0],
       [   0, 1122,    5,    2,    1,    0,    1,    1,    3,    0],
       [   3,    4, 1009,    2,    1,    0,    1,    6,    6,    0],
       [   0,    2,    2,  984,    0,    7,    0,    3,   10,    2],
       [   1,    1,    3,    0,  952,    0,    4,    2,    2,   17],
       [   1,    1,    1,   18,    0,  861,    4,    0,    4,    2],
       [   3,    2,    2,    0,    7,    3,  939,    0,    2,    0],
       [   1,    4,    5,    3,    7,    0,    0,  993,    3,   12],
       [   7,    1,    4,   12,    2,    4,    3,    3,  928,   10],
       [   4,    5,    0,   11,   10,    3,    1,    7,    2,  966]])
```

Pour cette deuxième mesure, on a fait varier le nombre d'orientations. On a plutôt de bons résultats, la courbe de tests ne fait pas une grande descente, mais elle suit bien la courbe d'entraînement. On a un loss relativement faible. Les chiffres les plus confondus reste comme pour le précèdent, le 5 qui est confondu avec le 3. Une nouvelle confusion est apparue de manière plus importante, le chiffre 4 qui est confondu avec le 9. Il y a aussi le 8 qui est confondu avec le 3 et le 7 qui est confondu avec le 9.

Changement du nombre de neurones à 300

```
Test score: 0.05930040776729584
Test accuracy: 0.9812999963760376
```

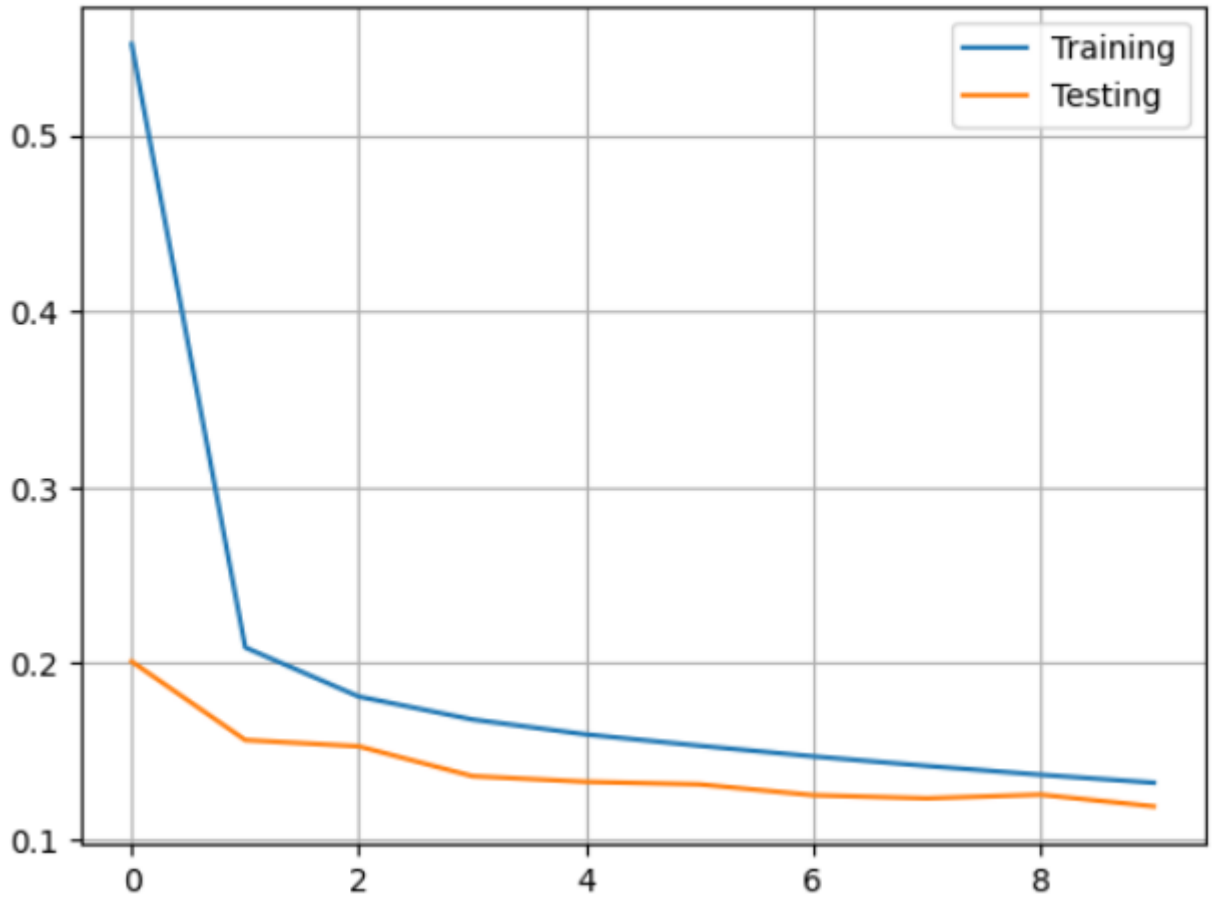



```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[ 967,    0,    1,    0,    0,    2,    4,    3,    2,    1],
       [   3, 1118,    1,    2,    0,    1,    4,    2,    4,    0],
       [   2,    1, 1016,    1,    1,    0,    1,    5,    5,    0],
       [   0,    0,    2,  991,    0,    6,    0,    3,    8,    0],
       [   0,    2,    2,    1,  966,    0,    1,    1,    5,    4],
       [   2,    1,    0,   16,    0,  869,    3,    0,    1,    0],
       [   4,    2,    1,    0,    3,    2,  946,    0,    0,    0],
       [   0,    2,    4,    4,    4,    0,    0, 1006,    4,    4],
       [   4,    0,    0,    2,    1,    1,    2,    5,  955,    4],
       [   1,    3,    1,    6,    6,    3,    0,    5,    5,  979]])
```

Pour la troisième mesure, on a changé cette fois-ci le nombre de neurones. On peut voir qu'on a de l'overfitting, on a complexifié le modèle en augmentant le nombre de neurones. On a un loss assez bas. Concernant les confusions, le plus grand reste toujours la même, le 5 qui est confondu avec le 3. On a une nouvelle confusion avec le 3 qui est confondu avec le 8. Le reste c'est peu les même que pour la première mesure.

Changement du pix_p_cell à 7

