Handwritten Digit Recognition

Importing Necessary Modules

In [ ]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
import math
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
import matplotlib.pyplot as plt

import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
tf.autograph.set_verbosity(0)
```

Load the Training Data

In [ ]:
```
train = pd.read_csv("/Users/mridulsharma/Desktop/ML_project/train.csv")
```

In [ ]:
```
print(f"Dataset Shape :-")
print (train.shape)
```

```
Dataset Shape :-
(42000, 785)
```

In [ ]:
```
#seperate labels and dataset in order to train the model
X = train.iloc[:, 1:785]
y = train.iloc[:, 0]
```

In [ ]:
```
# split the dataset into training, validation and testing dataset
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2,
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size
```

In [ ]:
```
print('X_train:', X_train.shape)
print('y_train:', y_train.shape)
print('X_validation:', X_val.shape)
print('y_validation:', y_val.shape)
print('X_test:', X_test.shape)
print('y_test:', y_test.shape)
```

```
X_train: (33600, 784)
y_train: (33600,)
X_validation: (4200, 784)
y_validation: (4200,)
X_test: (4200, 784)
y_test: (4200,)
```

In [ ]:
```
# convert all the data into image shapes i.e. (28 x 28) pixels
# so that it can easily be converted to images using plt.imshow()
x_train_re = X_train.to_numpy().reshape(33600, 28, 28)
y_train_re = y_train.values
x_validation_re = X_val.to_numpy().reshape(4200, 28, 28)
y_validation_re = y_val.values
X_test_re = X_test.to_numpy().reshape(4200, 28, 28)
y_test_re = y_test.values
```

In [ ]:
```
# Save image parameters to the constants that we will use later for data
(_, IMAGE_WIDTH, IMAGE_HEIGHT) = x_train_re.shape
IMAGE_CHANNELS = 1

