Practical Work 04 – Deep Neural Networks

auhtors: Rachel Tranchida, Eva Ray

Introduction

Dans ce laboratoire, nous allons en premier lieu explorer trois méthodes différentes pour classer des images de chiffres provenant du jeu de données MNIST : un MLP, un MLP à partir de l'histogramme des gradients (HOG) et un réseau neuronal convolutif (CNN). Dans une seconde partie, nous allons ensuite créer un autre CNN qui devra être capable de classifier des radios thoraciques entre "normal" et "pneumonie". Pour ce faire, nous allons travailler avec le framework `Keras`, qui est une bibliothèque d'outils liés aux réseaux de neuronea de haut niveau, que nous avons déjà utilisé dans le laboratoire précédent.

Buts Pédagogiques

Les buts pédagogiques de ce laboratoire sont les suivants:

- Développer une meilleure compréhension de la différence entre `shallow` et `deep` neural networks.
- Comprendre les principes fondamentaux des réseaux de neurones convolutifs.
- Apprendre les bases du framework `Keras`.

Partie 1

Quel est l'algorithme d'apprentissage utilisé pour optimiser les poids du réseau de neurones?

L'algorithme utilisé pour optimisé les poids est `RMSprop`. RMSprop est un algorithme d'optimisation utilisé pour ajuster les poids d'un réseau de neurones. Il adapte les taux d'apprentissage des poids en utilisant une moyenne mobile des carrés des gradients précédents, ce qui permet une convergence plus rapide et une meilleure performance d'apprentissage.

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

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

où:

- E[g] est la moyenne mobile des gradients au carré
- δc/δw est le gradient de la fonction de coût par rapport au poids
- η est le taux d'apprentissage
- β est le paramètre de la moyenne mobile

Quels sont les paramètres (arguments) utilisés par cet algorithme?

Les paramètres de l'algorithme `RMSprop` peuvent être trouvés dans la documentation de `Keras` et sont les suivants:

```
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=None,
    loss_scale_factor=None,
    gradient_accumulation_steps=None,
    name="rmsprop",
    **kwargs
)
```

Quelle fonction de loss est utilisée?

La fonction de loss utilisée est `categorical_crossentropy`. L'équation de la fonction de loss `categorical_crossentropy` est

```
L = - Σ(y * log(y_pred))
```

où :

- L est la perte (loss) calculée pour un échantillon donné,
- y représente les valeurs cibles réelles (sous forme d'un vecteur codé à chaud ou "one-hot"),
- y_pred représente les probabilités prédites par le modèle pour chaque classe (également sous forme d'un vecteur).

Partie 2

Digit Recognition from Raw Data

Dans cet exercice, nous entraînons un réseaux de neurones en utilisant les données brutes des pixels de la base de données MNIST. Chaque chiffre de la base de données est une image de 28x28 pixels. Il y a 10 classes différentes, qui sont les chiffres de 0 à 9.

Modèle 1

Topologie du modèle

Vous trouvez ci-dessous les paramètres que nous avons choisi d'utiliser pour le premier modèle.

```
model = Sequential()
model.add(Dense(512, input_shape=(784,), activation='sigmoid'))
#model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))
batch_size = 128
n_epoch = 10
```

Pour commencer, nous avons choisi d'ajouter des neurones dans la couche cachée en passant de 2 à 512, afin de donner une meilleure chance au modèle de capturer les motifs des données. Nous travaillons avec des images de chiffres, ce qui est une donnée relativement complexe donc un plus grand nombre de neurones a plus de chance de la comprendre.

Nous avons aussi augmenté le nombre d'epochs en passant de 3 à 10. Cela permet au modèle de passer par plus d'itérations d'apprentissage, ce qui peut améliorer sa capacité à converger vers une solution optimale. Ce choix est pertinent lorsque le modèle est plus complexe, comme dans notre cas avec 512 neurones dans la couche cachée.

Nous avons choisi de ne pas ajouter de couche dopout pour l'instant car le modèle ne montre pas de signe d'overfitting avec les paramètres actuels.

Poids du modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 512)	401,920
dense_5 (Dense)	(None, 10)	5,130

Total params: 407,050 (1.55 MB)

Trainable params: 407,050 (1.55 MB)