```
Test score: 0.1267075091600418
Test accuracy: 0.9581000208854675
```




```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
array([[ 962,    2,    3,    0,    0,    0,    5,    1,    3,    4],
       [   0, 1120,    2,    2,    2,    0,    3,    2,    4,    0],
       [   4,    5,  983,   16,    7,    1,    1,    5,    9,    1],
       [   2,    1,    5,  957,    2,   13,    1,    6,   19,    4],
       [   0,    2,    2,    0,  942,    1,    1,    9,    4,   21],
       [   1,    0,    0,   15,    0,  853,    1,    0,   19,    3],
       [  14,    2,    0,    1,    7,    8,  919,    0,    5,    2],
       [   0,    4,   13,    2,    7,    0,    0,  970,    4,   28],
       [   3,    5,    4,   10,    3,   10,    3,    4,  919,   13],
       [   0,    3,    2,   10,   11,    1,    0,   18,    8,  956]])
```

Pour cette dernière mesure, on a une courbe assez bien, le loss est cependant un peu plus élevé que les précédentes mesures. Les confusions sont donc plus nombreuses. La plus grande confusion est entre le 4 et le 9. Le 3 et le 5 sont souvent confondu avec le 8. Une nouvelle confusion avec le 2 qui est confondu avec le 3. Et sinon de manière assez élevée, on a toujours le 5 qui est avec le 3.

Le meilleur modèle pour nous est le 2ème avec un nombre d’orientations à 4. Il a un bon loss avec
la meilleur courbe sans overfitting.

4. Convolutional neural network digit recognition

1. What is the learning algorithm being used to optimize the weights of the neural networks?

```
RMSprop c’est une variante de la descente de gradient stochastique.
```

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

```
```

Comme on a aucune valeurs explicite, on utilise les valeurs par défaut.
Learning_rate : 0.001
Rho : 0.9
Momentum : 0.0
Epsilon : 1e-7
Centered : false
Use_ema :false
Ema_momentum : 0.99
Ema_overwrite_frquency : None

https://keras.io/api/optimizers/rmsprop/

```
```

2. What loss function is being used ?

```
   categorical_crossentropy
```

3. Please, give the equation(s)

```
RMSprop:
```

$$E[g^2](t) = \beta E[g^2](t-1) + (1-\beta)(\frac{\partial c}{\partial w})^2$$

$$w_{ij}(t) = w_{ij}(t-1) - \frac{\eta}{\sqrt{E[g^2]}} \frac{\partial c}{\partial w_{ij}}$$

```
    categorical_crossentropy :
    CE = - ∑p dp log(yp)
```

1. For each experiment excepted the last one (shallow network learning from raw data, shallow network learning from features and CNN)
    1. Select a neural network topology and describe the inputs, indicate how many are they, and how many outputs?

```
CNN prend en entrée des images de 28x28 pixels avec un seul canal et produit en
sortie des probabilités pour chacune des 10 classes de chiffre.

Il a trois couches de convolution avec des noyaux de taille 2x2 suivies de
MaxPooling de taille 2x2. Une couche d'applatissement pour convertir les
caractéristiques en un vecteur unidirectionnel. Puis deux couches denses avec la
dernière couche produisant des probabilités pour 10 classes de chiffres.

Après plusieurs essais, notre modèle choisi a pour la première couche de
convolution un noyau de taille 7x7, la deuxième 5x5 et la troisième 3x3. On a
aussi mis 10 neurones par couche.
```

2. 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 calculer le poids sur les couches de convolution, j'ai utilisé la formule
suivante:
nombre de filtres * taille du noyau * nombre de filtres couche précédente + biais
Le biais il y en a 1 par filtre.
```

Layer (type)	Output Shape	Param #
l0 (InputLayer)	(None, 28, 28, 1)	0
l1 (Conv2D)	(None, 28, 28, 10)	500
l1_mp (MaxPooling2D)	(None, 14, 14, 10)	0
l2 (Conv2D)	(None, 14, 14, 10)	2,510
l2_mp (MaxPooling2D)	(None, 7, 7, 10)	0
l3 (Conv2D)	(None, 7, 7, 10)	910
l3_mp (MaxPooling2D)	(None, 3, 3, 10)	0
flat (Flatten)	(None, 90)	0
l4 (Dense)	(None, 10)	910
l5 (Dense)	(None, 10)	110

Total params: 4,940 (19.30 KB)
Trainable params: 4,940 (19.30 KB)
Non-trainable params: 0 (0.00 B)