print('IMAGE_WIDTH:', IMAGE_WIDTH)
print('IMAGE_HEIGHT:', IMAGE_HEIGHT)
print('IMAGE_CHANNELS:', IMAGE_CHANNELS)
```

```
IMAGE_WIDTH: 28
IMAGE_HEIGHT: 28
IMAGE_CHANNELS: 1
```

Explore The Data

In [ ]:
```
pd.DataFrame(x_train_re[0])
```

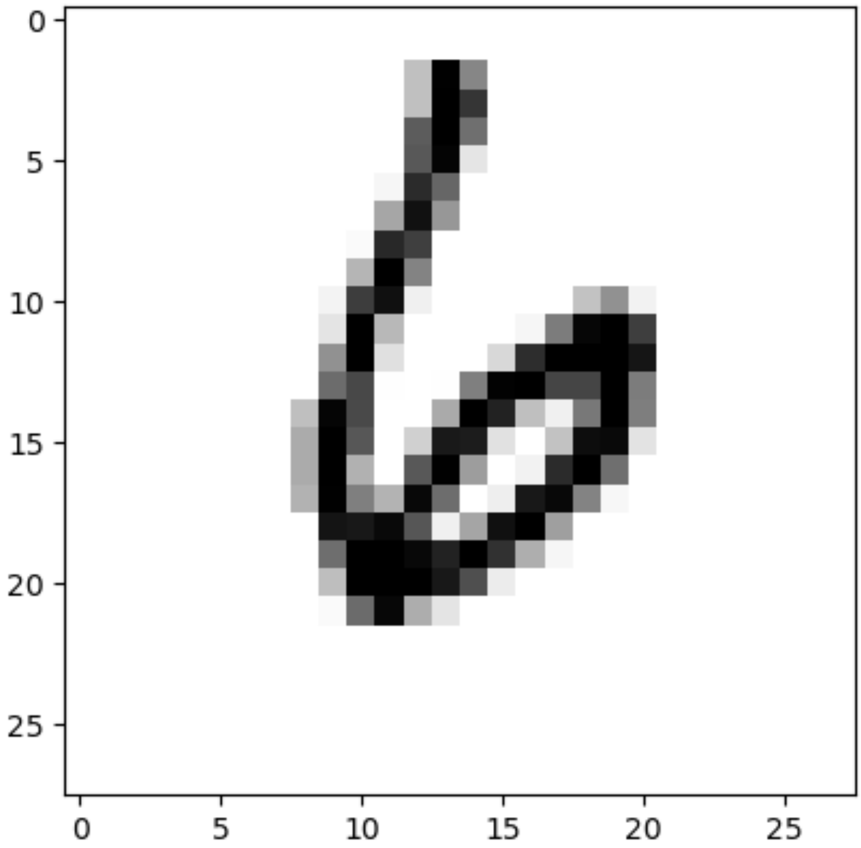
Out[ ]:

	0	1	2	3	4	5	6	7	8	9	...	18	19	20	21	22	23	24	25	26
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	12	...	61	110	13	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	27	...	249	254	193	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	110	...	254	254	234	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	147	...	186	254	131	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	64	250	...	135	254	129	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	84	254	...	242	245	28	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	84	254	...	254	144	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	77	253	...	123	8	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	235	...	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	145	...	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	65	...	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	5	...	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0

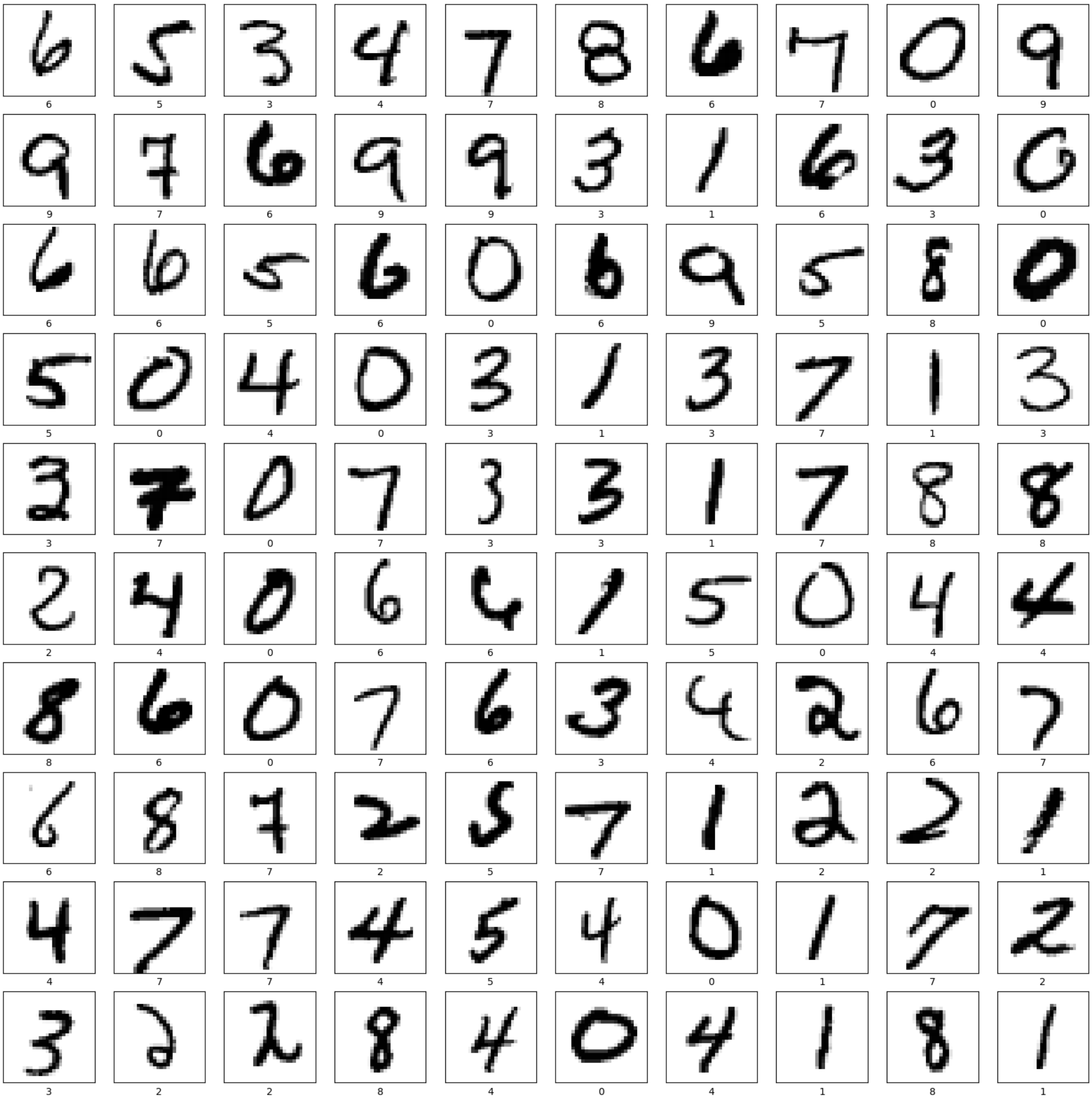
28 rows × 28 columns

Try visualising some of the data using pictures made using the function plt.imshow()

In [ ]:
```
plt.imshow(x_train_re[0], cmap=plt.cm.binary)
plt.show()
```



In [ ]:
```
numbers_to_display = 100
num_cells = math.ceil(math.sqrt(numbers_to_display))
plt.figure(figsize=(20,20))
for i in range(numbers_to_display):
    plt.subplot(num_cells, num_cells, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_train_re[i], cmap=plt.cm.binary)
    plt.xlabel(y_train_re[i])
plt.show()
```



Build the Model

In [ ]:
```
tf.random.set_seed(1234) # for consistent results
model = Sequential(
[
    tf.keras.Input(shape=(784,)),
    Dense(25, activation='relu', name = "L1"),
    Dense(15, activation='relu', name = "L2"),
    Dense(10, activation='linear', name = "L3"),
],
name = "my_model"
)
```

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

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 L1 (Dense)                  (None, 25)                19625

 L2 (Dense)                  (None, 15)                390

 L3 (Dense)                  (None, 10)                160

=================================================================
Total params: 20,175
Trainable params: 20,175
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 L1 (Dense)                  (None, 25)                19625

 L2 (Dense)                  (None, 15)                390

 L3 (Dense)                  (None, 10)                160

=================================================================
Total params: 20,175
Trainable params: 20,175
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
[layer1, layer2, layer3] = model.layers
```

