# Handwritten Digit Recognition

In this mini-project we would train a neural network for the task of hand-written digit recognition, using the standard mnist dataset

## Importing Packages

In [ ]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import linear, relu, sigmoid
from sklearn.model_selection import train_test_split
```

## Dataset

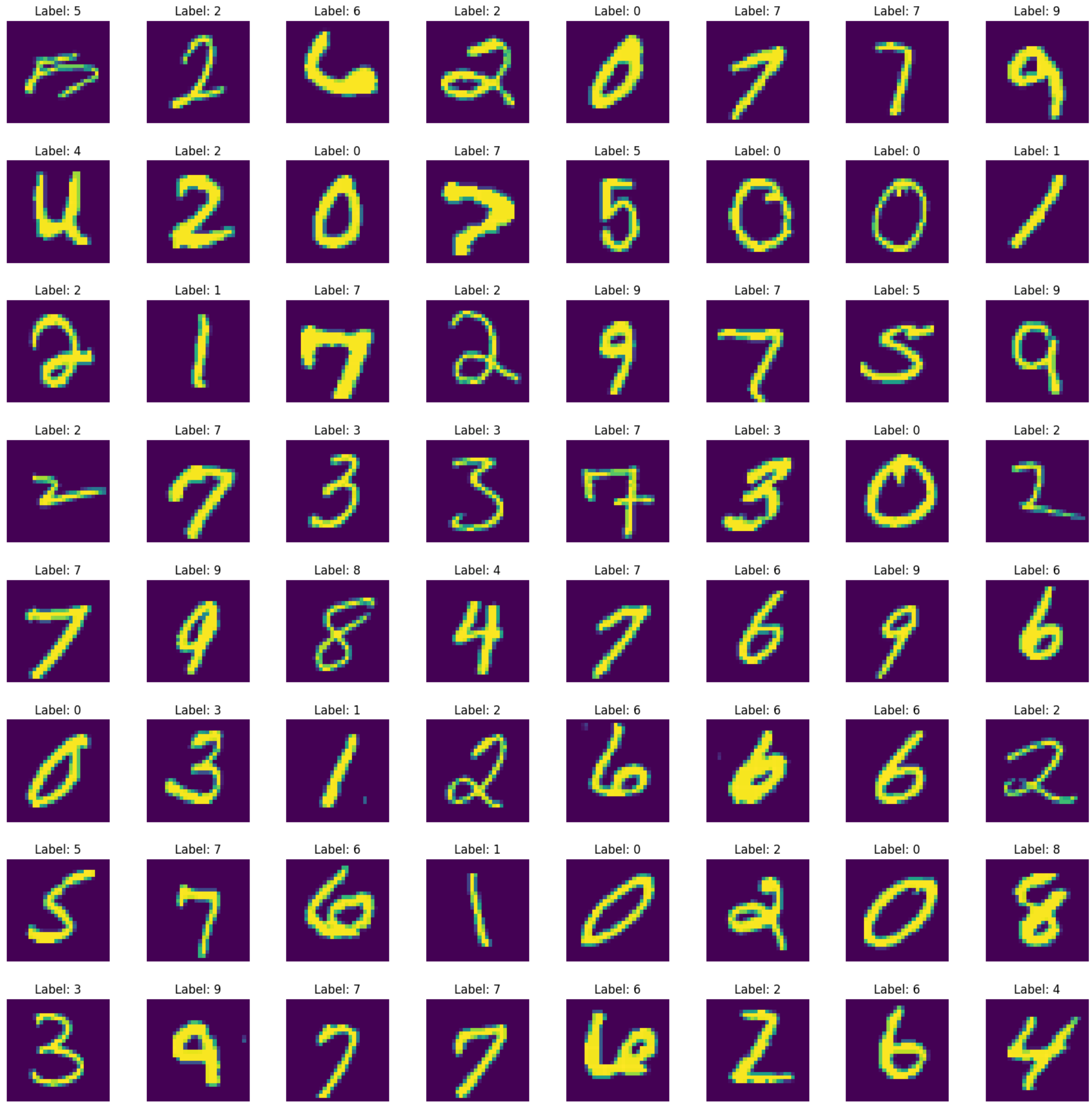
In [ ]:
```
data = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = data.load_data()
print("Number of training examples:", x_train.shape[0])
print("Number of test examples:", x_test.shape[0])
```