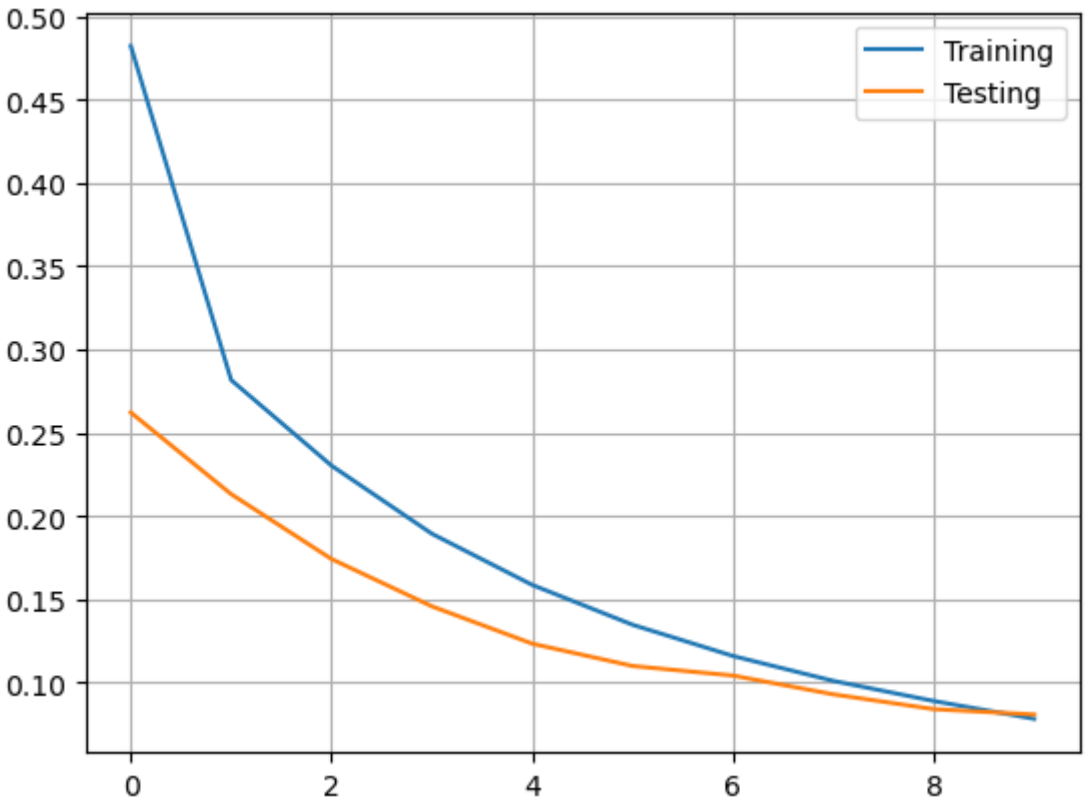
Non-trainable params: 0 (0.00 B)

Pour calculer ces poids manuellement, on peut procéder couche par couche.

- Pour la première couche: (784 * 512) + 512 = 401'920
- Pour la seconde couche (de sortie): (512 * 10) + 10 = 5'130

Pour avoir le nombre total de poids, on additionne le nombre de poids de toutes les couches, ce qui nous donne 407'050 poids.

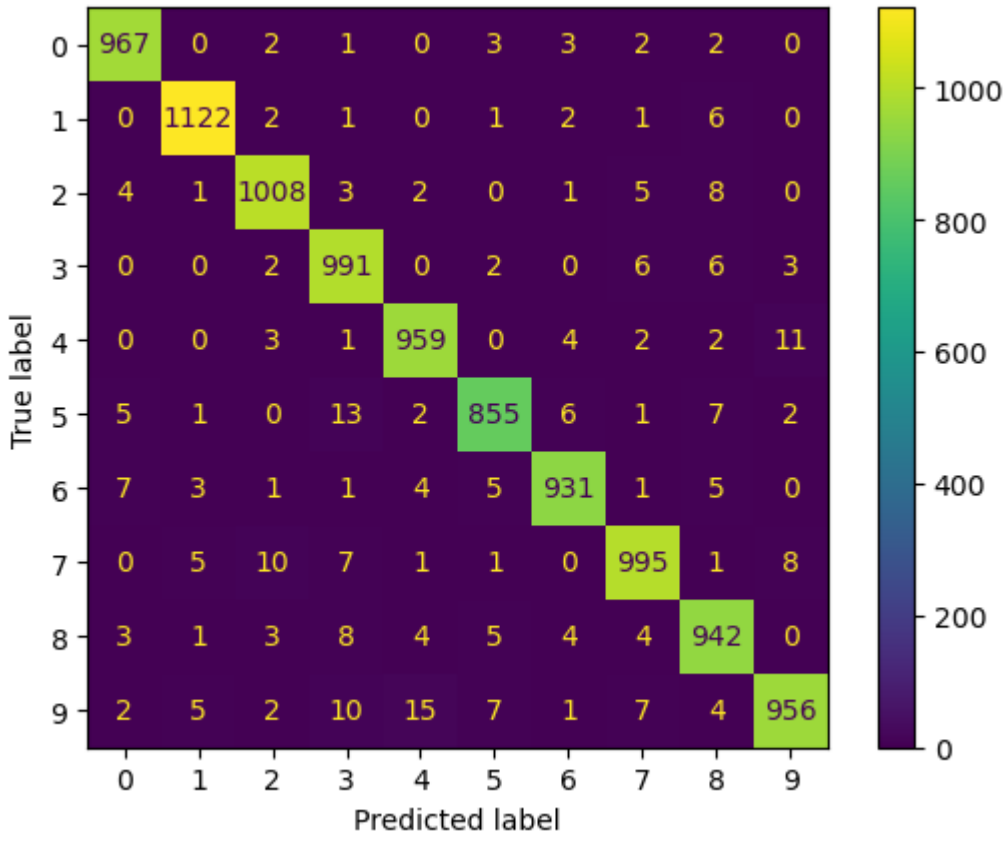
Graphique de l'historique d'entraînement