In [ ]:

```
#### Examine Weights shapes
W1,b1 = layer1.get_weights()
W2,b2 = layer2.get_weights()
W3,b3 = layer3.get_weights()
print(f"W1 shape = {W1.shape}, b1 shape = {b1.shape}")
print(f"W2 shape = {W2.shape}, b2 shape = {b2.shape}")
print(f"W3 shape = {W3.shape}, b3 shape = {b3.shape}")
```

```
W1 shape = (784, 25), b1 shape = (25,)
W2 shape = (25, 15), b2 shape = (15,)
W3 shape = (15, 10), b3 shape = (10,)
```

In [ ]:

```
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy']
)
training_history = model.fit(
    X_train, y_train,
    epochs=40,
    validation_data=(X_val, y_val)
```

```
)
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.Adam`.
WARNING:absl:There is a known slowdown when using v2.11+ Keras optimizers on M1/M2 Macs. Falling back to the legacy Keras optimizer, i.e., `tf.keras.optimizers.legacy.Adam`.
Epoch 1/40
1050/1050 [==============================] - 1s 581us/step - loss: 2.5013 - accuracy: 0.3415 - val_loss: 1.3178 - val_accuracy: 0.5345
Epoch 2/40
1050/1050 [==============================] - 1s 506us/step - loss: 1.1294 - accuracy: 0.5988 - val_loss: 0.9859 - val_accuracy: 0.6990
Epoch 3/40
1050/1050 [==============================] - 1s 497us/step - loss: 0.7766 - accuracy: 0.7462 - val_loss: 0.7357 - val_accuracy: 0.7362
Epoch 4/40
1050/1050 [==============================] - 1s 486us/step - loss: 0.6506 - accuracy: 0.7828 - val_loss: 0.6107 - val_accuracy: 0.7945
Epoch 5/40
1050/1050 [==============================] - 1s 490us/step - loss: 0.5684 - accuracy: 0.8028 - val_loss: 0.5714 - val_accuracy: 0.7960
Epoch 6/40
1050/1050 [==============================] - 1s 514us/step - loss: 0.5141 - accuracy: 0.8190 - val_loss: 0.5368 - val_accuracy: 0.8307
Epoch 7/40
1050/1050 [==============================] - 1s 486us/step - loss: 0.4750 - accuracy: 0.8364 - val_loss: 0.4797 - val_accuracy: 0.8298
Epoch 8/40
1050/1050 [==============================] - 1s 489us/step - loss: 0.4387 - accuracy: 0.8547 - val_loss: 0.4556 - val_accuracy: 0.8671
Epoch 9/40
1050/1050 [==============================] - 1s 494us/step - loss: 0.3908 - accuracy: 0.8808 - val_loss: 0.3933 - val_accuracy: 0.8910
Epoch 10/40
1050/1050 [==============================] - 1s 489us/step - loss: 0.3551 - accuracy: 0.8985 - val_loss: 0.3539 - val_accuracy: 0.9029
Epoch 11/40
1050/1050 [==============================] - 1s 502us/step - loss: 0.3218 - accuracy: 0.9119 - val_loss: 0.3485 - val_accuracy: 0.9079
Epoch 12/40
1050/1050 [==============================] - 1s 499us/step - loss: 0.2881 - accuracy: 0.9216 - val_loss: 0.3521 - val_accuracy: 0.9107
Epoch 13/40
1050/1050 [==============================] - 1s 488us/step - loss: 0.2701 - accuracy: 0.9285 - val_loss: 0.3286 - val_accuracy: 0.9160
Epoch 14/40
1050/1050 [==============================] - 1s 491us/step - loss: 0.2506 - accuracy: 0.9335 - val_loss: 0.2958 - val_accuracy: 0.9231
Epoch 15/40
1050/1050 [==============================] - 1s 516us/step - loss: 0.2320 - accuracy: 0.9388 - val_loss: 0.2931 - val_accuracy: 0.9231
Epoch 16/40
```

```
1050/1050 [==============================] - 1s 537us/step - loss: 0.2190
- accuracy: 0.9429 - val_loss: 0.2998 - val_accuracy: 0.9305
Epoch 17/40
1050/1050 [==============================] - 1s 500us/step - loss: 0.2093
- accuracy: 0.9439 - val_loss: 0.2691 - val_accuracy: 0.9324
Epoch 18/40
