CrashCoursesDL_FNN_CNN_January24

January 9, 2024

#

Deep Learning Crash Course with tensorflow

Abdulrahman ALLOUCHE / 9 January 2024

Tensorflow is a free software Deep Learning library, developped by Google for the Python programming language.

https://www.tensorflow.org/

0.1 Set up TensorFlow

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import warnings
warnings.filterwarnings("ignore")
print("TensorFlow version:", tf.__version__)
```

```
2024-01-09 10:39:59.180872: I tensorflow/core/util/port.cc:110] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-01-09 10:39:59.182340: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2024-01-09 10:39:59.209709: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2024-01-09 10:39:59.210525: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
2024-01-09 10:39:59.623397: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT

TensorFlow version: 2.12.0
```

0.2 1.Suppervised DL using a Full Neural Networks

0.2.1 Load a dataset

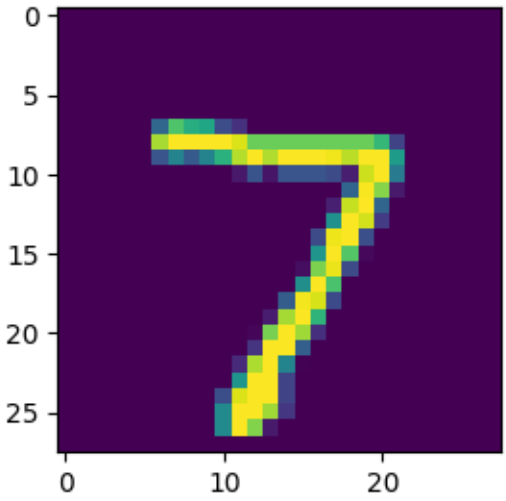
[2]:
```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

[3]:
```
print("xtrain shape =", x_train.shape)
```

```
xtrain shape = (60000, 28, 28)
```

[4]:
```
plt.figure(figsize=(3,3))
plt.imshow(x_test[0])
```

[4]:
```
<matplotlib.image.AxesImage at 0x14aaa5aa47c0>
```



0.2.2 Build a model

utiliser le model nommé "séquential"

des réseau de neuronnes en série

[5]:
```
warnings.filterwarnings("ignore")
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
model.summary()
```

to flatten a matrix into a vector

definir le nombre de couches

fonction pour rendre le fonction non-linéaire, 128 nombre de neuronne du layer

shift+tab=> autre fonction

ne pas faire de overfitting (0.2 est un pourcentage)

nombre de neuronne dans une layer qui ne sont pas cachés

```
Model: "sequential"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
2024-01-09 10:40:05.244644: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_gpu_executor.cc:996]
successful NUMA node read from SysFS had negative value (-1), but there must be
at least one NUMA node, so returning NUMA node zero. See more at
https://github.com/torvalds/linux/blob/v6.0/Documentation/ABI/testing/sysfs-bus-
pci#L344-L355
2024-01-09 10:40:05.246108: W
tensorflow/core/common_runtime/gpu/gpu_device.cc:1956] Cannot dlopen some GPU
libraries. Please make sure the missing libraries mentioned above are installed
properly if you would like to use GPU. Follow the guide at
https://www.tensorflow.org/install/gpu for how to download and setup the
required libraries for your platform.
Skipping registering GPU devices…
```

[6]:
```
predictions = model(x_train[:1]).numpy()
predictions
```

on donne le premier élément

[6]:
```
array([[-2.6300317e-01,  3.7921870e-01, -1.7462549e-04, -1.5340197e-01,
        -8.2738286e-01,  2.8128016e-01, -1.1971046e-01, -4.2450556e-01,
         3.5356525e-01, -1.8688932e-01]], dtype=float32)
```

