L'apprentissage profond avec Python de François Chollet

https://github.com/fchollet/deep-learning-with-python-notebooks/tree/master

Loïc ALAVOINE

3 janvier 2026

Table des matières

Chapitre 1

Les éléments mathématiques constitutifs des réseaux neuronaux

1.1 Code utilisé dans le terminal Windows du dossier dans lequel on travaille. On suppose que Python et Pandoc sont installés.

python -m venv DeepLearningBook
.\DeepLearningBook\Scripts\activate
python.exe -m pip install –upgrade pip
pip install notebook
pip install ipykernel
pip install keras
pip install tensorflow
pip install matplotlib
jupyter notebook

1.2 Un premier aperçu d'un réseau neuronal

1.2.1 Chargement du jeu de données MNIST dans Keras

[1]:
```
from keras.datasets import mnist
```

```
C:\Users\loica\OneDrive\Desktop\Deep Learning Chollet\DeepLearningBook\Lib\site-
packages\keras\src\export\tf2onnx_lib.py:8: FutureWarning: In the future
`np.object` will be defined as the corresponding NumPy scalar.
  if not hasattr(np, "object"):
```

[2]:
```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

[3]:
```
train_images.shape
```

[3]:
```
(60000, 28, 28)
```

[4]:
```
len(train_labels)
```

[4]:
```
60000
```

[5]:
```
train_labels
```

[5]:
```
array([5, 0, 4, ..., 5, 6, 8], shape=(60000,), dtype=uint8)
```

[6]:
```
test_images.shape
```

[6]:
```
(10000, 28, 28)
```

[7]:
```
len(test_labels)
```

[7]:
```
10000
```

[8]:
```
test_labels
```

[8]:
```
array([7, 2, 1, ..., 4, 5, 6], shape=(10000,), dtype=uint8)
```

1.2.2 L'architecture du réseau

[9]:
```
import keras
```

[10]:
```
from keras import layers
```

[11]:
```
model = keras.Sequential(
    [
        layers.Dense(512, activation="relu"),
        layers.Dense(10, activation="softmax"),
    ]
)
```

1.2.3 L'étape de compilation

[12]:
```
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"],
)
```

1.2.4 Préparation des données (image)

[13]:
```
train_images = train_images.reshape((60000, 28 * 28))
```

[14]:
```
train_images = train_images.astype("float32") / 255
```

[15]:
```
test_images = test_images.reshape((10000, 28 * 28))
```

[16]:
```
test_images = test_images.astype("float32") / 255
```

1.2.5 Ajustement du modèle

[17]:
```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469  4s 6ms/step -
accuracy: 0.9248 - loss: 0.2671
Epoch 2/5
469/469  3s 6ms/step -
accuracy: 0.9685 - loss: 0.1099
Epoch 3/5
469/469  3s 7ms/step -
accuracy: 0.9789 - loss: 0.0714
Epoch 4/5
469/469  3s 7ms/step -
accuracy: 0.9849 - loss: 0.0516
Epoch 5/5
469/469  3s 7ms/step -
accuracy: 0.9879 - loss: 0.0398
```

[17]:
```
<keras.src.callbacks.history.History at 0x19cc3e74980>
```

1.2.6 Utiliser le modèle pour faire des prédictions

[18]:
```
test_digits = test_images[0:10]
```

[19]:
```
predictions = model.predict(test_digits)
```

```
1/1  0s 76ms/step
```

[20]:
```
predictions[0]
```

[20]:
```
array([6.1895157e-08, 7.3658493e-09, 1.2752834e-05, 1.0990108e-05,
       1.7984386e-10, 2.7613081e-07, 9.1607633e-12, 9.9997544e-01,
       1.2313654e-07, 3.2242909e-07], dtype=float32)
```

[21]:
```
predictions[0].argmax()
```

[21]:
```
np.int64(7)
```

[22]:
```
predictions[0][7]
```

[22]:
```
np.float32(0.99997544)
```

[23]:
```
test_labels[0]
```

[23]:
```
np.uint8(7)
```

1.2.7 Évaluation du modèle sur de nouvelles données

[24]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
313/313  1s 2ms/step -
accuracy: 0.9779 - loss: 0.0692
```