1050/1050 [==============================] - 1s 514us/step - loss: 0.1960
- accuracy: 0.9464 - val_loss: 0.2979 - val_accuracy: 0.9288
Epoch 19/40
1050/1050 [==============================] - 1s 528us/step - loss: 0.1893
- accuracy: 0.9488 - val_loss: 0.3340 - val_accuracy: 0.9260
Epoch 20/40
1050/1050 [==============================] - 1s 486us/step - loss: 0.1850
- accuracy: 0.9507 - val_loss: 0.2543 - val_accuracy: 0.9398
Epoch 21/40
1050/1050 [==============================] - 1s 498us/step - loss: 0.1731
- accuracy: 0.9525 - val_loss: 0.2772 - val_accuracy: 0.9326
Epoch 22/40
1050/1050 [==============================] - 0s 475us/step - loss: 0.1689
- accuracy: 0.9526 - val_loss: 0.2779 - val_accuracy: 0.9333
Epoch 23/40
1050/1050 [==============================] - 1s 486us/step - loss: 0.1588
- accuracy: 0.9558 - val_loss: 0.2625 - val_accuracy: 0.9402
Epoch 24/40
1050/1050 [==============================] - 1s 511us/step - loss: 0.1525
- accuracy: 0.9568 - val_loss: 0.2697 - val_accuracy: 0.9414
Epoch 25/40
1050/1050 [==============================] - 1s 496us/step - loss: 0.1490
- accuracy: 0.9576 - val_loss: 0.2746 - val_accuracy: 0.9400
Epoch 26/40
1050/1050 [==============================] - 1s 488us/step - loss: 0.1467
- accuracy: 0.9584 - val_loss: 0.2560 - val_accuracy: 0.9395
Epoch 27/40
1050/1050 [==============================] - 1s 492us/step - loss: 0.1427
- accuracy: 0.9595 - val_loss: 0.2666 - val_accuracy: 0.9374
Epoch 28/40
1050/1050 [==============================] - 1s 492us/step - loss: 0.1353
- accuracy: 0.9615 - val_loss: 0.3186 - val_accuracy: 0.9352
Epoch 29/40
1050/1050 [==============================] - 1s 511us/step - loss: 0.1332
- accuracy: 0.9614 - val_loss: 0.2618 - val_accuracy: 0.9395
Epoch 30/40
1050/1050 [==============================] - 1s 487us/step - loss: 0.1289
- accuracy: 0.9627 - val_loss: 0.2910 - val_accuracy: 0.9407
Epoch 31/40
1050/1050 [==============================] - 1s 488us/step - loss: 0.1253
- accuracy: 0.9636 - val_loss: 0.2638 - val_accuracy: 0.9405
Epoch 32/40
1050/1050 [==============================] - 1s 491us/step - loss: 0.1202
- accuracy: 0.9648 - val_loss: 0.2703 - val_accuracy: 0.9440
Epoch 33/40
1050/1050 [==============================] - 1s 487us/step - loss: 0.1209
- accuracy: 0.9653 - val_loss: 0.3011 - val_accuracy: 0.9393
```

```
Epoch 34/40
1050/1050 [==============================] - 1s 505us/step - loss: 0.1202
- accuracy: 0.9657 - val_loss: 0.2634 - val_accuracy: 0.9438
Epoch 35/40
1050/1050 [==============================] - 1s 486us/step - loss: 0.1127
- accuracy: 0.9671 - val_loss: 0.2801 - val_accuracy: 0.9417
Epoch 36/40
1050/1050 [==============================] - 1s 492us/step - loss: 0.1111
- accuracy: 0.9686 - val_loss: 0.3340 - val_accuracy: 0.9324
Epoch 37/40
1050/1050 [==============================] - 1s 492us/step - loss: 0.1100
- accuracy: 0.9685 - val_loss: 0.2710 - val_accuracy: 0.9417
Epoch 38/40
1050/1050 [==============================] - 1s 491us/step - loss: 0.1142
- accuracy: 0.9677 - val_loss: 0.3400 - val_accuracy: 0.9362
Epoch 39/40
1050/1050 [==============================] - 1s 513us/step - loss: 0.1098
- accuracy: 0.9689 - val_loss: 0.2844 - val_accuracy: 0.9421
Epoch 40/40
1050/1050 [==============================] - 1s 487us/step - loss: 0.1033
- accuracy: 0.9712 - val_loss: 0.3029 - val_accuracy: 0.9417
```

