```
# Train a fully connected deep neural network for recognizing English
digits, i.e., 0, 1, 2, ..., 9.
# Sangeeta Biswas
# 31.12.2021

from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Input, Dense, Flatten
from tensorflow.keras import Model
from tensorflow.keras.optimizers import RMSprop
import numpy as np
from tensorflow.keras.callbacks import EarlyStopping, History
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical

DIR = '/home/bibrity/DeepLearning/'

def main():
        # Prepare data sets
        trainX, trainY, testX, testY = prepare_data()

        # Build a model
        model = build_model()

        # Train the model
        callbackList = [EarlyStopping(monitor = 'val_loss', patience =
10), History()]
        history = model.fit(trainX, trainY, epochs = 300, batch_size =
16, callbacks = callbackList, validation_split = 0.2)
        plot_loss(history)

        # Check what the model predicts.
        predictY = model.predict(testX)
        for i in range(10):
                y = np.argmax(testY[i])
                pY = np.argmax(predictY[i])
                print('Original Y: {}, Predicted Y: {}'.format(y, pY))

        # Estimate the performance of the NN.
        model.compile(metrics = 'accuracy')
        model.evaluate(testX, testY)

def build_model():
        inputs = Input((28, 28))
        x = Flatten()(inputs)
        x = Dense(32, activation = 'sigmoid')(x)
        x = Dense(16, activation = 'sigmoid')(x)
        outputs = Dense(10)(x)

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

        model.compile(loss = 'mse', optimizer = RMSprop(learning_rate
= 0.001))

        return model
```

```
def prepare_data():
        # Load data
        (trainX, trainY), (testX, testY) = mnist.load_data()
        #plot_digits(trainX[:9], trainY[:9])
        print(trainX.shape, trainY.shape, testX.shape, testY.shape)

        # Convert numeric digit labels into one-hot vectors.
        # 0: 1 0 0 0 0 0 0 0 0 0
        # 1: 0 1 0 0 0 0 0 0 0 0
        # 2: 0 0 1 0 0 0 0 0 0 0
        print('Labels: {}, DataType: {}'.format(trainY[:10],
trainY[:10].dtype))
        classN = 10
        trainY = to_categorical(trainY, classN)
        testY = to_categorical(testY, classN)
        print('Labels: {}, DataType: {}'.format(trainY[:10],
trainY[:10].dtype))

        # To convert pixel values from 0-255 into 0-1.
        print('DataType: {}, Max: {}, Min: {}'.format(trainX.dtype,
trainX.max(), trainX.min()))
        trainX = trainX.astype(np.float32)
        testX = testX.astype(np.float32)
        trainX /= 255
        testX /= 255
        print('DataType: {}, Max: {}, Min: {}'.format(trainX.dtype,
trainX.max(), trainX.min()))

        return trainX, trainY, testX, testY

def plot_digits(x, y):
        n = len(y)
        plt.figure(figsize = (20,20))
        for i in range(n):
                plt.subplot(3, 3, i+1)
                plt.imshow(x[i], cmap = 'gray')
                plt.title(y[i])
        plt.show()
        plt.close()

def plot_loss(history):
        loss = history.history['loss']
        valLoss = history.history['val_loss']
        epochs = range(1, len(loss) + 1)

        plt.figure(figsize = (20, 20))
        plt.rcParams['font.size'] = '20'
        plt.plot(epochs, loss, 'bo-', label = 'Training loss')
        plt.plot(epochs, valLoss, 'k*-', label = 'Validation loss')
        plt.title('Training and validation loss')
        plt.legend()

        figPath = DIR + 'Digit_Recognizer_TrainvsVal_Loss.png'
        plt.savefig(figPath)
        plt.close()

if __name__ == '__main__':
```

```
main()
```