The **tf.nn.softmax** function converts these logits to probabilities for each class:

transform values into proba

[7]:
```
tf.nn.softmax(predictions).numpy()
```

[7]:
```
array([[0.07970797, 0.15150087, 0.10366847, 0.08894076, 0.04533093,
        0.13736653, 0.09198837, 0.06782067, 0.14766377, 0.08601169]],
      dtype=float32)
```

losses.SparseCategoricalCrossentropy takes a vector of logits and a True index and returns a scalar loss for each example. This loss is equal to the negative log probability of the true class: **The loss is zero if the model is sure of the correct class.**

on minimize la fonction de l'entropie

ajout de parameter: example: True

[8]:
```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

[9]:
```
loss_fn(y_train[:1], predictions).numpy()
```

[9]:
```
1.9851025
```

0.2.3 Before you start training, configure and compile the model

[10]:
```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

define the optimizer
define the fonction Loss that we are going to minimize
define the metric that we are going to optimize with respect to
MSE, R^2....

0.2.4 Train and evaluate your model

batch= paquets de donnés, que l'on donne au model, par defaut 32 données

[11]:
```
model.fit(x_train, y_train, epochs=5)
```

epochs=nombre d'iterations, le training se fait par batch, ici nombre de batch est epochs, ex: si 60 000 données, alors epochs= #donnes/32

```
Epoch 1/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2980 -
accuracy: 0.9140
Epoch 2/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1459 -
accuracy: 0.9574
Epoch 3/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1073 -
accuracy: 0.9668
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0882 -
accuracy: 0.9729
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0762 -
accuracy: 0.9765
```

ou model.fit(x_train, y_train, epochs=5, validation_split=0.2)

[11]:
```
<keras.callbacks.History at 0x14aa71f5d960>
```

The Model.evaluate method checks the models performance, usually on a "Validation-set" or "Test-set".

[12]:
```
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.0782 - accuracy: 0.9769 - 177ms/epoch - 566us/step
```

[12]:
```
[0.07823929190635681, 0.9768999814987183]
```

If you want your model to return a probability, you can wrap the trained model, and attach the softmax to it:

[13]:
```
probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
```

```
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
# patience: valeur d'iteration pdt lequel le output ne change pas
history = model.fit(x_train, y_train, epochs=20, callbacks=[callbacks], validation_split=0.2)

len(history.history['loss'])

model.save # saugarder une fitting précise avec les parametres trouvés
model.load
```

```
])
```

[14]:
```
probability_model(x_test[:1])
```

[14]:
```
<tf.Tensor: shape=(1, 10), dtype=float32, numpy=
array([[6.8671959e-08, 1.1341825e-07, 8.4587737e-06, 5.8121228e-04,
        7.7612516e-10, 4.7873891e-07, 5.1690284e-12, 9.9940526e-01,
        2.1167273e-06, 2.3484934e-06]], dtype=float32)>
```