Performances

Test score: 0.09036532044410706

Test accuracy: 0.972599983215332



Analyse

Modèle 2

Topologie du modèle

```
batch_size = 32
n_epoch = 20
model = Sequential()
model.add(Dense(512, input_shape=(784,), activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))
```

Poids du modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Layer (type)	Output Shape	Param #
dense_19 (Dense)	(None, 512)	401,920
dropout_7 (Dropout)	(None, 512)	0

```
| dense_20 (Dense)                | (None, 10)             |         5,130 |
```

```
Total params: 407,050 (1.55 MB)
```

```
Trainable params: 407,050 (1.55 MB)
```

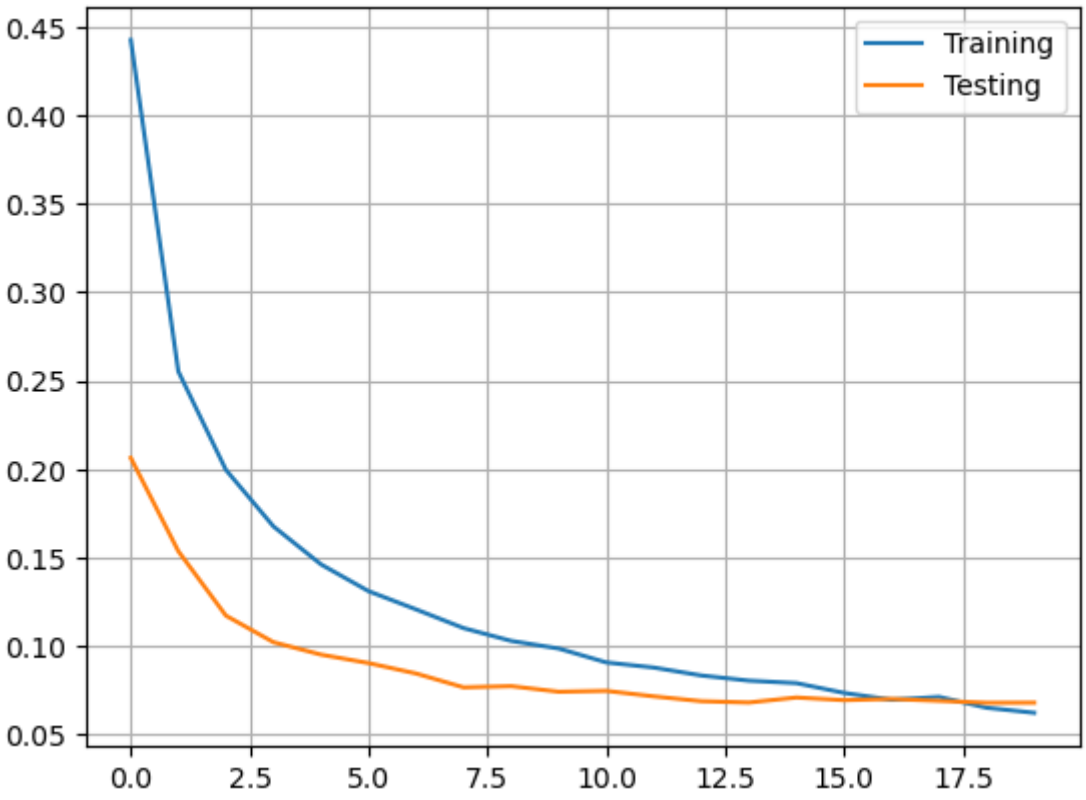
```
Non-trainable params: 0 (0.00 B)
```

Pour calculer ces poids manuellement, on peut procéder couche par couche.

- Pour la première couche: (784 * 512) + 512 = 401'920
- La couche dropout ne possède pas de paramètres à entraîner.
- Pour la seconde couche (de sortie): (512 * 10) + 10 = 5'130

Pour avoir le nombre total de poids, on additionne le nombre de poids de toutes les couches, ce qui nous done 407'050 poids.