Analysing Performance

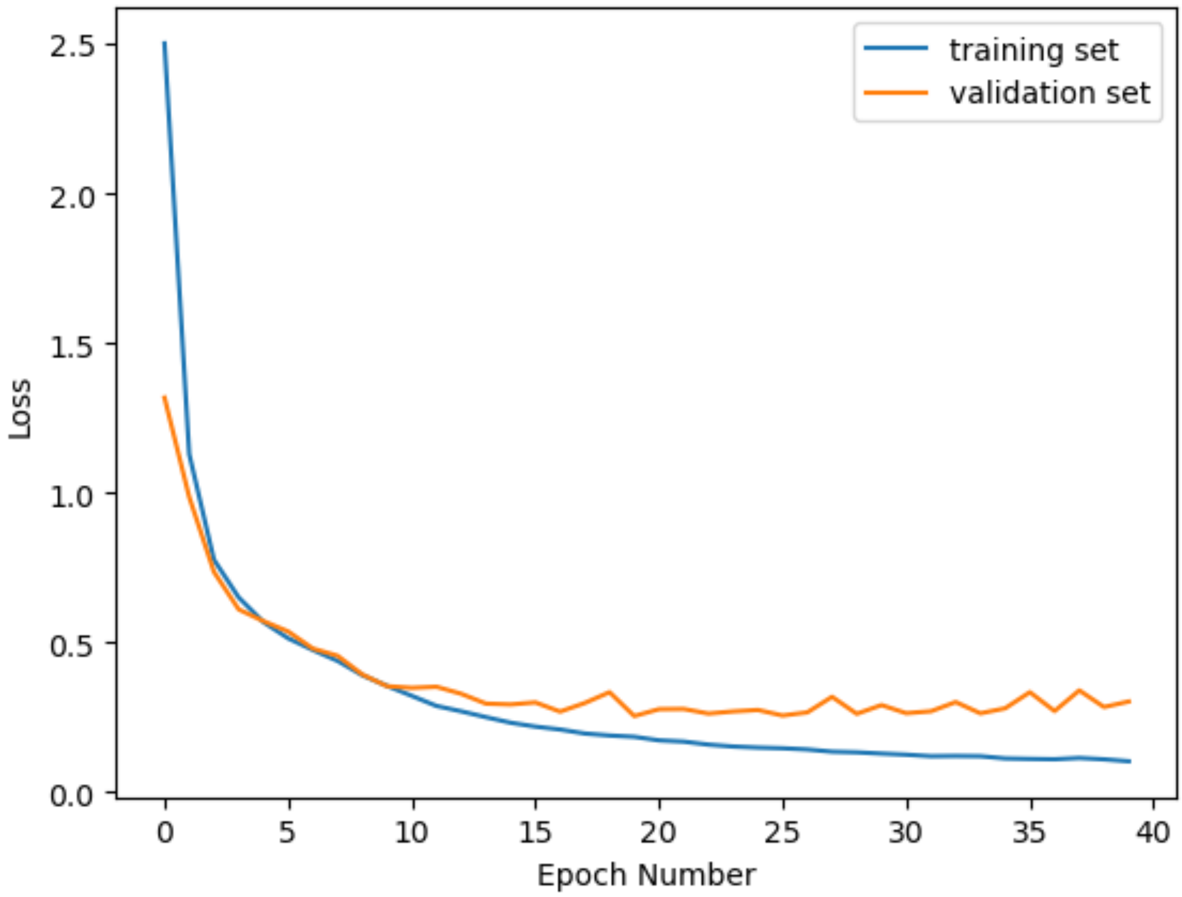
1. Plot Loss function w.r.t number of epochs

