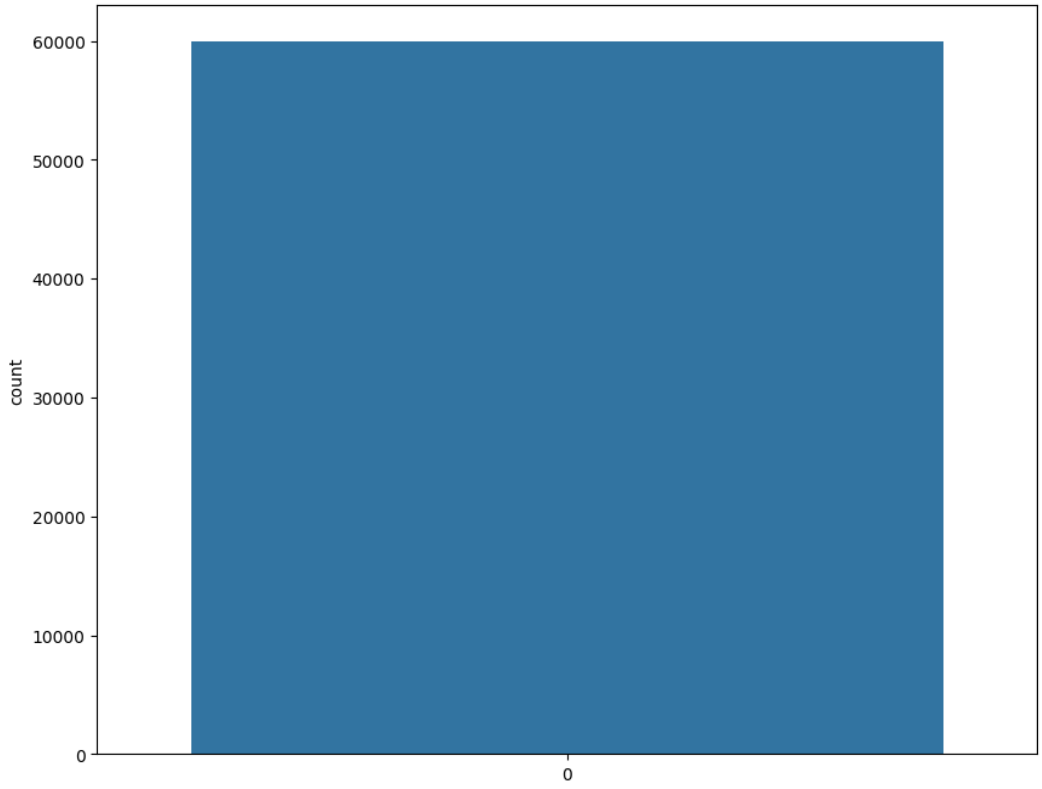
```
#Importing packages
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns


#loading dataset
(x_train,y_train),(x_test,y_test)=tf.keras.datasets.mnist.load_data()
```

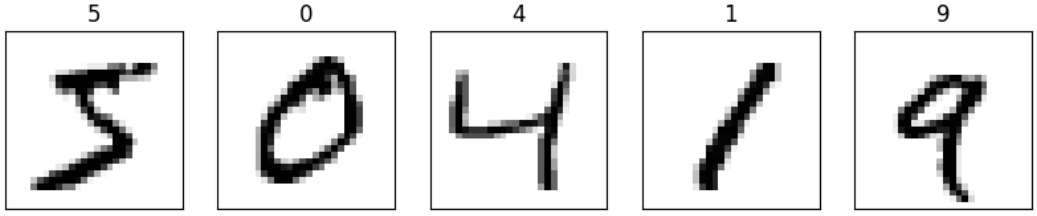
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
plt.figure(figsize = (10,8))
sns.countplot(y_train)
```

```
<Axes: ylabel='count'>
```




```
#Dataset properties
# Display some images
fig, axes = plt.subplots(ncols=5, sharex=False,
    sharey=True, figsize=(10, 4))
for i in range(5):
    axes[i].set_title(y_train[i])
    axes[i].imshow(x_train[i], cmap='gray_r')
    axes[i].get_xaxis().set_visible(False)
    axes[i].get_yaxis().set_visible(False)
plt.show()
```




```
# Pre-processing the data
print('Training images shape : ',x_train.shape)
print('Testing images shape : ',x_test.shape)
```

```
Training images shape :  (60000, 28, 28)
Testing images shape :  (10000, 28, 28)
```

```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
#applying normalization
x_train=x_train/255.0
x_testg=x_test/255.0
num_classes = 10


from tensorflow import keras
from keras.layers import Dense
from keras.models import Sequential, load_model


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten,Activation
from tensorflow.keras.layers import Conv2D,MaxPooling2D
from tensorflow.keras.layers import BatchNormalization


model = Sequential()

model.add(Conv2D(128, kernel_size=(3, 3),
                 activation=tf.nn.relu,
                 input_shape=input_shape))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3), activation=tf.nn.relu))
model.add(BatchNormalization())
model.add(Dropout(0.3))

model.add(Conv2D(32, (3, 3), activation=tf.nn.relu))
model.add(BatchNormalization())
model.add(Dropout(0.3))


model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation=tf.nn.softmax))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 128)       1280

 batch_normalization (BatchN  (None, 26, 26, 128)      512
 ormalization)

 dropout (Dropout)           (None, 26, 26, 128)       0

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        73792

 batch_normalization_1 (Batc  (None, 24, 24, 64)       256
 hNormalization)

 dropout_1 (Dropout)         (None, 24, 24, 64)        0

 conv2d_2 (Conv2D)           (None, 22, 22, 32)        18464

 batch_normalization_2 (Batc  (None, 22, 22, 32)       128
 hNormalization)

 dropout_2 (Dropout)         (None, 22, 22, 32)        0

 max_pooling2d (MaxPooling2D  (None, 11, 11, 32)       0
 )

 dropout_3 (Dropout)         (None, 11, 11, 32)        0

 flatten (Flatten)           (None, 3872)              0

 dense (Dense)               (None, 128)               495744

 dropout_4 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 591,466
```

```
Trainable params: 591,018
Non-trainable params: 448
_________________________________________________________________
```