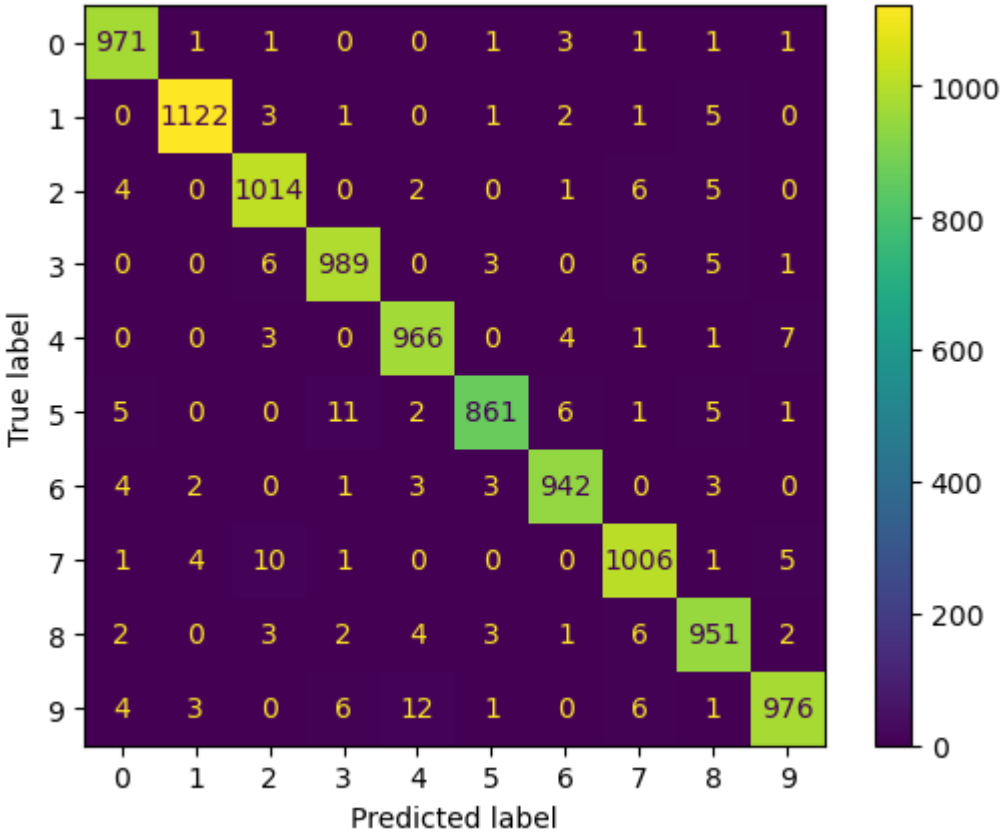
Graphique de l'Historique d'Entraînement



Performances

Test score: 0.07116231322288513

Test accuracy: 0.9797999858856201



Modèle 3

Topologie du modèle

```
batch_size = 128
n_epoch = 30
model = Sequential()
model.add(Dense(512, input_shape=(784,), activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(256, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))
```

Poids du modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Layer (type)	Output Shape	Param #
dense_21 (Dense)	(None, 512)	401,920

dropout_8 (Dropout)	(None, 512)	0
dense_22 (Dense)	(None, 256)	131,328
dropout_9 (Dropout)	(None, 256)	0
dense_23 (Dense)	(None, 10)	2,570

Total params: 535,818 (2.04 MB)

Trainable params: 535,818 (2.04 MB)

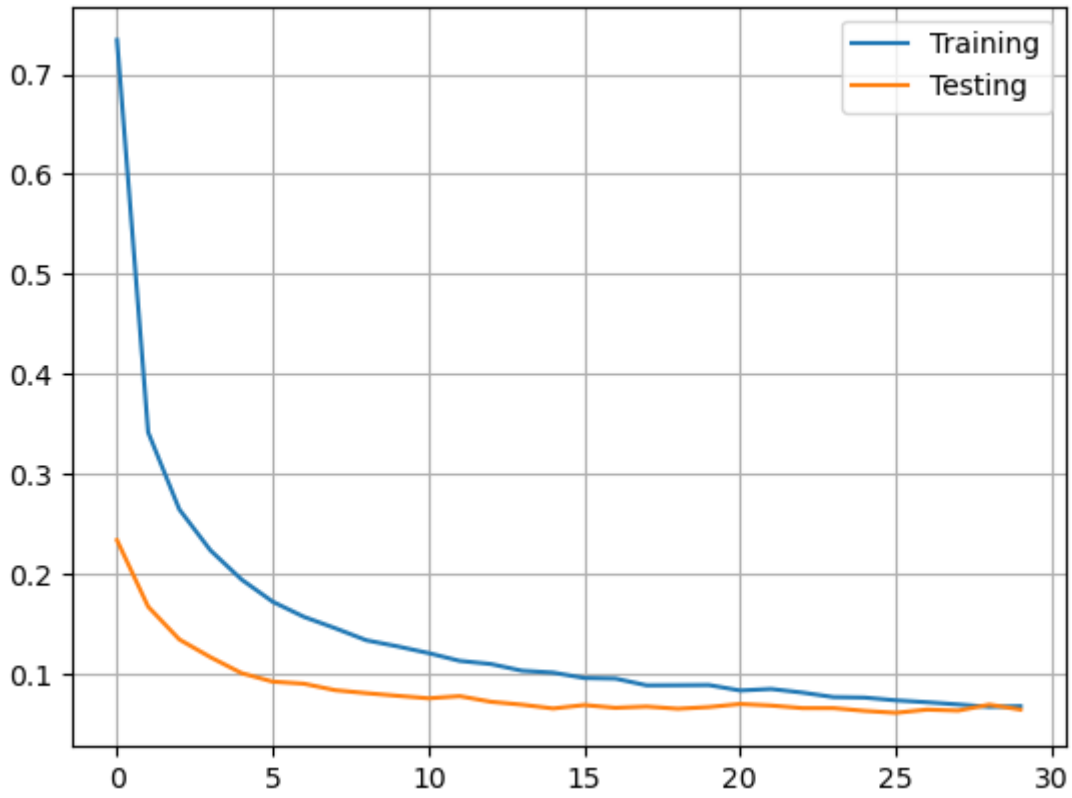
Non-trainable params: 0 (0.00 B)

Pour calculer ces poids manuellement, on peut procéder couche par couche.