In [ ]:

```python
plt.xlabel('Epoch Number')
plt.ylabel('Loss')
plt.plot(training_history.history['loss'], label='training set')
plt.plot(training_history.history['val_loss'], label='validation set')
plt.legend()
```

Out[ ]:

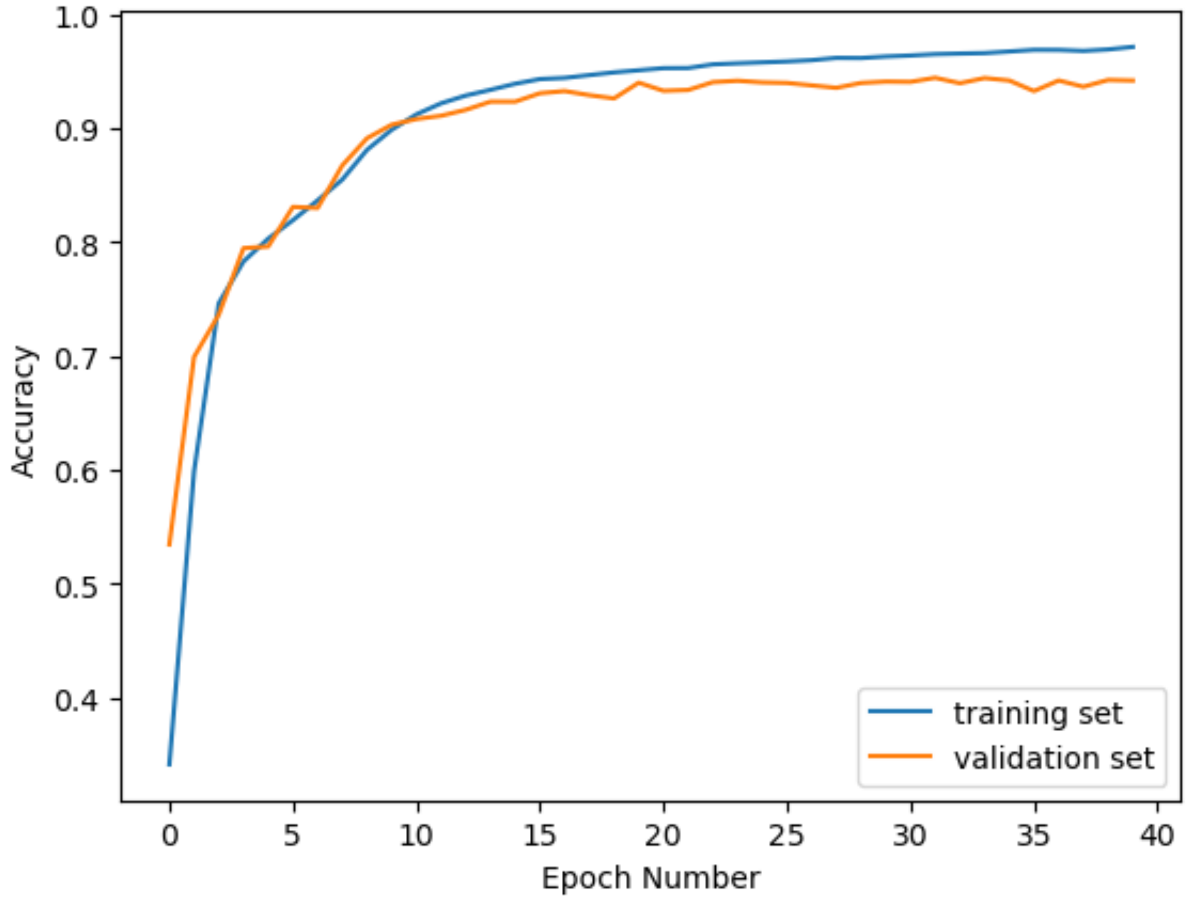
```
<matplotlib.legend.Legend at 0x30d44a150>
```



2. Plot accuracy w.r.t number of epochs

In [ ]:
```
plt.xlabel('Epoch Number')
plt.ylabel('Accuracy')
plt.plot(training_history.history['accuracy'], label='training set')
plt.plot(training_history.history['val_accuracy'], label='validation set'
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x305620790>
```