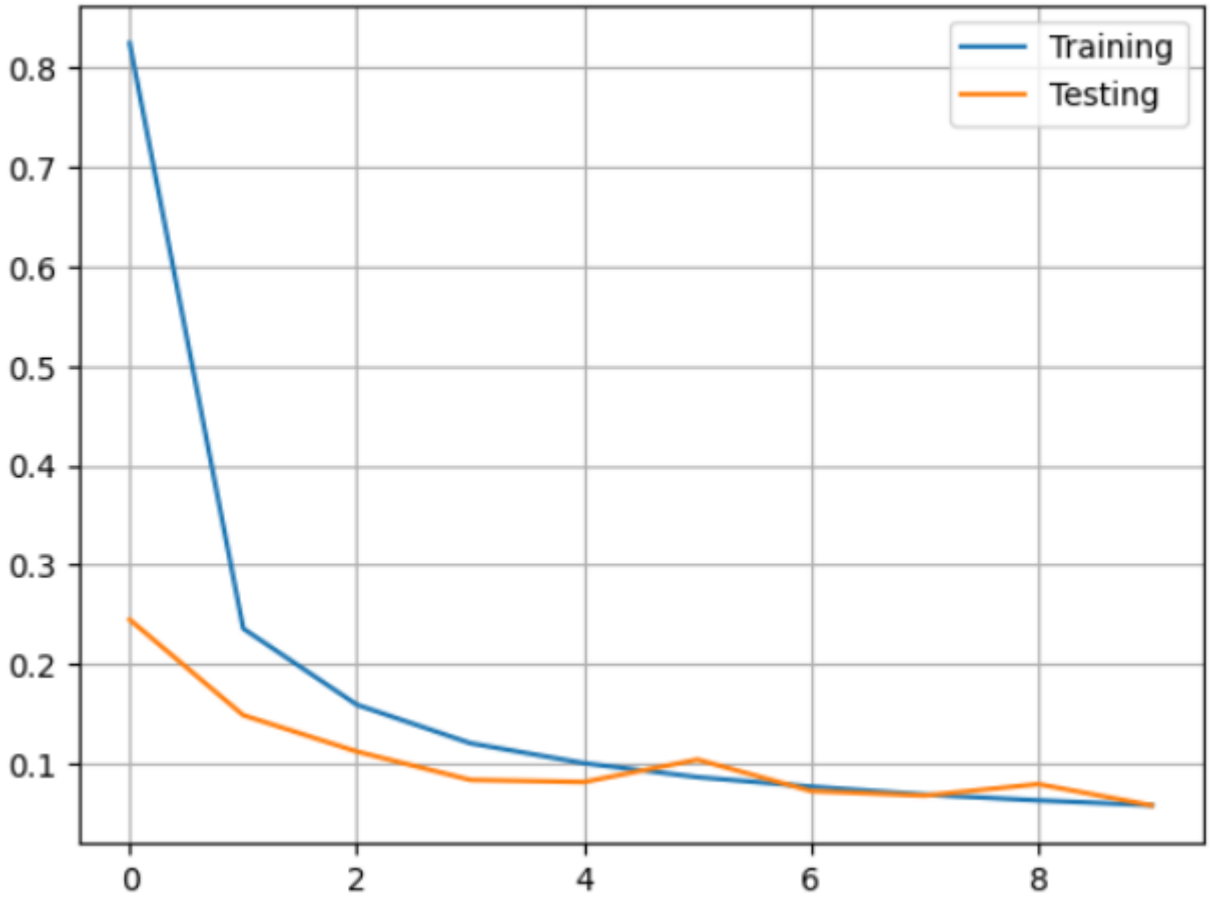
```
Voici ci-dessus le modele choisi.
calcul pour la couche l1:
nombre de filtres = 10
taille du noyau = 7x7
nombre de filtres de la couche précédente = 1
biais = 10
10 * 7 * 7 * 1 + 10 = 500

Couche l2:
nombre de filtres = 10
taille du noyau = 5x5
nombre de filtres de la couche précédente = 10
biais = 10
10 * 5 * 5 * 10 + 10 = 2'510

Couche l3:
nombre de filtres = 10
taille du noyau = 3x3
nombre de filtres de la couche précédente = 10
biais = 10
10 * 3 * 3 * 10 + 10 = 910

Ensuite pour le calcul des couches Dense, j'ai utilisé la formule suivante:
nombre d'entrée * nombre de sorties + biais

Calcul pour l4:
nombre d'entrée = 90
nombre de sorties = 10
biais = 10
```

```
90 * 10 + 10 = 910

Calcul pour l5:
nombre d'entrée = 10
nombre de sorties = 10
biais = 10
10 * 10 + 10 = 110
```

3. Test at least three different meaningful cases

10 neurones dans feed-forward

```
Test score: 0.046614594757556915
Test accuracy: 0.9846000075340271
```

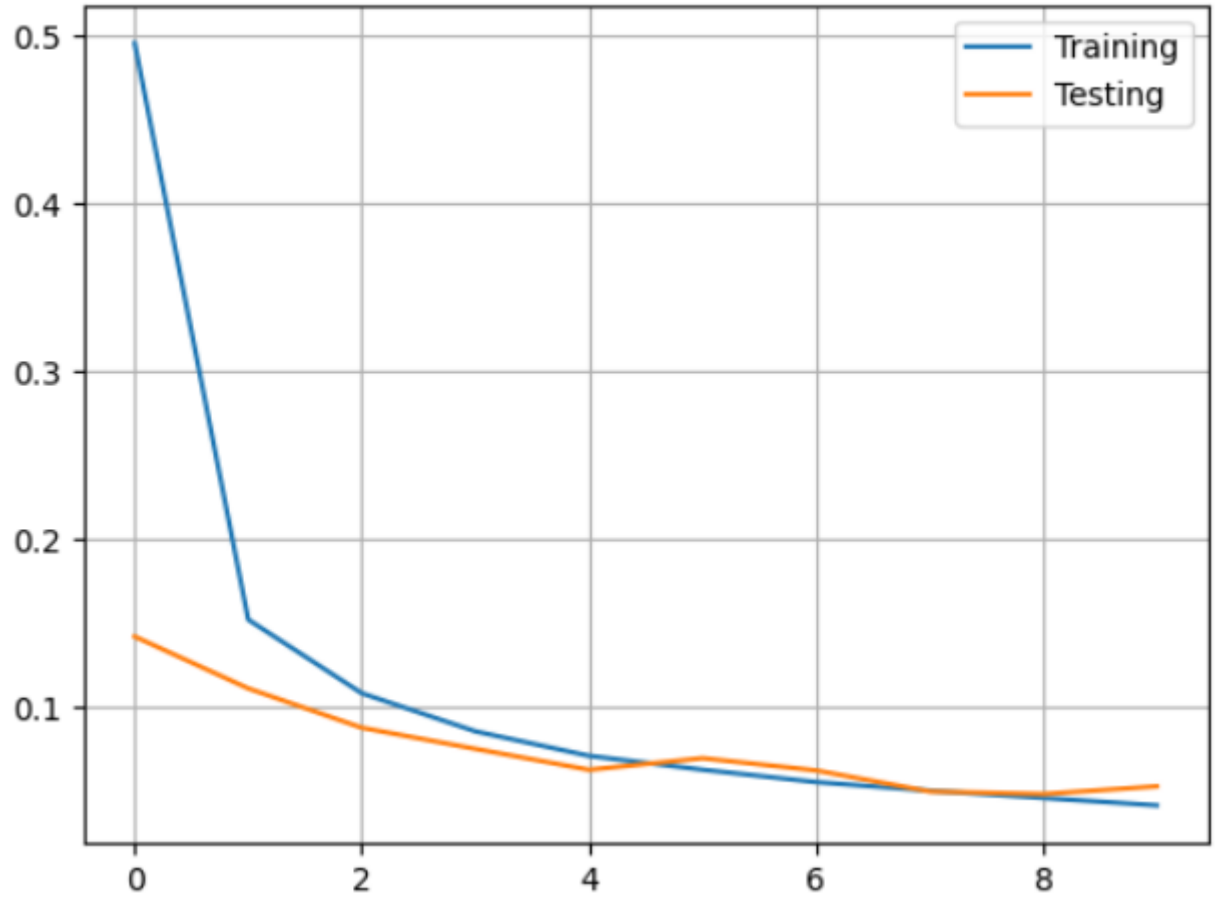



```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step
pred.shape =  (10000, 10)
array([[ 963,    0,    0,    0,    1,    1,    8,    1,    3,    3],
       [   0, 1124,    1,    5,    0,    0,    1,    1,    2,    1],
       [   1,    1, 1019,    1,    1,    0,    0,    2,    7,    0],
       [   0,    0,    2, 1000,    0,    3,    0,    2,    3,    0],
       [   1,    1,    3,    0,  969,    0,    1,    0,    0,    7],
       [   0,    0,    0,   12,    0,  878,    1,    0,    1,    0],
       [   5,    2,    2,    0,    4,    3,  938,    0,    4,    0],
       [   0,    1,    9,    2,    0,    0,    0, 1005,    3,    8],
       [   0,    0,    2,    3,    1,    2,    1,    1,  964,    0],
       [   0,    4,    0,    5,    8,    1,    0,    1,    4,  986]])
```

```
Cette mesure a servi de point de départ, on a donc utilisé les même valeurs de
paramètres pour les autres mesures.
On a uniquement modifié un seul paramètre par mesure qui est le nombre de neurones
dans le feed-forward.
Pour cette première mesure, on peut voir qu’on a de très bons résultats. On a un
loss très faible avec une courbe qui descend bien avec la courbe d'entrainement.
Si on regarde la matrice on peut voir que la plus grande erreur est pour le
chiffre 5 qui est confondu avec le chiffre 3. Autrement avec un peu moin, mais
il y a aussi de la confusion avec le chiffre 7 qui est confondu avec le 2.
```