- Pour la première couche: (784 * 512) + 512 = 401'920
- La première couche de dropout ne possède pas de paramètres à entraîner.
- Pour la seconde couche: (512 * 256) + 256 = 131'328
- La seconde couche de dropout ne possède pas de paramètres à entraîner.
- Pour la troisième couche (de sortie): (256 * 10) + 10 = 2'570

Pour avoir le nombre total de poids, on additionne le nombre de poids de toutes les couches, ce qui nous done 535'818 poids.

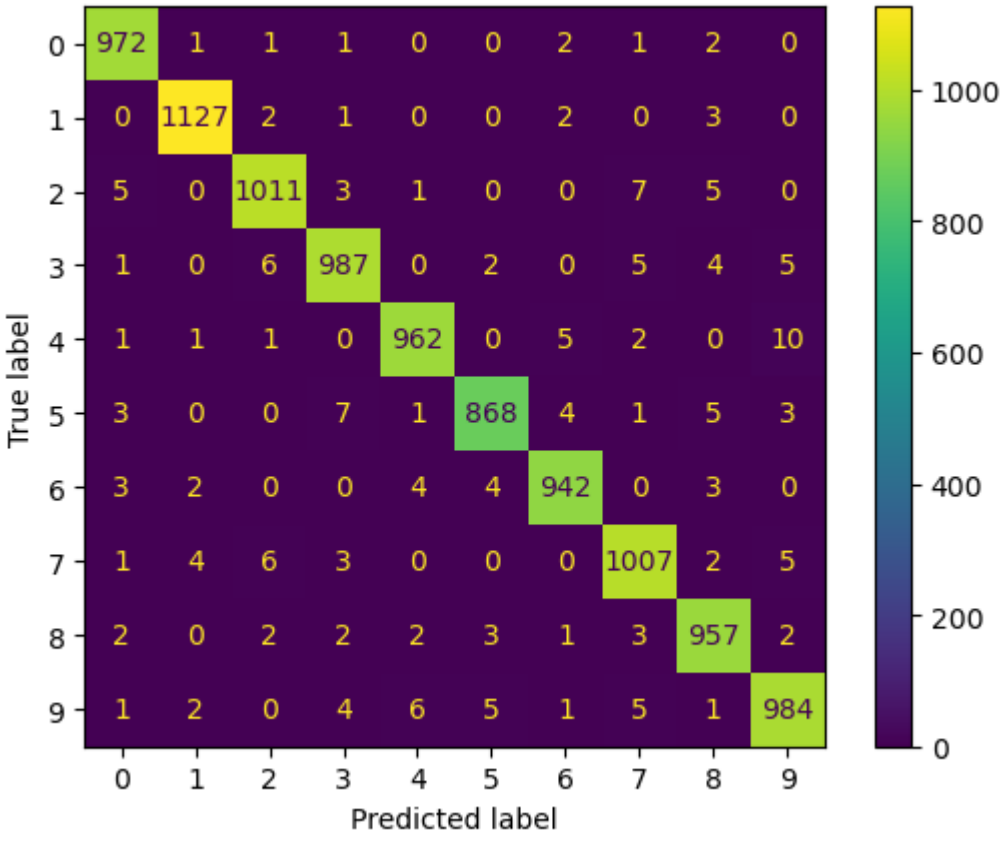
Graphique de l'historique d'entraînement



Performances

Test score: 0.07027491182088852

Test accuracy: 0.9817000031471252



Digit recognition from features of the input data

Modèle 1

Topologie du Modèle

```
batch_size = 128
n_epoch = 50
model = Sequential()
model.add(Dense(64, input_shape=(hog_size,), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))
```

Poids du modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	25,152
dropout (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 10)	650

```
Total params: 25,802 (100.79 KB)
```

```
Trainable params: 25,802 (100.79 KB)
```