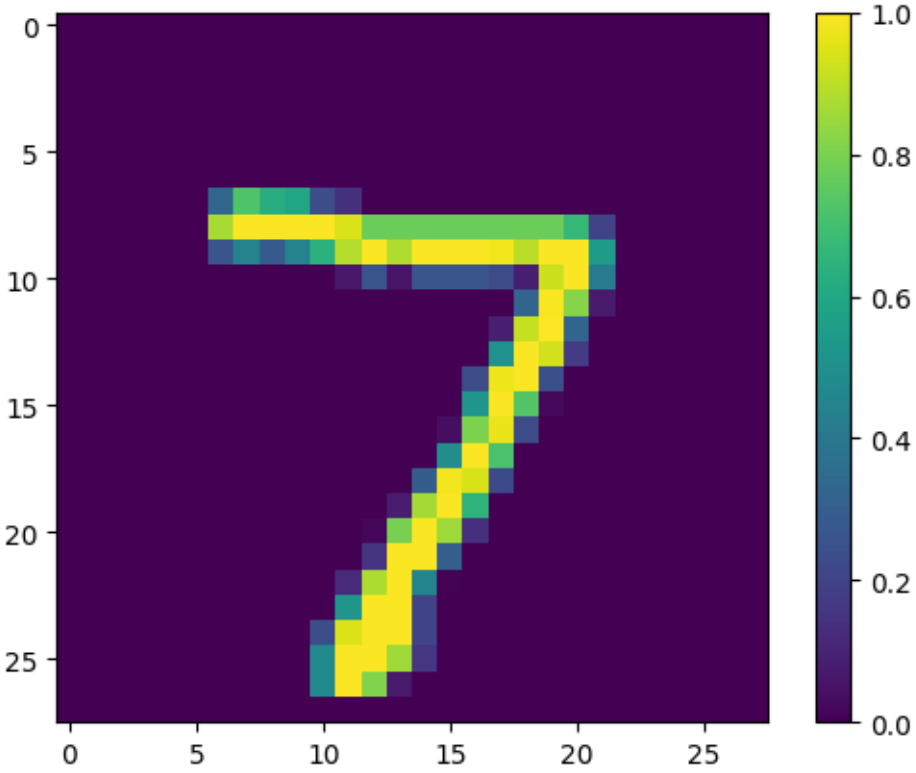
[15]:
```
#Print predicted values vs real values
def printErrValues(x_test,y_test, model):
    probability_model = tf.keras.Sequential([
      model,
      tf.keras.layers.Softmax()
    ])
    class_names = [0, 1, 2, 3, 4,5, 6, 7, 8, 9]
    #n=1000
    n=y_test.shape[0]
    class_namesp = tf.constant([class_names]*n)
    class_namesp = tf.reshape(class_namesp, [n*10])
    pm=probability_model(x_test[0:n])
    t=(pm>=tf.reduce_max(pm))
    #print(t)
    t = tf.reshape(t, [n*10])
    #print(t.shape)
    #print(class_namesp.shape)
    #print(class_namesp[t].shape)


    #print(class_namesp[t].numpy())
    #print(y_test[0:n+1])
    y_pred= class_namesp[t].numpy()
    nerr=0
    for i in range(y_pred.shape[0]):
        if y_test[i] != y_pred[i]:
            nerr += 1
            #print("#index = ",i,' y_test=',y_test[i], ' y_predic=',y_pred[i])
    print("nerr/n=",nerr, " / ",n)
```

[16]:
```
#Print predicted values vs real values
printErrValues(x_test,y_test, model)
```

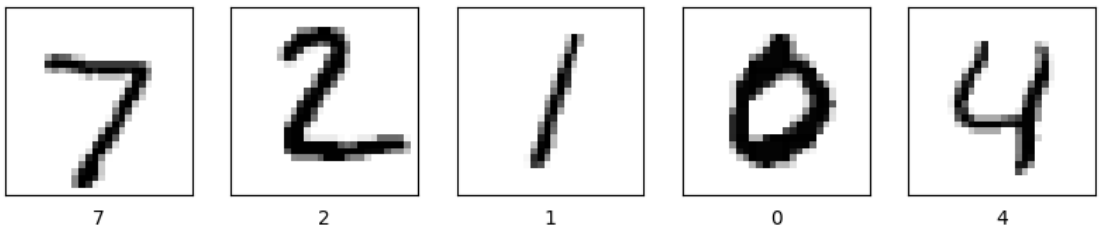
```
nerr/n= 226  /  10000
```

[17]:
```
plt.figure()
plt.imshow(x_test[0])
plt.colorbar()
plt.grid(False)
```

```
plt.show()
```



[18]:
```
class_names = ['0', '1', '2', '3', '4','5', '6', '7', '8', '9']

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_test[i]])
plt.show()
```