[25]:
```
print(f"test_acc: {test_acc}")
```

```
test_acc: 0.9779000282287598
```

1.3 Représentations de données pour les réseaux neuronaux

1.3.1 Scalaires (tenseurs de rang 0)

[26]:
```
import numpy as np
```

[27]:
```
x = np.array(12)
```

[28]:
```
x
```

[28]:
```
array(12)
```

[29]:
```
x.ndim
```

[29]:
```
0
```

1.3.2 Vecteurs (tenseurs de rang 1)

[30]:
```
x = np.array([12, 3, 6, 14, 7])
```

[31]:
```
x
```

[31]:
```
array([12,  3,  6, 14,  7])
```

[32]:
```
x.ndim
```

[32]:
```
1
```

1.3.3 Matrices (tenseurs de rang 2)

[33]:
```
x = np.array([[5, 78, 2, 34, 0],
              [6, 79, 3, 35, 1],
              [7, 80, 4, 36, 2]])
```

[34]:
```
x
```

[34]:
```
array([[ 5, 78,  2, 34,  0],
       [ 6, 79,  3, 35,  1],
```

```
       [ 7, 80,  4, 36,  2]])
```

[35]:
```
x.ndim
```

[35]:
```
2
```

1.3.4 Tenseurs de rang 3 et tenseurs de rang supérieur

[36]:
```
x = np.array([[[5, 78, 2, 34, 0],
               [6, 79, 3, 35, 1],
               [7, 80, 4, 36, 2]],
              [[5, 78, 2, 34, 0],
               [6, 79, 3, 35, 1],
               [7, 80, 4, 36, 2]],
              [[5, 78, 2, 34, 0],
               [6, 79, 3, 35, 1],
               [7, 80, 4, 36, 2]]])
```

[37]:
```
x
```

[37]:
```
array([[[ 5, 78,  2, 34,  0],
        [ 6, 79,  3, 35,  1],
        [ 7, 80,  4, 36,  2]],

       [[ 5, 78,  2, 34,  0],
        [ 6, 79,  3, 35,  1],
        [ 7, 80,  4, 36,  2]],

       [[ 5, 78,  2, 34,  0],
        [ 6, 79,  3, 35,  1],
        [ 7, 80,  4, 36,  2]]])
```

[38]:
```
x.ndim
```

[38]:
```
3
```

1.3.5 Caractéristiques principales

[39]:
```
from keras.datasets import mnist
```

[40]:
```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

[41]:
```
train_images.ndim
```

[41]:
```
3
```

[42]:
```
train_images.shape
```

[42]:
```
(60000, 28, 28)
```

[43]:
```
train_images.dtype
```

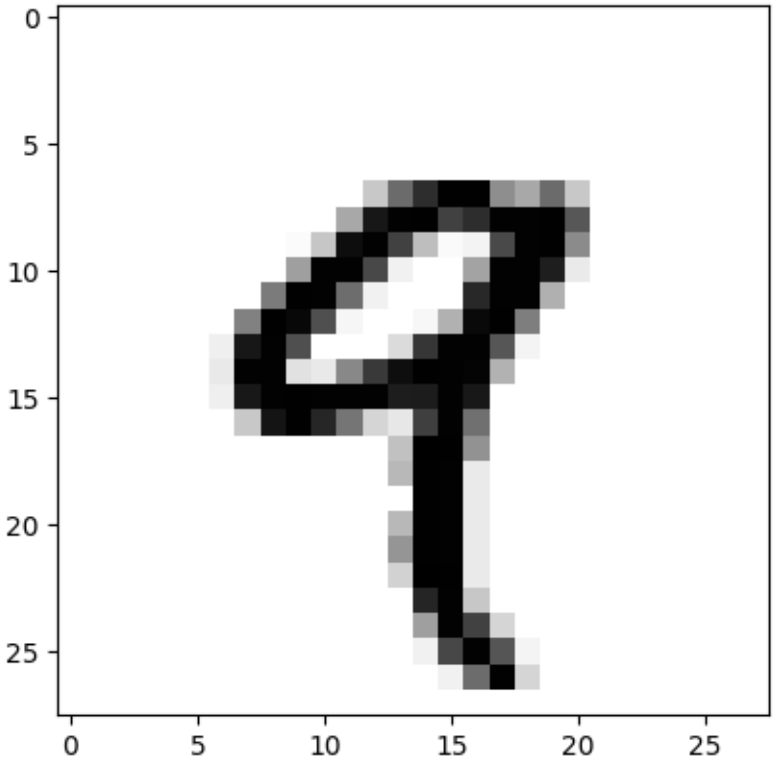
[43]:
```
dtype('uint8')
```

1.3.5.1 Affichage du cinquième chiffre

[44]:
```
import matplotlib.pyplot as plt
```

[45]:
```
digit = train_images[4]
```

[46]:
```
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```