Changement à 20 neurones dans feed-forward

```
Test score: 0.04692921042442322
Test accuracy: 0.9842000007629395
```

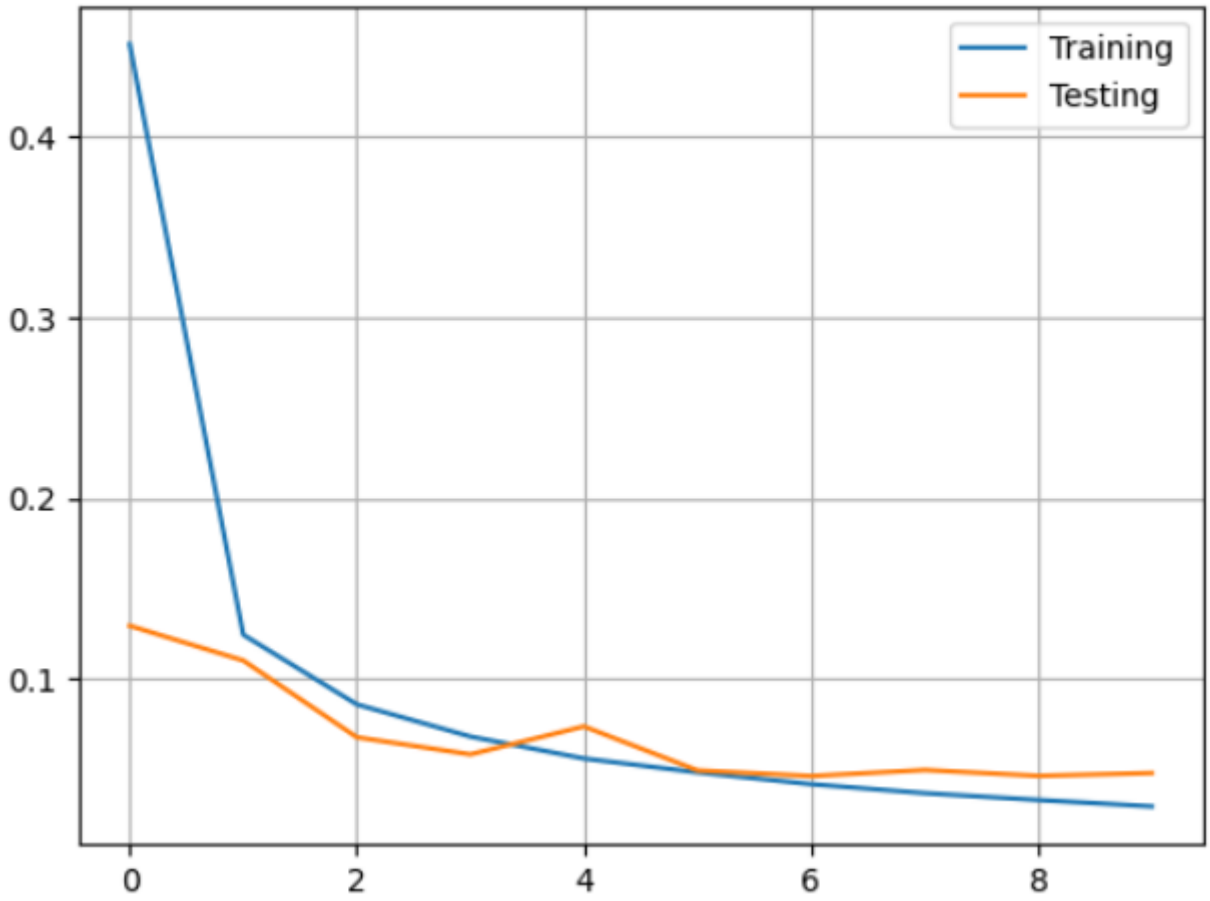



```
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step
pred.shape =  (10000, 10)
array([[ 973,    0,    2,    0,    0,    1,    2,    1,    1,    0],
       [   0, 1122,    3,    2,    0,    1,    1,    3,    3,    0],
       [   1,    1, 1023,    2,    1,    0,    0,    4,    0,    0],
       [   0,    0,    0, 1005,    0,    4,    0,    1,    0,    0],
       [   1,    1,    3,    0,  961,    0,    1,    5,    1,    9],
       [   1,    0,    1,    8,    0,  881,    1,    0,    0,    0],
       [  10,    1,    0,    0,    1,    6,  940,    0,    0,    0],
       [   0,    1,    5,    6,    0,    1,    0, 1014,    1,    0],
       [   6,    0,    3,   12,    0,    6,    2,    4,  934,    7],
       [   2,    4,    0,    5,    3,    4,    0,    2,    0,  989]])
```