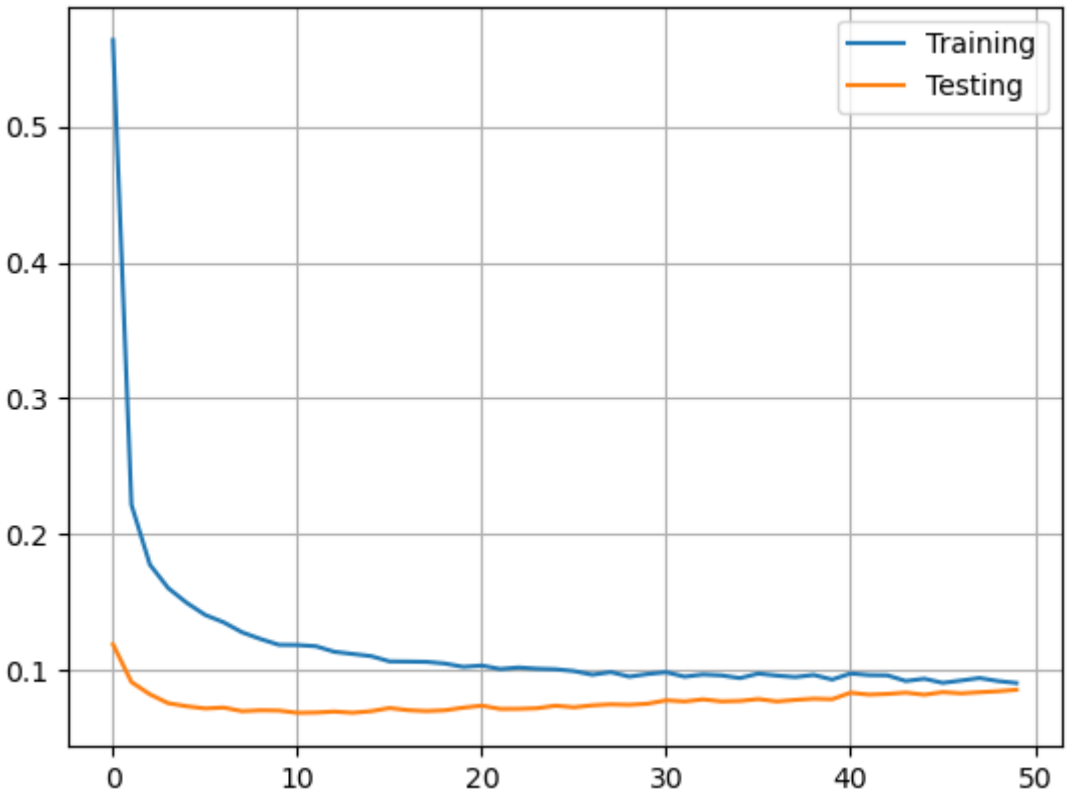
```
Non-trainable params: 0 (0.00 B)
```

Pour calculer ces poids manuellement, on peut procéder couche par couche.

- Pour la première couche: (392 * 64) + 64 = 25'152, où hog_size = 392
- La première couche de dropout ne possède pas de paramètres à entraîner.
- Pour la seconde couche: (640 * 10) + 10 = 650

Pour avoir le nombre total de poids, on additionne le nombre de poids de toutes les couches, ce qui nous done 25'802 poids.

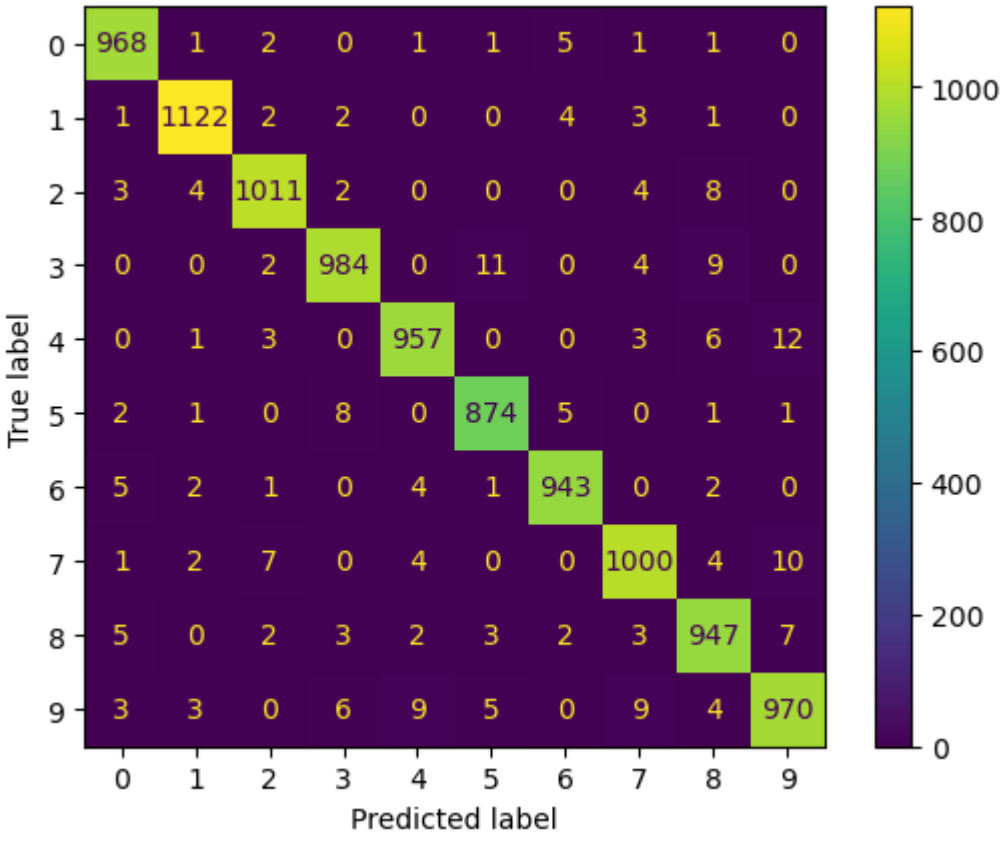
Graphique de l'historique d'entraînement