Testing the Trained Model

In [ ]:
```
predictions = model.predict([X_test])
print('predictions:', predictions.shape)
```

```
132/132 [==============================] - 0s 324us/step
predictions: (4200, 10)
```

In [ ]:
```
# Predictions in form of one-hot vectors (arrays of probabilities).
# The highest probability is the predicted class.
pd.DataFrame(predictions)
```

Out[ ]:

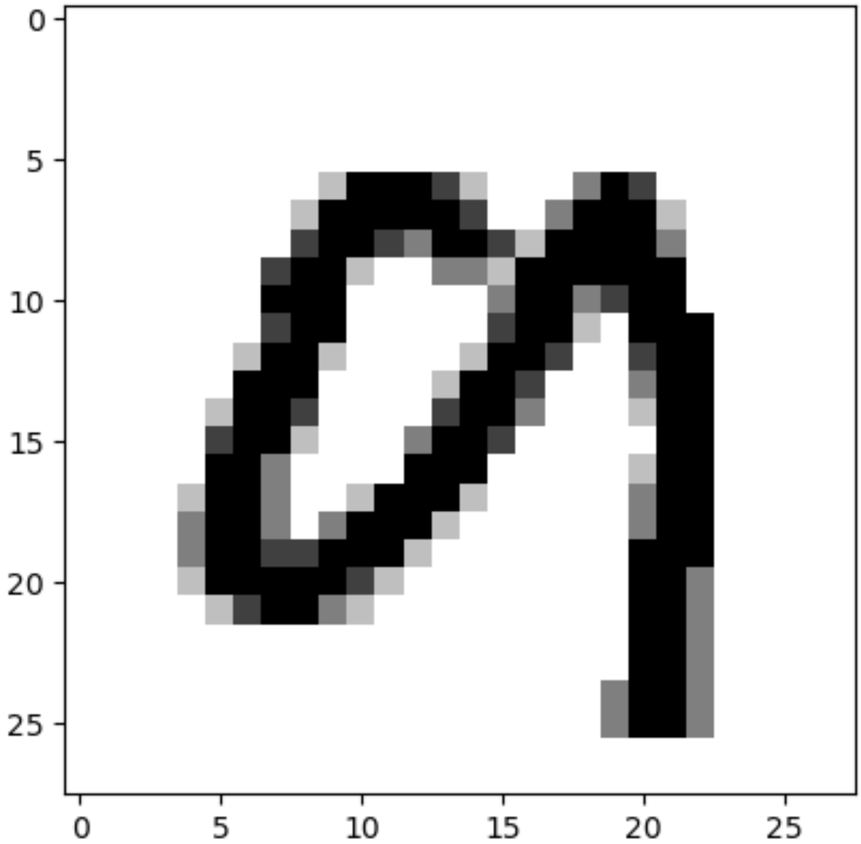
	0	1	2	3	4	5	
0	59.417007	59.607513	59.644722	61.889397	56.934025	60.664894	54.33
1	32.039215	31.337374	31.485971	29.114258	37.966961	31.156721	25.82
2	95.898804	78.203308	63.843170	42.432873	67.131920	67.010452	73.29
3	114.342758	94.099541	76.296234	50.702309	80.552811	79.425209	87.79
4	53.371029	57.970497	56.352951	57.192131	61.070526	53.997154	47.91
...	...	...	...	...	...	...	
4195	2.908936	1.704827	3.460591	3.307315	2.837388	5.725966	3.32
4196	27.676325	37.955547	35.471077	34.723713	47.603481	33.308849	31.43
4197	77.735466	49.176083	70.576866	54.588432	68.246208	113.150208	95.77
4198	97.880356	115.261284	98.793098	86.306343	111.852722	85.642281	96.87
4199	55.062454	78.796783	76.516144	85.519310	67.212250	61.468903	62.32

4200 rows × 10 columns

In [ ]:
```
# Let's extract predictions with highest probabilites and detect what dig
predictions_ = np.argmax(predictions, axis=1)
```