```
Number of training examples: 60000
Number of test examples: 10000
```

Let us visualise some of the training images along with the given labels

In [ ]:
```
np.random.seed(100)
fig, axes = plt.subplots(8,8, figsize=(16,16))
fig.tight_layout(pad=0.15)
m = x_train.shape[0]

for i,ax in enumerate(axes.flat):
    index = np.random.randint(m)
    ax.imshow(x_train[index])
    ax.set_title(f"Label: {y_train[index]}")
    ax.set_axis_off()
```




# Building the neural network

We would have three layers in the neural network:

1. Layer 1 has 128 units with ReLU activation
2. Layer 2 has 64 units with ReLU activation
3. Layer 3 has 10 units (for the 10 digits) with softmax activation

Each training example is an 28 x 28 image, we would have to reshape it to (784,) to feed it to the layer 1 of the neural networks.

In [ ]:

```
tf.random.set_seed(100)
model = Sequential(
    [
        tf.keras.layers.InputLayer((784,)),
        tf.keras.layers.Dense(128, activation="relu", name="layer1"),
        tf.keras.layers.Dense(64, activation="relu", name="layer2"),
        tf.keras.layers.Dense(10, activation="softmax", name="layer3")
    ], name = "recognizer"
)
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_log
```

Let us just cross-verify if the model made is what we desired for

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

```
Model: "recognizer"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 layer1 (Dense)              (None, 128)               100480    
                                                                 
 layer2 (Dense)              (None, 64)                8256      
                                                                 
 layer3 (Dense)              (None, 10)                650       
                                                                 
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

# Training the Model

- Now that the model is built , we specify the loss function and the optimiser to be used.
- We use the Sparse Categorical Crossentropy loss function which is the ideal loss function for the multiclass classification problems
- And finally we fit the model to the given dataset.

In [ ]:
```
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
)

model.fit(
    x_train.reshape(m,784),y_train,
    epochs=50
)
```

```
Epoch 1/50
1875/1875 [==============================] - 9s 4ms/step - loss: 1.5079
Epoch 2/50
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2903
Epoch 3/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2038
Epoch 4/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1643
Epoch 5/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1398
Epoch 6/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1249
Epoch 7/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1126
Epoch 8/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1018
Epoch 9/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0866
Epoch 10/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0850
Epoch 11/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0780
Epoch 12/50
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0728
Epoch 13/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0709
Epoch 14/50
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0683
Epoch 15/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0621
Epoch 16/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0604
Epoch 17/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0617
Epoch 18/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0529
Epoch 19/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0556
Epoch 20/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0582
Epoch 21/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0574
Epoch 22/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0498
Epoch 23/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0531
Epoch 24/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0507
Epoch 25/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0506
Epoch 26/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0556
Epoch 27/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0462
Epoch 28/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0487
Epoch 29/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0455
Epoch 30/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0548
```

```
Epoch 31/50
1875/1875 [==============================] - 16s 9ms/step - loss: 0.0430
Epoch 32/50
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0518
Epoch 33/50
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0435
Epoch 34/50
1875/1875 [==============================] - 14s 7ms/step - loss: 0.0486
Epoch 35/50
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0525
Epoch 36/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0463
Epoch 37/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0432
Epoch 38/50
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0449
Epoch 39/50
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0498
Epoch 40/50
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0437
Epoch 41/50
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0487
Epoch 42/50
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0488
Epoch 43/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0533
Epoch 44/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0432
Epoch 45/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0443
Epoch 46/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0495
Epoch 47/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0434
Epoch 48/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0393
Epoch 49/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0563
Epoch 50/50
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0542
```

Out[ ]:

```
<keras.callbacks.History at 0x7f80a13c8a90>
```

## Testing the Model

Let us now get the predictions of the model for the testing dataset and subsequently check how accuarte did the model turn out :)

In [ ]:

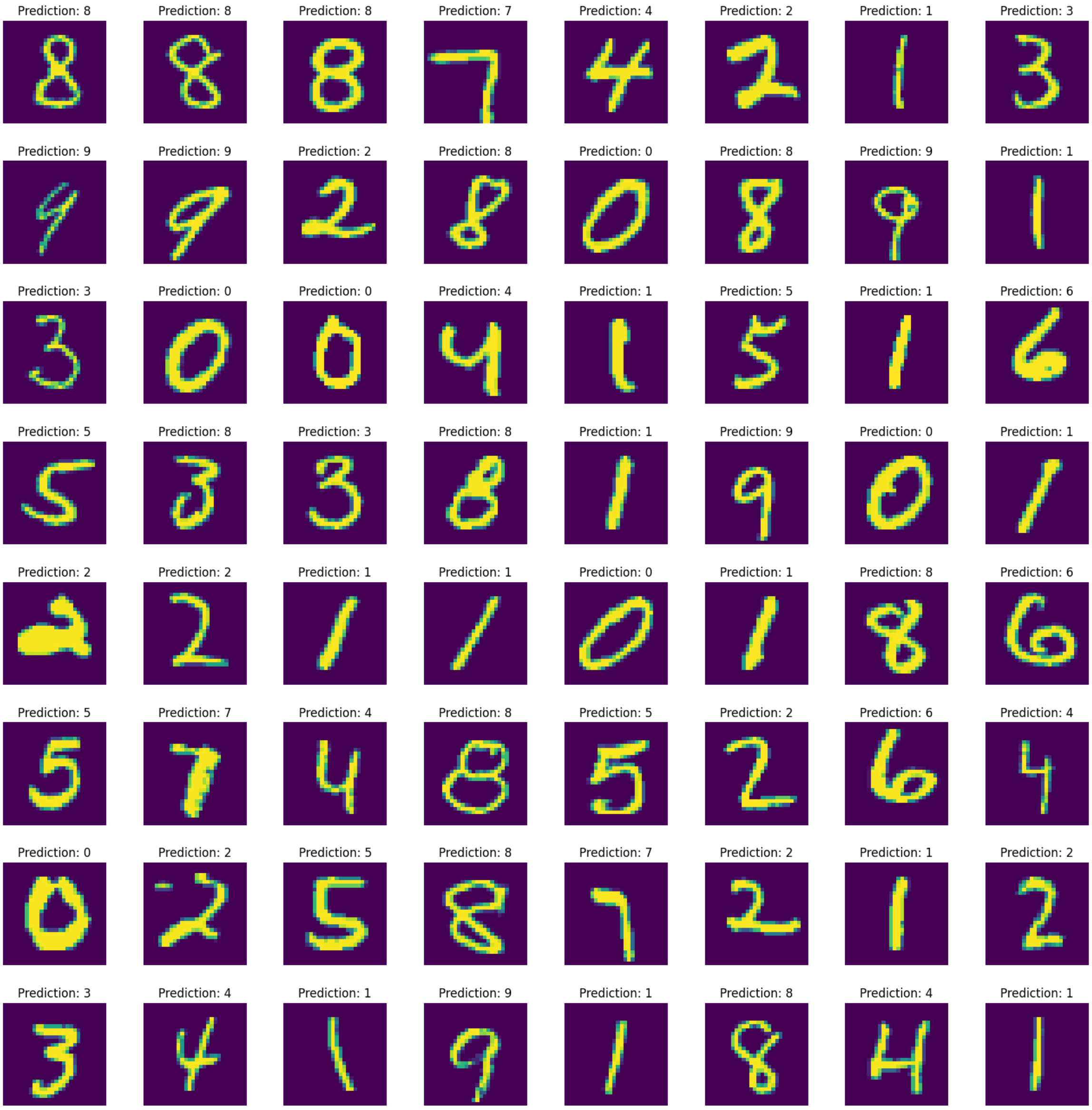
```python
n = x_test.shape[0]
predictions_vector = model.predict(x_test.reshape(n,784))
predictions = np.argmax(predictions_vector, axis = 1)
```

```
313/313 [==============================] - 1s 3ms/step
```

In [ ]:

```
num_rows = 8
num_columns = 8

fig, ax = plt.subplots(num_rows, num_columns, figsize=(16,16))
fig.tight_layout(pad=0.15)
for i in range(num_rows):
    for j in range(num_columns):
        index = np.random.randint(n)
        ax[i,j].imshow(x_test[index])
        ax[i,j].set_title(f'Prediction: {predictions[index]}')
        ax[i,j].set_axis_off()
```




Now that we are done with building the model and also visualised some of the images and their outputs , let us check how accurate is the model , by using the testing dataset

In [ ]:

```
correct_vec = (predictions==y_test)
correct = correct_vec.sum()
accuracy = correct*100/n
print(f"The accuracy of the model is {accuracy}%")
```

```
The accuracy of the model is 97.26%
```