[47]:
```
train_labels[4]
```

[47]:
```
np.uint8(9)
```

1.3.6 Manipulation des tenseurs dans NumPy

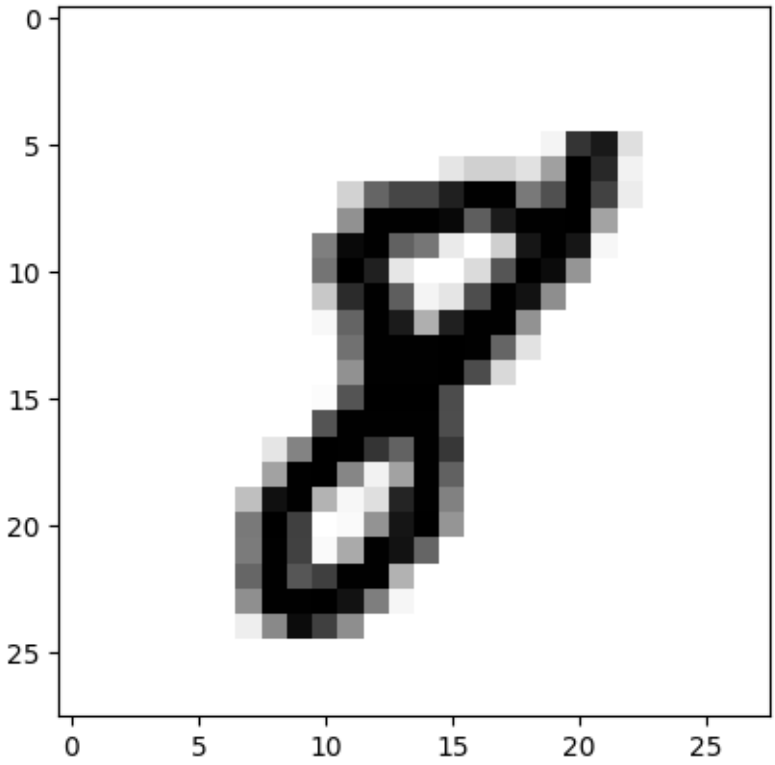
[48]:
```
my_slice = train_images[10:100]
```

[49]:
```
my_slice.shape
```

[49]:
```
(90, 28, 28)
```

[50]:
```
import matplotlib.pyplot as plt
```

[51]:
```
plt.imshow(my_slice[7], cmap=plt.cm.binary)
plt.show()
```



1.3.6.1 Equivalent à l'exemple précédent

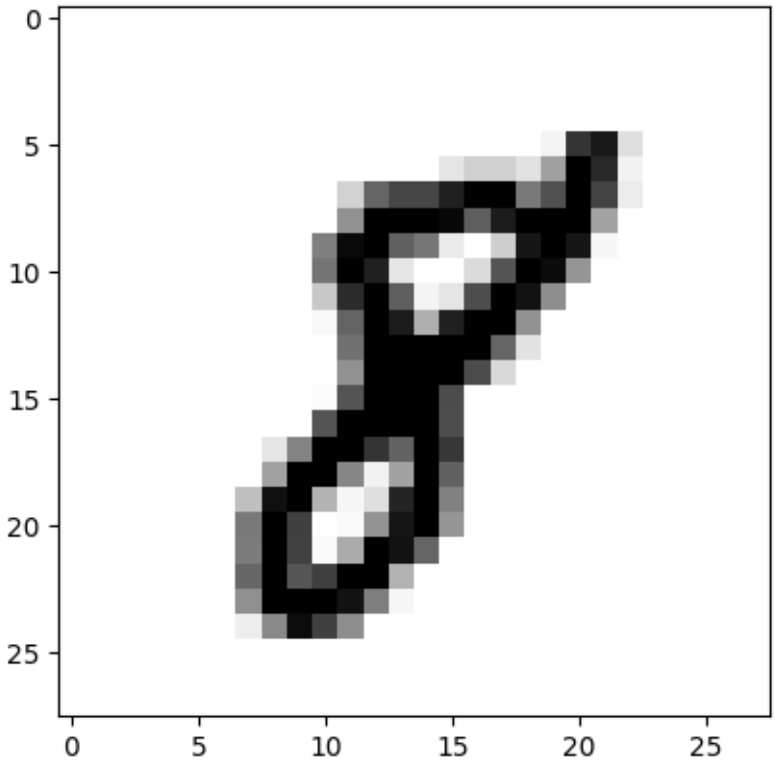
[52]:
```
my_slice = train_images[10:100, :, :]
```