0.2.5 Build a model with Functional API

autre façon de faire, des réseaux de neuronnes + complexes

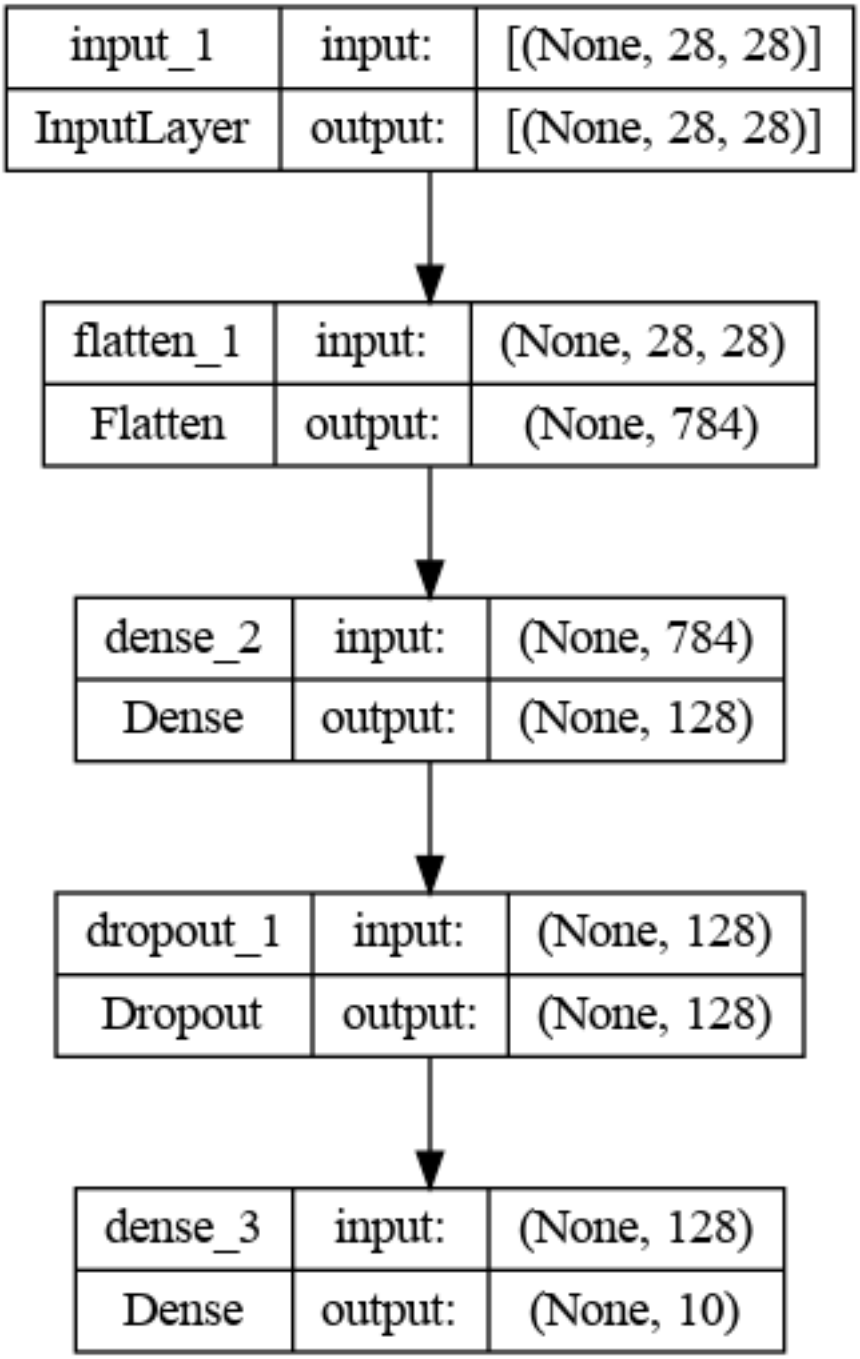
[19]:
```
inputs = tf.keras.Input(shape=(28,28))
flatten= tf.keras.layers.Flatten(input_shape=(28, 28))
x=flatten(inputs)
dense = tf.keras.layers.Dense(128, activation="relu")
x = dense(x)
dropout=tf.keras.layers.Dropout(0.2)
x = dropout(x)
outputs = tf.keras.layers.Dense(10)(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs, name="mnist_model")
```

[20]:
```
model.summary()
```

```
Model: "mnist_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 28, 28)]          0

 flatten_1 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 128)               100480

 dropout_1 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

[21]:
```
#tf.keras.utils.plot_model(model)
#tf.keras.utils.plot_model(model, "my_first_model_with_shape_info.png",␣
 ↪show_shapes=True)
tf.keras.utils.plot_model(model, show_shapes=True)
```

[21]:



[22]:
```
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(),
    metrics=["accuracy"],
)
```

[23]:
```
#reload data if necessary
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