Pour cette deuxième mesure, on est passé à 20 neurones. On a plutôt de bons résultats,
la courbe de tests suit bien la coube d'entrainement. On a également un loss relativement faible, comme pour la première mesure.
Si on regarde la matrice on peut voir que la plus grande erreur est pour le chiffre 8 qui est confondu avec le chiffre 3. Autrement avec un peu moin, mais il y a aussi de la confusion avec le chiffre 6 qui est confondu avec le 0. Puis également le 4 qui est confondu avec le 9.

Changement à 100 neurones dans feed-forward

```
Test score: 0.04461170360445976
Test accuracy: 0.9850000143051147
```

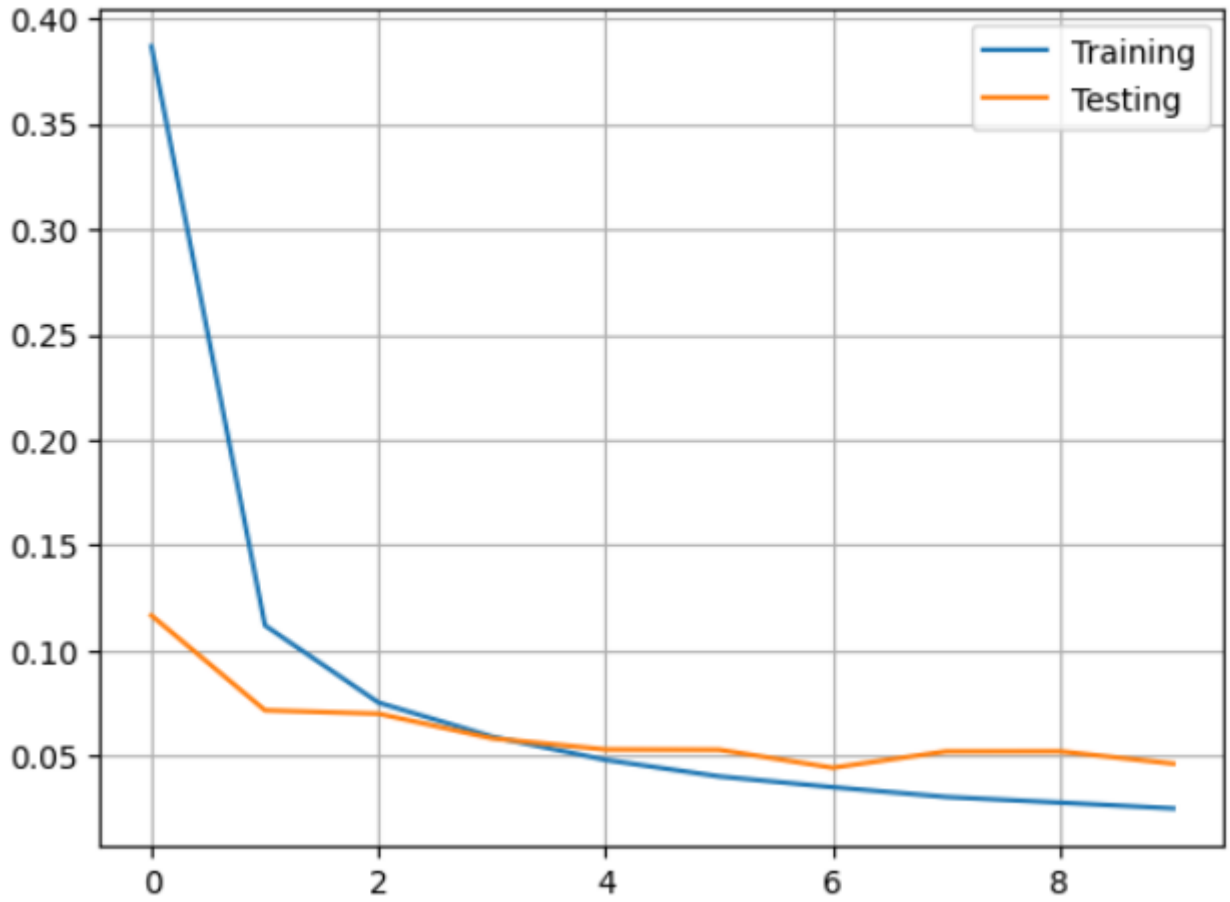



```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step
pred.shape =  (10000, 10)
array([[ 978,    0,    1,    0,    0,    0,    0,    1,    0,    0],
       [   2, 1125,    1,    2,    0,    1,    1,    1,    2,    0],
       [   2,    2, 1017,    0,    2,    0,    0,    3,    5,    1],
       [   0,    0,    2, 1004,    0,    3,    0,    0,    1,    0],
       [   0,    0,    1,    0,  962,    1,    0,    0,    3,   15],
       [   1,    0,    0,    5,    0,  882,    1,    0,    1,    2],
       [  12,    2,    0,    0,    4,    2,  933,    0,    5,    0],
       [   0,    4,   11,    2,    0,    1,    0,  998,    1,   11],
       [   3,    0,    1,    3,    0,    1,    0,    2,  961,    3],
       [   0,    1,    0,    1,    5,    4,    0,    2,    6,  990]])
```

Pour la troisième mesure, on a changé à 100 le nombre de neurones. On peut voir qu'on commence a avoir de l'overfitting, on a complexifié le modèle en augmentant le nombre de neurones. On a quand même un loss assez bas.
Concernant les confusions, le plus grand est cette fois-ci le 4 qui est confondu avec le 9.
On a de nouveau une confusion avec le 6 qui est confondu avec le 0. On a aussi le 7 qui est confondu avec le 2 et le 9.