[53]:
```
my_slice.shape
```

[53]:
```
(90, 28, 28)
```

[54]:
```
import matplotlib.pyplot as plt
```

[55]:
```
plt.imshow(my_slice[7], cmap=plt.cm.binary)
plt.show()
```



1.3.6.2 Egalement équivalent à l'exemple précédent

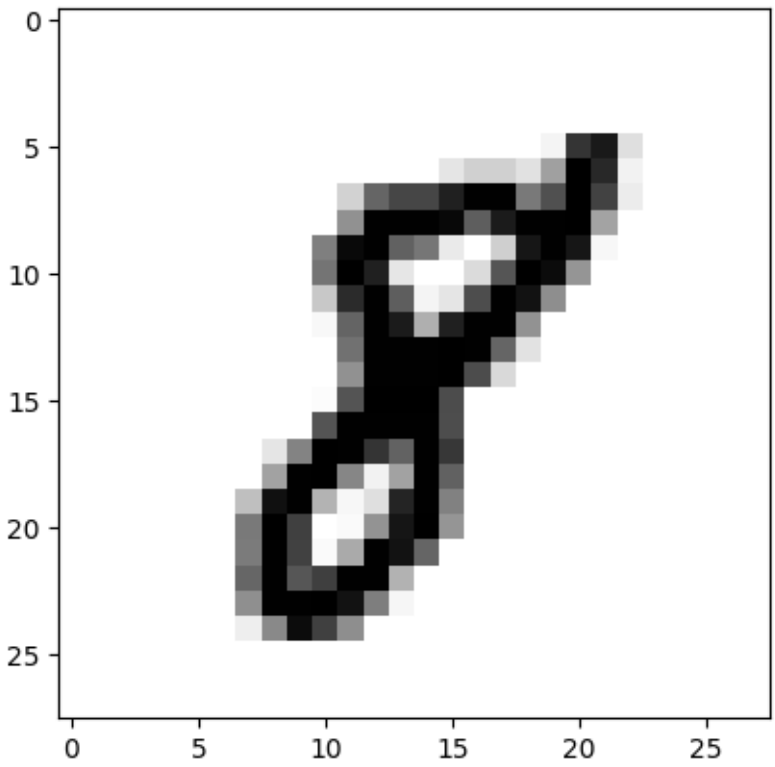
[56]:
```
my_slice = train_images[10:100, 0:28, 0:28]
```