[24]:
```
history = model.fit(x_train, y_train, batch_size=64, epochs=5,␣
 ↪validation_split=0.2)

test_scores = model.evaluate(x_test, y_test, verbose=2)
print("Test loss:", test_scores[0])
print("Test accuracy:", test_scores[1])
```

```
Epoch 1/5
750/750 [==============================] - 4s 5ms/step - loss: 0.3674 -
accuracy: 0.8966 - val_loss: 0.1764 - val_accuracy: 0.9512
Epoch 2/5
750/750 [==============================] - 3s 4ms/step - loss: 0.1786 -
accuracy: 0.9480 - val_loss: 0.1326 - val_accuracy: 0.9618
Epoch 3/5
750/750 [==============================] - 3s 5ms/step - loss: 0.1332 -
accuracy: 0.9611 - val_loss: 0.1116 - val_accuracy: 0.9684
Epoch 4/5
750/750 [==============================] - 3s 4ms/step - loss: 0.1097 -
accuracy: 0.9673 - val_loss: 0.0975 - val_accuracy: 0.9699
Epoch 5/5
750/750 [==============================] - 3s 4ms/step - loss: 0.0922 -
accuracy: 0.9724 - val_loss: 0.0905 - val_accuracy: 0.9739
313/313 - 0s - loss: 0.0833 - accuracy: 0.9750 - 156ms/epoch - 499us/step
Test loss: 0.08331608027219772
Test accuracy: 0.9750000238418579
```

0.3 2. Convolutional Neural Network (CNN)

kernel = filters

pooling= reduir la taille de l'image

0.3.1 Load a dataset

[25]:
```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))
```

0.3.2 Build a model

[26]:
```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(28, (3, 3), activation='relu',␣
↪input_shape=(28, 28,1)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(56, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(56, (3, 3), activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(56, activation='relu'))
model.add(tf.keras.layers.Dense(10))
```

convolution

taille du kernel

28 kernels

pour 2 D, le 1 est exigé

tranformer en une dimension

10 probability as outputs

[27]:
```
model.summary()
```

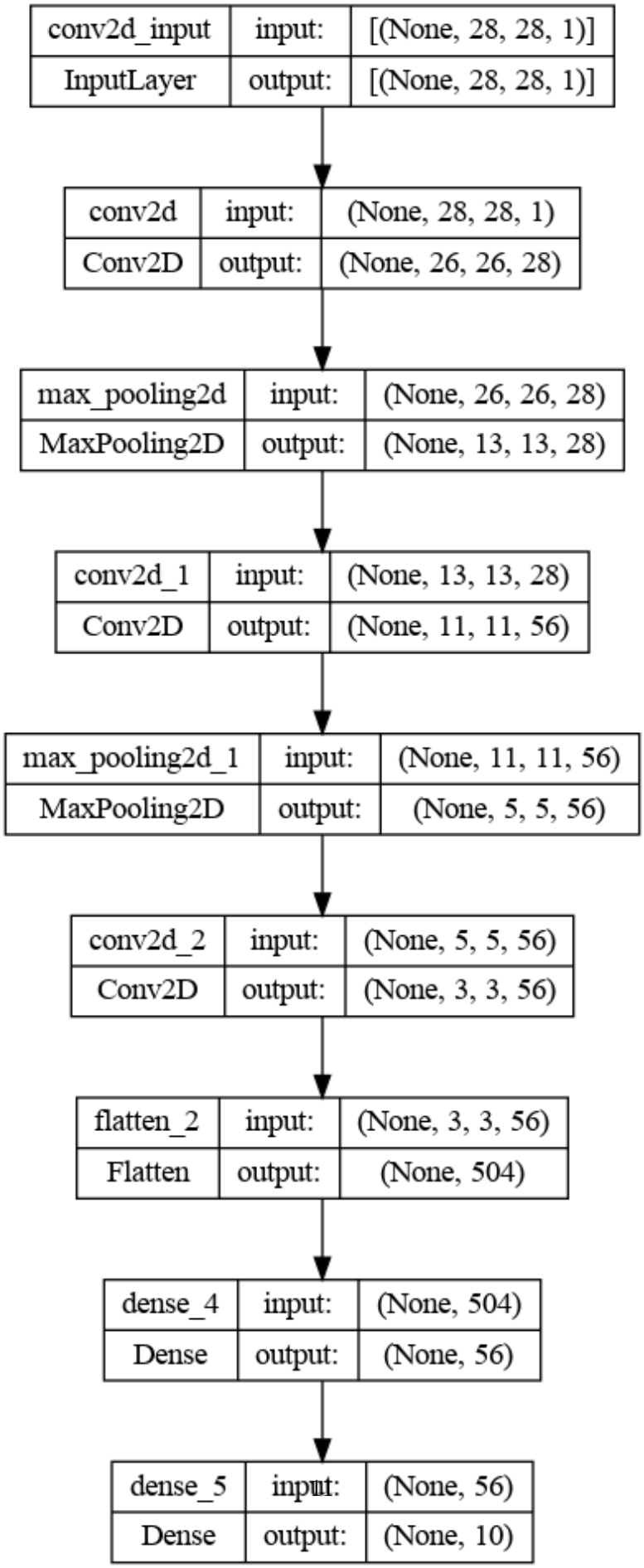
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 28)        280

