This example is similar to, although not identical to, examples given in Chapter 2 of the book Deep Learning with Python, Second Edition.

Non CNN fully connected network

This first example looks at generating a simple fully connected network. Its function is both to investigate the network topology and it abailities but also to (re-)familiarise you with how to construct such a network in KERAS.

This example will use the functional API components of KERAS which is more flexible that the Sequaential API employed in COMP8270 although hopefully will not prove more challenging to learn.

The network will again use a character recognition task but this time using greyscaled numerals as found in the standard MNIST dataset. This first cell loads this dataset for us to use.

In [1]:
```
import tensorflow as tf

tf.__version__
```

```
2025-02-10 12:41:01.971313: E tensorflow/compiler/xla/stream_execut
or/cuda/cuda_dnn.cc:9342] Unable to register cuDNN factory: Attempt
ing to register factory for plugin cuDNN when one has already been
registered
2025-02-10 12:41:01.971428: E tensorflow/compiler/xla/stream_execut
or/cuda/cuda_fft.cc:609] Unable to register cuFFT factory: Attempti
ng to register factory for plugin cuFFT when one has already been r
egistered
2025-02-10 12:41:01.971474: E tensorflow/compiler/xla/stream_execut
or/cuda/cuda_blas.cc:1518] Unable to register cuBLAS factory: Attem
pting to register factory for plugin cuBLAS when one has already be
en registered
2025-02-10 12:41:03.598182: W tensorflow/compiler/tf2tensorrt/util
s/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
```

Out[1]:
```
'2.14.0'
```

In [2]:
```
from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.lo
```

Note the images consist of 28 x 28 pixels each. There are 60000 trsining images.

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

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

Defining the network archiecture

This is the first section you need to write yourselves.

The workshop script takes you through what you need to do. Note the **Inputs** are defined for you however. Your function is to fill in the missing sections to define the layers you will need to employ.

Note the section concludes by printing out of summary of the model that has been defined.

In [4]:

```
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(28 * 28)) # 28 x 28 grey scaled images

first = layers.Dense(784, activation="sigmoid")(inputs)
second = layers.Dense(784, activation="sigmoid")(first)
outputs = layers.Dense(10, activation="softmax")(second)  # 10 outp

model = keras.Model(inputs=inputs, outputs=outputs)

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0

 dense (Dense)               (None, 784)               615440

 dense_1 (Dense)             (None, 784)               615440

 dense_2 (Dense)             (None, 10)                7850

=================================================================
Total params: 1238730 (4.73 MB)
Trainable params: 1238730 (4.73 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling the model

The next stage is to compile the modeul using an optimiser and an error calculation. Follow the script to deduce what to put here.

In [5]:
```python
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

Preparing the image data

This cell prepares the image data for presentation to the network. It is provided for you although you should examine it to ensure you understand its function.

It essentially reformats the input image to be a flat vector of pixel values with values ranging from 0 to 255.

In [6]:
```python
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255
```

Training the model

To define this sections, the number of epochs need to be selected together with the batch size, the training algorithm itself has already been selected by the **compile** method but the call to the **fit** method actually performs the training.

In [7]:
```python
print("Train images shape:", train_images.shape)  # (60000, 784)
print("Test images shape:", test_images.shape)  # (10000, 784)
```

```
Train images shape: (60000, 784)
Test images shape: (10000, 784)
```

In [8]:
```python
model.fit(train_images, train_labels, epochs=5, batch_size=32, vali
```

```
Epoch 1/5
1875/1875 [==============================] - 31s 16ms/step - loss: 
0.3292 - accuracy: 0.9013 - val_loss: 0.1823 - val_accuracy: 0.9426
Epoch 2/5
1875/1875 [==============================] - 28s 15ms/step - loss: 
0.1334 - accuracy: 0.9598 - val_loss: 0.1016 - val_accuracy: 0.9674
Epoch 3/5
1875/1875 [==============================] - 28s 15ms/step - loss: 
0.0846 - accuracy: 0.9732 - val_loss: 0.0848 - val_accuracy: 0.9749
Epoch 4/5
1875/1875 [==============================] - 28s 15ms/step - loss: 
0.0583 - accuracy: 0.9811 - val_loss: 0.0718 - val_accuracy: 0.9772
Epoch 5/5
1875/1875 [==============================] - 27s 14ms/step - loss: 
0.0420 - accuracy: 0.9865 - val_loss: 0.0639 - val_accuracy: 0.9802
```

Out[8]:
```
<keras.src.callbacks.History at 0x7fa4180fd330>
```

testing the network

The following prints out the probability outputs that arise in the final layer. Change the range of test images to investigate different images

Each element in the **predictions** array represents the result for a given test pattern

In [9]:
```
test_digits = test_images[0:10]
predictions = model.predict(test_digits)
predictions[0]
```

```
1/1 [==============================] - 0s 123ms/step
```

Out[9]:
```
array([9.9463897e-08, 7.8158200e-07, 2.4941264e-06, 2.6641439e-05,
       8.6344470e-10, 6.7925754e-07, 2.9656325e-12, 9.9996108e-01,
       3.9186173e-08, 8.2140286e-06], dtype=float32)
```

In [10]:
```
predictions[0].argmax()
```

Out[10]:
```
7
```

The **evaluate** method provides metric values for the entire test set

In [11]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f"test_acc: {test_acc}")
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.06
39 - accuracy: 0.9802
test_acc: 0.9801999926567078
```

In [ ]:
```
```