[57]:
```
my_slice.shape
```

[57]:
```
(90, 28, 28)
```

[58]:
```
import matplotlib.pyplot as plt
```

[59]:
```
plt.imshow(my_slice[7], cmap=plt.cm.binary)
plt.show()
```

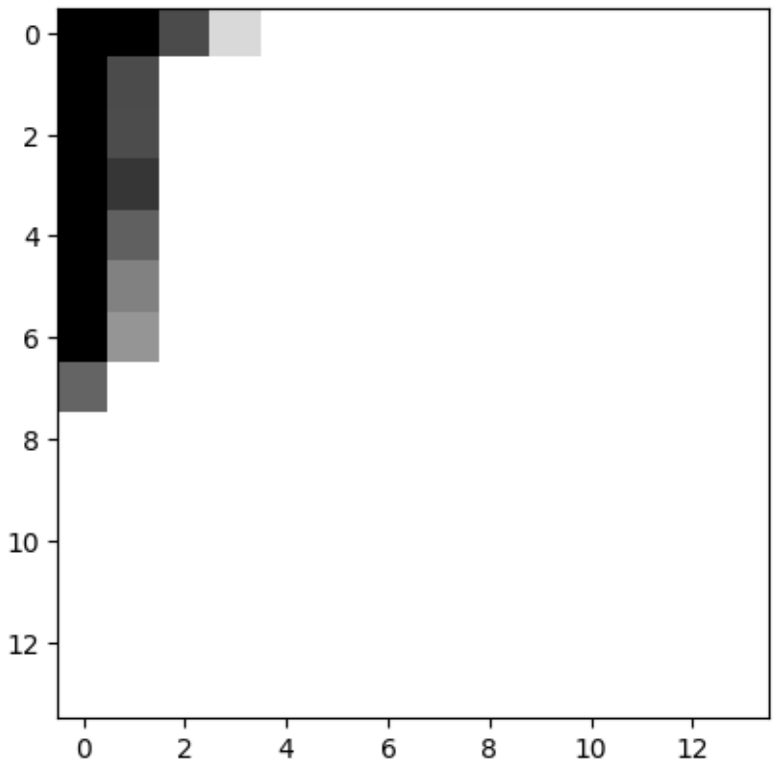


1.3.6.3 Saucissonnage

[60]:
```
my_slice = train_images[:, 14:, 14:]
```

[61]:
```
import matplotlib.pyplot as plt
```

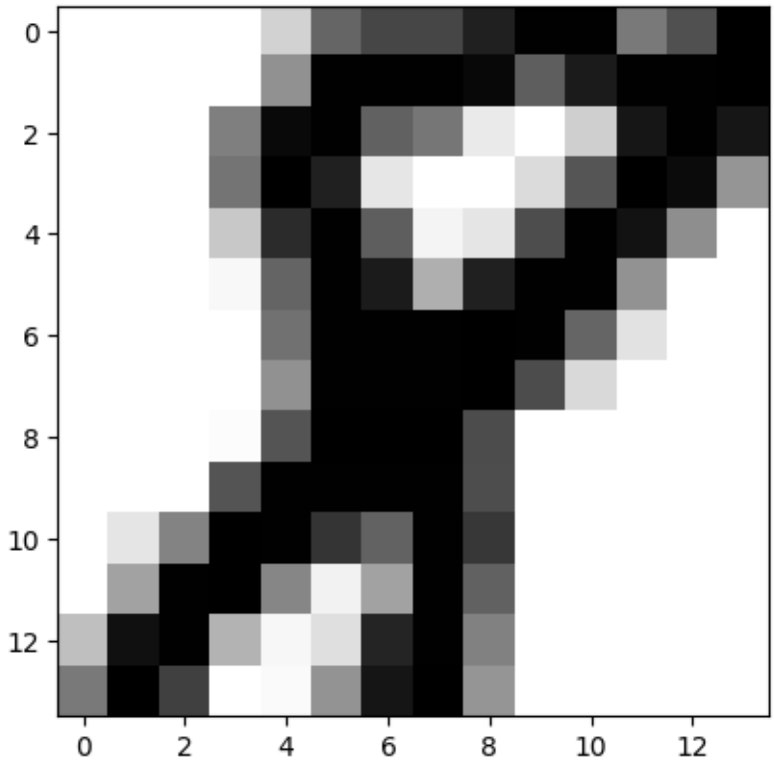
[62]:
```
plt.imshow(my_slice[17], cmap=plt.cm.binary)
plt.show()
```



[63]:
```
my_slice = train_images[:, 7:-7, 7:-7]
```

[64]:
```
import matplotlib.pyplot as plt
```

[65]:
```
plt.imshow(my_slice[17], cmap=plt.cm.binary)
plt.show()
```



1.3.7 La notion de lots de données

[66]:
```
batch = train_images[:128]
```

[67]:
```
batch = train_images[128:256]
```

[68]:
```
n = 3
batch = train_images[128 * n : 128 * (n + 1)]
```

1.3.8 Exemples concrets de tenseurs de données

[ ]:
```

```