We have correctly identified below image as 9

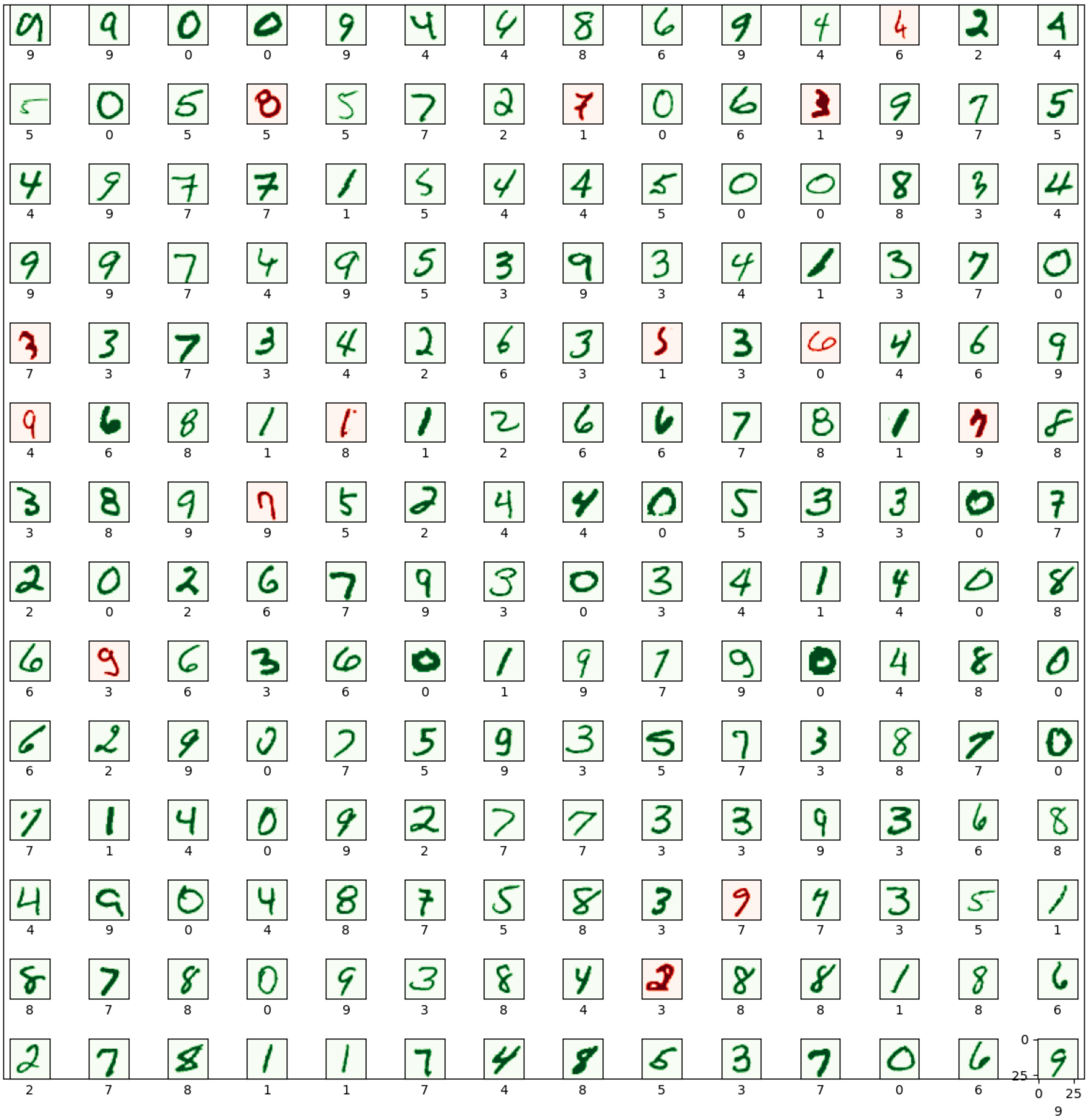
In [ ]:
```
plt.imshow(X_test_re[0], cmap=plt.cm.binary)
plt.show()
```



In [ ]:
```
# green colour depicts the correct prediction and red colour depicts the
numbers_to_display = 196
num_cells = math.ceil(math.sqrt(numbers_to_display))
plt.figure(figsize=(15, 15))

for plot_index in range(numbers_to_display):
    predicted_label = predictions_[plot_index]
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    color_map = 'Greens' if predicted_label == y_test_re[plot_index] else
    plt.subplot(num_cells, num_cells, plot_index + 1)
    plt.imshow(X_test_re[plot_index], cmap=color_map)
    plt.xlabel(predicted_label)

plt.subplots_adjust(hspace=1, wspace=0.5)
plt.show()
```



In [ ]:
```
test_loss, test_accuracy = model.evaluate(X_test,y_test)
```

```
132/132 [==============================] - 0s 358us/step - loss: 0.3237 -
accuracy: 0.9362
```

In [ ]:
```
print(f'Validation set accuracy: {test_accuracy}')
```

```
Validation set accuracy: 0.9361904859542847
```