Performances

Test score: 0.09391739219427109

Test accuracy: 0.9775999784469604



Anlayse

Modèle 2

Topologie du Modèle

```
batch_size = 128
n_epoch = 20
model = Sequential()
model.add(Dense(64, input_shape=(hog_size,), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(n_classes, activation='softmax'))

n_orientations = 16
pix_p_cell = 4
hog_size = int(height * width * n_orientations / (pix_p_cell * pix_p_cell))// =784
```

Poids du modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Layer (type)	Output Shape	Param #
dense_18 (Dense)	(None, 64)	50,240
dropout_7 (Dropout)	(None, 64)	0
dense_19 (Dense)	(None, 10)	650

Total params: 50,890 (198.79 KB)

Trainable params: 50,890 (198.79 KB)

Non-trainable params: 0 (0.00 B)

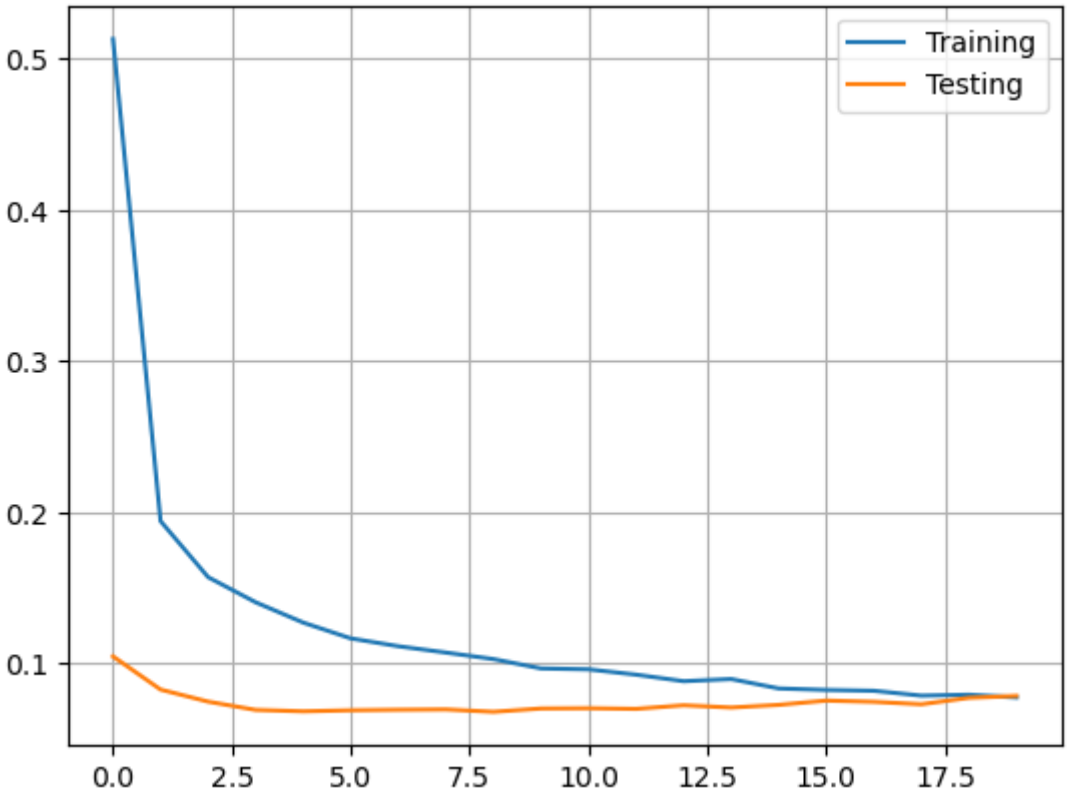
Pour calculer ces poids manuellement, on peut procéder couche par couche.

- Pour la première couche: (784 * 64) + 64 = 50'240, où hog_size = 784
- La première couche de dropout ne possède pas de paramètres à entraîner.

- Pour la seconde couche: (640 * 10) + 10 = 650

Pour avoir le nombre total de poids, on additionne le nombre de poids de toutes les couches, ce qui nous done 50'890 poids.

