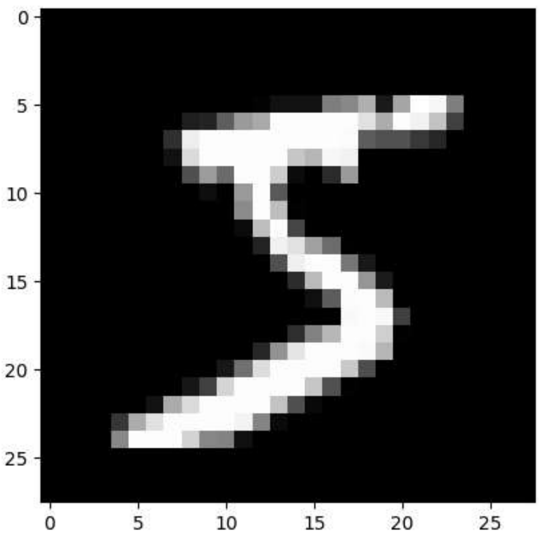
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
from sklearn import metrics
```

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
plt.imshow(x_train[0], cmap='gray') # imshow() function which simply displays an image.
plt.show()
```




```
print(x_train[0])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
```

```
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```

```
print("X_train shape", x_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", x_test.shape)
print("y_test shape", y_test.shape)
```

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
```

```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32') # use 32-bit precision when training a neural network, so at one point

x_test = x_test.astype('float32')
x_train /= 255 # Each image has Intensity from 0 to 255
x_test /= 255
```

```
num_classes = 10
y_train = np.eye(num_classes)[y_train]
y_test = np.eye(num_classes)[y_test]
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2)) # DROP OUT RATIO 20%
model.add(Dense(512, activation='relu')) #returns a sequence of another vectors of dimension 512
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', # for a multi-class classification problem
optimizer=RMSprop(),
metrics=['accuracy'])
```

```
# Train the model
batch_size = 128 # batch_size argument is passed to the layer to define a batch size for the inputs.
epochs = 20
history = model.fit(x_train, y_train,
batch_size=batch_size,
epochs=epochs,
verbose=1, # verbose=1 will show you an animated progress bar eg. [==========]
validation_data=(x_test, y_test))

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

```
Epoch 1/20
469/469 [==============================] - 19s 37ms/step - loss: 0.2498 - accuracy: 0.9238 - val_loss: 0.1036 - val_accuracy: 0.9665
```

```
Epoch 2/20
469/469 [==============================] - 10s 22ms/step - loss: 0.1015 - accuracy: 0.9691 - val_loss: 0.0706 - val_accuracy: 0.9781
Epoch 3/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0743 - accuracy: 0.9769 - val_loss: 0.0777 - val_accuracy: 0.9760
Epoch 4/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0580 - accuracy: 0.9824 - val_loss: 0.0745 - val_accuracy: 0.9776
Epoch 5/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0466 - accuracy: 0.9852 - val_loss: 0.0582 - val_accuracy: 0.9830
Epoch 6/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0394 - accuracy: 0.9874 - val_loss: 0.0670 - val_accuracy: 0.9835
Epoch 7/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0330 - accuracy: 0.9893 - val_loss: 0.0642 - val_accuracy: 0.9837
Epoch 8/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0289 - accuracy: 0.9907 - val_loss: 0.0650 - val_accuracy: 0.9841
Epoch 9/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0259 - accuracy: 0.9917 - val_loss: 0.0617 - val_accuracy: 0.9837
Epoch 10/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0234 - accuracy: 0.9924 - val_loss: 0.0591 - val_accuracy: 0.9848
Epoch 11/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0199 - accuracy: 0.9941 - val_loss: 0.0715 - val_accuracy: 0.9830
Epoch 12/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0180 - accuracy: 0.9940 - val_loss: 0.0754 - val_accuracy: 0.9837
Epoch 13/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0169 - accuracy: 0.9943 - val_loss: 0.0687 - val_accuracy: 0.9853
Epoch 14/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0149 - accuracy: 0.9951 - val_loss: 0.0688 - val_accuracy: 0.9859
Epoch 15/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0126 - accuracy: 0.9960 - val_loss: 0.0820 - val_accuracy: 0.9831
Epoch 16/20
469/469 [==============================] - 10s 22ms/step - loss: 0.0138 - accuracy: 0.9958 - val_loss: 0.0707 - val_accuracy: 0.9854
Epoch 17/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0127 - accuracy: 0.9959 - val_loss: 0.0786 - val_accuracy: 0.9850
Epoch 18/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0114 - accuracy: 0.9962 - val_loss: 0.0837 - val_accuracy: 0.9839
Epoch 19/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0109 - accuracy: 0.9964 - val_loss: 0.0767 - val_accuracy: 0.9859
Epoch 20/20
469/469 [==============================] - 10s 20ms/step - loss: 0.0085 - accuracy: 0.9973 - val_loss: 0.0877 - val_accuracy: 0.9840
Test loss: 0.0877445712685585
Test accuracy: 0.984000027179718
```