Changement à 300 neurones dans feed-forward

```
Test score: 0.04091471806168556
Test accuracy: 0.9861000180244446
```




```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step
pred.shape =  (10000, 10)
array([[ 972,    0,    1,    0,    0,    1,    2,    1,    0,    3],
       [   0, 1127,    1,    1,    0,    1,    3,    0,    2,    0],
       [   2,    1, 1027,    0,    0,    0,    1,    1,    0,    0],
       [   0,    0,    3,  986,    0,   16,    0,    0,    3,    2],
       [   0,    0,    1,    0,  963,    0,    3,    0,    0,   15],
       [   1,    0,    0,    2,    0,  888,    1,    0,    0,    0],
       [   2,    2,    0,    0,    1,    4,  949,    0,    0,    0],
       [   0,    1,   13,    3,    0,    2,    0,  994,    2,   13],
       [   1,    0,    2,    2,    0,    5,    1,    2,  952,    9],
       [   0,    0,    0,    0,    1,    1,    1,    2,    1, 1003]])
```

Pour cette dernière mesure, on a de l'overfitting, le loss est reste stable par rapport aux autres mesures. Comme on pouvait s'attendre plus on ajoute de neurones plus le modèle est complexe, plus c'est complexe plus il y a de risque d'overfitting.
La plus grande confusion est entre le 3 et le 5. Suivi par le 4 avec le 9. Comme pour la mesure précédente, on a aussi le 7 qui est confondu avec le 2 et le 9.

Le CNN est celui qui a présenté de meilleurs résultats. C'est celui qui a le loss le plus faible et les courbes sont plutôt bien.

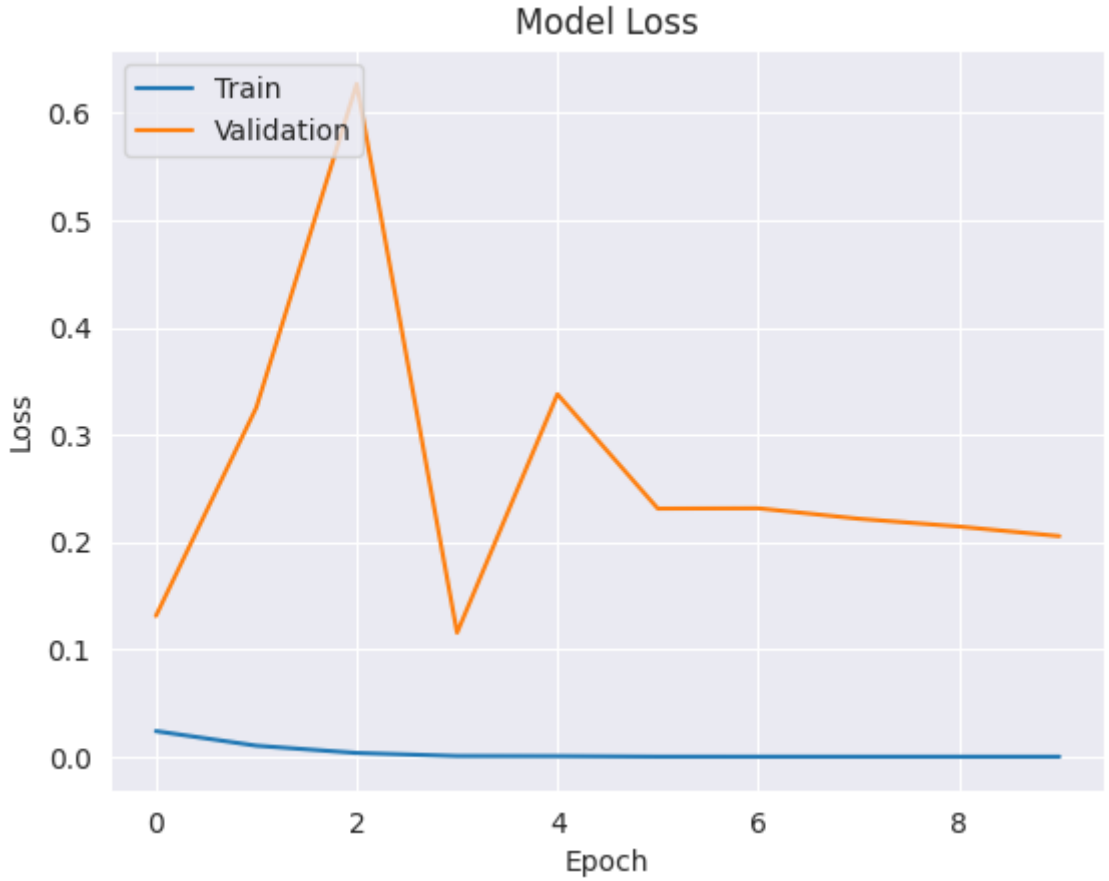
5. Chest X-ray to detect pneumonia

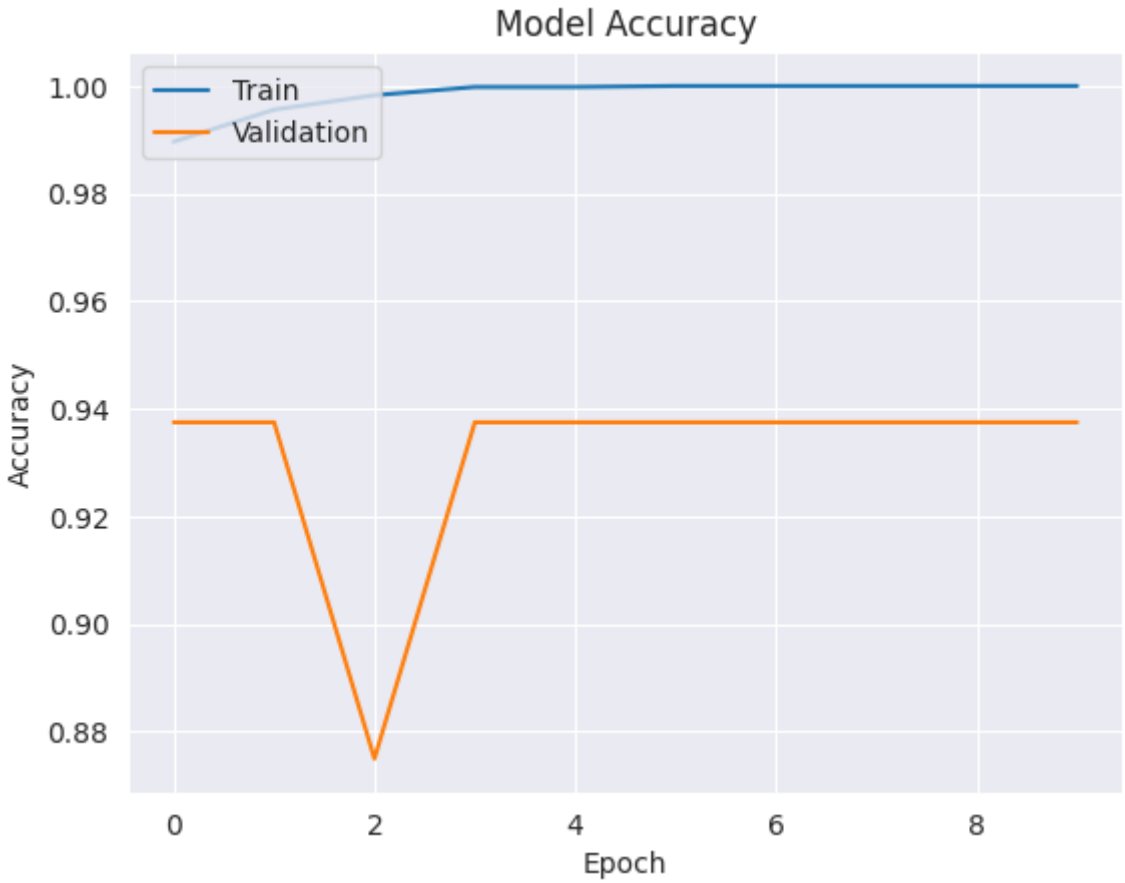
Voici le code de notre model CNN :

```
input = layers.Input((IMG_HEIGHT, IMG_WIDTH, 1), name='input_8')

l1 = Conv2D(8, (3, 3), padding='same', activation='relu', name='conv_1')(input)
l1_mp = MaxPooling2D(pool_size=(2, 2), name='max_pooling_1')(l1)

l2 = Conv2D(16, (3, 3), padding='same', activation='relu', name='conv_2')(l1_mp)
l2_mp = MaxPooling2D(pool_size=(2, 2), name='max_pooling_2')(l2)

l3 = Conv2D(32, (3, 3), padding='same', activation='relu', name='conv_3')(l2_mp)
l3_mp = MaxPooling2D(pool_size=(2, 2), name='max_pooling_3')(l3)

l4 = Conv2D(64, (3, 3), padding='same', activation='relu', name='conv_4')(l3_mp)
l4_mp = MaxPooling2D(pool_size=(2, 2), name='max_pooling_4')(l4)

l5 = Conv2D(128, (3, 3), padding='same', activation='relu', name='conv_5')(l4_mp)
l5_mp = MaxPooling2D(pool_size=(2, 2), name='max_pooling_5')(l5)

flat = Flatten(name='flatten_7')(l5_mp)


l6 = Dense(32, activation='relu', name='dens_21')(flat)
cnn_output = Dense(16, activation='relu', name='dense_22')(l6)

cnn_output = layers.Dense(1, activation='sigmoid',name='dense_23')(cnn_output)
cnn = Model(inputs=input, outputs=cnn_output)

# Compile CNN model
cnn.compile(optimizer=optimizers.Adam(0.001), loss=losses.BinaryCrossentropy(),
metrics=['accuracy'])
```

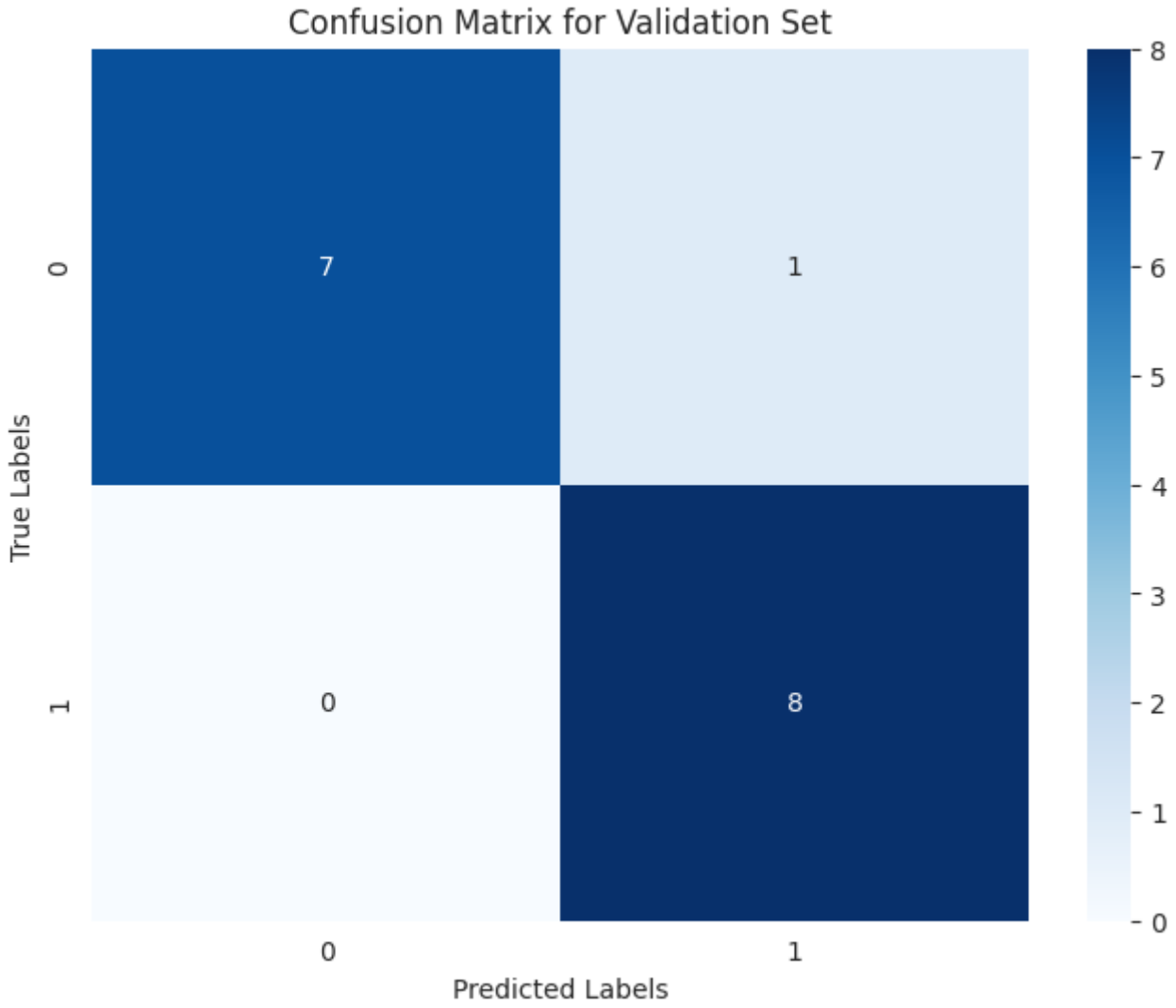
```
Après avoir testé plein de hyperparametre afin d'abaisser la loss de 0.7 qu'on
avait de base, on a modifié la fonction d'activation de la sortie à sigmoid, on a
liassé l'optimizer a 0.001 et avons changé le nombre d'epoch à 10.
```