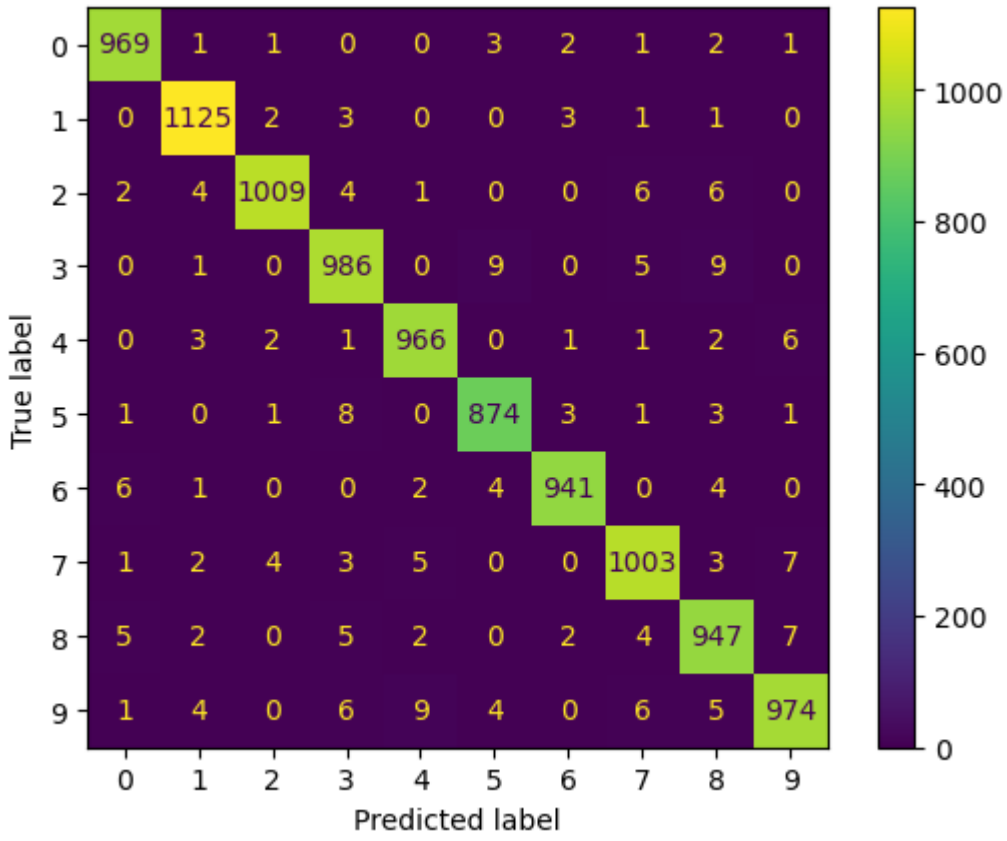
Graphique de l'historique d'entraînement



Performances

Test score: 0.07377872616052628

Test accuracy: 0.9793999791145325



Modèle 3

Topologie du Modèle

Poids du Modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Graphique de l'historique d'entraînement

Performances

Convolutional neural network digit recognition

Modèle 1

Topologie du Modèle

Poids du Modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Graphique de l'historique d'entraînement

Performances

Modèle 2

Topologie du Modèle

Poids du Modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Graphique de l'historique d'entraînement

Performances

Modèle 4

Topologie du Modèle

Poids du Modèle

Vous trouverez ci-dessous le résumé des poids et paramètres du modèle donné par la méthode `model.summary()` de `Keras`.

Graphique de l'historique d'entraînement

Performances

Partie 3

Les modèles CNN sont plus profonds (ont plus de couches), ont-ils plus de poids que les modèles shallow?

En général, les réseaux de neurones convolutifs (CNN) plus profonds ont tendance à avoir plus de poids que les modèles shallow. Cela est dû au fait que les CNN plus profonds ont généralement un plus grand nombre de couches, et chaque couche est composée de plusieurs filtres qui contiennent des poids.

Exemple

A VERIFIER CEST FAIT PAR CLAUDE

Supposons que nous ayons un CNN superficiel avec une seule couche de convolution suivie d'une couche entièrement connectée. La couche de convolution a 16 filtres de taille 3x3, et la couche entièrement connectée a 128 neurones. Dans ce cas, le nombre de poids dans la couche de convolution serait de 16 * (3 * 3) = 144, et le nombre de poids dans la couche entièrement connectée serait (16 * 3 * 3) * 128 = 73 728. Ainsi, le nombre total de poids dans le CNN superficiel serait de 73 728 + 144 = 73 872.

Maintenant, considérons un CNN plus profond avec trois couches de convolution, chacune suivie d'une couche de mise en commun (pooling), puis d'une couche entièrement connectée. Chaque couche de convolution a 32 filtres de taille 3x3, et la couche entièrement connectée a 256 neurones. Dans ce cas, le

nombre de poids dans chaque couche de convolution serait de 32 * (3 * 3) = 288, et le nombre de poids dans la couche entièrement connectée serait (32 * 3 * 3) * 256 = 294 912. Ainsi, le nombre total de poids dans le CNN plus profond serait (288 * 3) + 294 912 = 295 776.

Partie 4