```
# Train the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
history=model.fit(x=x_train,
                  y=y_train,
                  validation_split=0.1,
                  epochs=4)
```

```
Epoch 1/4
1688/1688 [==============================] - 732s 431ms/step - loss: 0.2556 - accuracy: 0.9258 - val_loss: 0.0936 - val_accuracy: 0
Epoch 2/4
1688/1688 [==============================] - 714s 423ms/step - loss: 0.1107 - accuracy: 0.9682 - val_loss: 0.1394 - val_accuracy: 0
Epoch 3/4
1688/1688 [==============================] - 690s 409ms/step - loss: 0.0923 - accuracy: 0.9741 - val_loss: 0.1313 - val_accuracy: 0
Epoch 4/4
1688/1688 [==============================] - 686s 406ms/step - loss: 0.0748 - accuracy: 0.9783 - val_loss: 0.1076 - val_accuracy: 0
```

```
model.save('MNproject.h5')
from tensorflow.keras.models import load_model
model = load_model('MNproject.h5')
```

```
# Evaluate the model
loss_and_acc=model.evaluate(x_test,y_test)
print("Test Loss", loss_and_acc[0])
print("Test Accuracy", loss_and_acc[1])
```

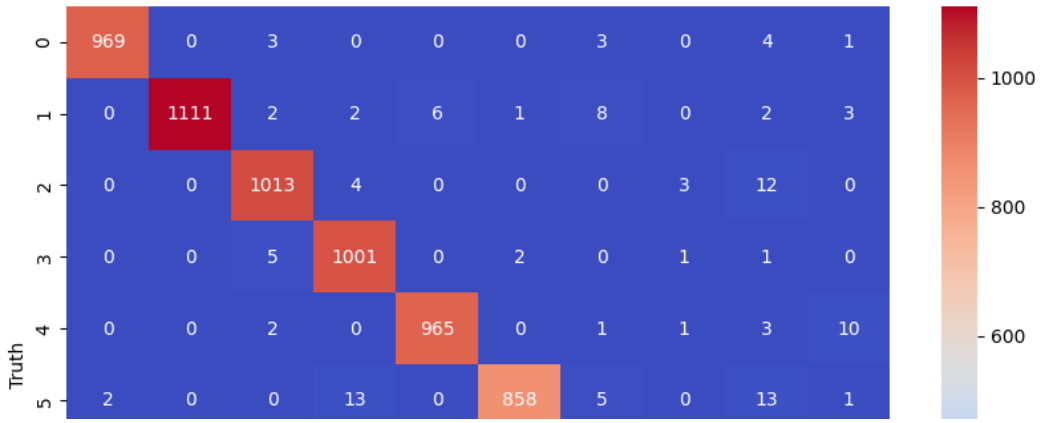
```
313/313 [==============================] - 25s 78ms/step - loss: 4.2186 - accuracy: 0.9806
Test Loss 4.218614101409912
Test Accuracy 0.9805999994277954
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 6))
ax[0].plot(epochs, acc, 'y', label='Training accuracy')
ax[0].plot(epochs, val_acc, 'g', label='Validation accuracy')
ax[0].legend(loc=0)
ax[1].plot(epochs, loss, 'y', label='Training loss')
ax[1].plot(epochs, val_loss, 'g', label='Validation loss')
ax[1].legend(loc=0)

plt.suptitle('Training and validation')
plt.show()
```

```
# Confusion Matrix
y_predicted = model.predict(x_test)
y_predicted_labels = [np.argmax(i) for i in y_predicted]
cm = tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
cm
```

```
plt.figure(figsize = (10,7))
sns.heatmap(cm, annot=True, fmt='d', cmap = 'coolwarm')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

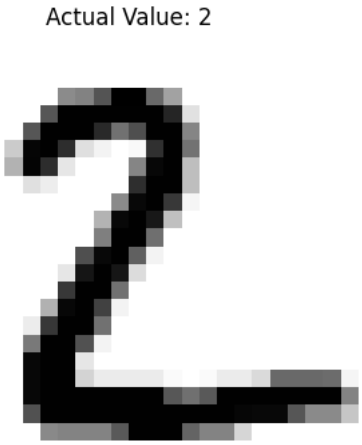
```
Text(95.72222222222221, 0.5, 'Truth')
```




```
# Testing the Model
plt.imshow(x_test[7],cmap='gray_r')
plt.title('Actual Value: {}'.format(y_test[7]))
prediction=model.predict(x_test)

plt.axis('off')
print('Predicted Value: ',np.argmax(prediction[7]))
if(y_test[7]==(np.argmax(prediction[7]))):
  print('Successful prediction')
else:
  print('Unsuccessful prediction')
```

```
plt.imshow(x_test[1],cmap='gray_r')
plt.title('Actual Value: {}'.format(y_test[1]))
prediction=model.predict(x_test)
plt.axis('off')
print('Predicted Value: ',np.argmax(prediction[1]))
if(y_test[1]==(np.argmax(prediction[1]))):
  print('Successful prediction')
else:
  print('Unsuccessful prediction')
```

```
313/313 [==============================] - 25s 80ms/step
Predicted Value:  2
Successful prediction
```



✓ 25s completed at 10:28 PM