Avec ces valeurs, on a réussi à avoir ces graphiques :

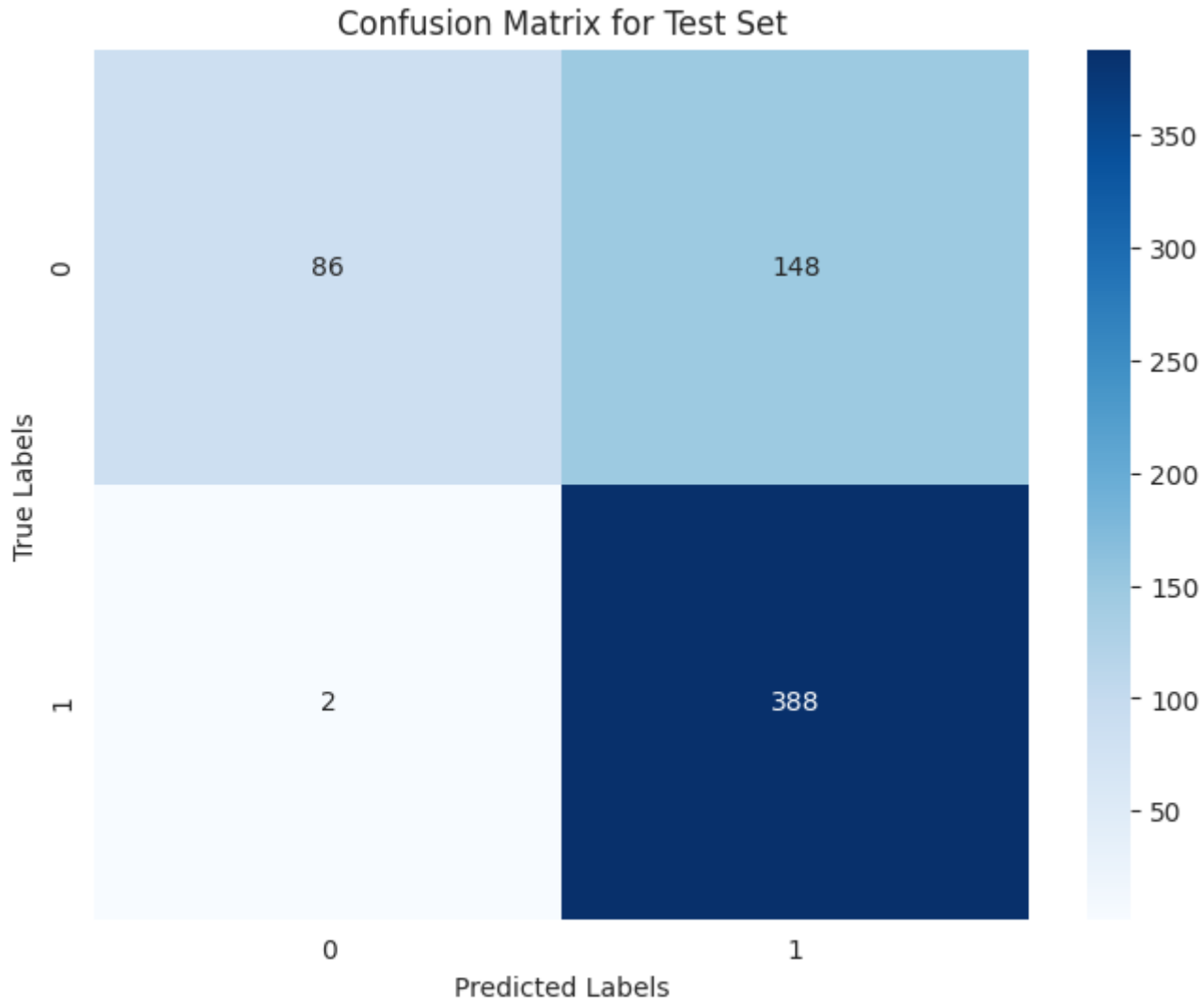




On peut remarquer qu'au niveau de la loss et de l'accuracy deux droite totalement indépendant. Après avoir fait plusieurs test, c'est le mieux qu'on a pu faire. A chaque entrainement les valeurs sur les graphes changeait beaucoup, on n'arrive a expliquer ce comportement très étrange de notre model.



Accuracy: 0.94
Precision: 0.89
Recall: 1.00
F1 Score: 0.94




```
Test Accuracy: 0.76
Test Precision: 0.72
Test Recall: 0.99
Test F1 Score: 0.84

Il n'empeche que lors de la validation notre model montre des reultats plutot bon
avec un f1-score très haut.
Lorsqu'on regarde le test set ont voit que notre model a du mal a preduire s'il y
a effectivement une pneumonie. ce qui est plutot génant car s'il est utilisé dans
le monde medical pourrait entrainer des morts. Le mieux dans ce cas aurait été
qu'il detecte des cas de pneumonie alors qu'il n'y en ait pas. Dans ce cas, on
pourrait effectué un test par un médecin.
```