 max_pooling2d (MaxPooling2D  (None, 13, 13, 28)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 56)        14168

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 56)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 56)          28280

 flatten_2 (Flatten)         (None, 504)               0

 dense_4 (Dense)             (None, 56)                28280

 dense_5 (Dense)             (None, 10)                570

=================================================================
Total params: 71,578
Trainable params: 71,578
Non-trainable params: 0
_________________________________________________________________
```

no parameters in MaxPooling

de réseaux de neuronnes

[28]:
```
tf.keras.utils.plot_model(model, show_shapes=True)
```

[28]:


conv2d_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 26, 26, 28)
max_pooling2d | input: | (None, 26, 26, 28)
MaxPooling2D | output: | (None, 13, 13, 28)
conv2d_1 | input: | (None, 13, 13, 28)
Conv2D | output: | (None, 11, 11, 56)
max_pooling2d_1 | input: | (None, 11, 11, 56)
MaxPooling2D | output: | (None, 5, 5, 56)
conv2d_2 | input: | (None, 5, 5, 56)
Conv2D | output: | (None, 3, 3, 56)
flatten_2 | input: | (None, 3, 3, 56)
Flatten | output: | (None, 504)
dense_4 | input: | (None, 504)
Dense | output: | (None, 56)
dense_5 | input: | (None, 56)
Dense | output: | (None, 10)

0.3.3 Compile the model

[29]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

0.3.4 Train and evaluate your model

[30]:
```
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1535 -
accuracy: 0.9534
Epoch 2/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0486 -
accuracy: 0.9850
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0354 -
accuracy: 0.9888
Epoch 4/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0280 -
accuracy: 0.9913
Epoch 5/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0218 -
accuracy: 0.9929
```

[30]:
```
<keras.callbacks.History at 0x14a9a711ed40>
```

[227]:
```
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 2s - loss: 0.0338 - accuracy: 0.9890
```

[227]:
```
[0.033760134130716324, 0.9890000224113464]
```

[226]:
```
printErrValues(x_test,y_test, model)
```

```
nerr/n= 1574  /  10000
```

0.4 3. Choose devices

[40]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
gpus = tf.config.list_physical_devices('GPU')
gpus
```

```
Num GPUs Available:  0
```

[40]: []

[41]:
```
print("Num CPUs Available: ", len(tf.config.list_physical_devices('CPU')))
cpus = tf.config.list_physical_devices('CPU')
cpus
```

```
Num CPUs Available:  1
```

[41]: [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]

[42]:
```
with tf.device('/CPU:0'):
    # Create some tensors
    a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
    b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])
    c = tf.matmul(a, b)
    print(c)
```

```
tf.Tensor(
[[22. 28.]
 [49. 64.]], shape=(2, 2), dtype=float32)
```

[ ]: