Import necessary packages

```
import tensorflow as tf
from tensorflow.keras import layers,models
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

Loading dataset

```
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
```

Scaling the train and test dataset

```
X_train = X_train / 255
X_test = X_test / 255
```

```
X_train[0]
```

```
array([[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.        , 0.01176471, 0.07058824, 0.07058824,
        0.07058824, 0.49411765, 0.53333333, 0.68627451, 0.10196078,
        0.65098039, 1.        , 0.96862745, 0.49803922, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.11764706, 0.14117647,
        0.36862745, 0.60392157, 0.66666667, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.88235294, 0.6745098 ,
        0.99215686, 0.94901961, 0.76470588, 0.25098039, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.19215686, 0.93333333, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
        0.99215686, 0.99215686, 0.98431373, 0.36470588, 0.32156863,
        0.32156863, 0.21960784, 0.15294118, 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.07058824, 0.85882353, 0.99215686,
        0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.77647059,
        0.71372549, 0.96862745, 0.94509804, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.31372549, 0.61176471,
        0.41960784, 0.99215686, 0.99215686, 0.80392157, 0.04313725,
```

```
0           0 16862745  0 60392157  0           0
```

```
X_train.shape
```

```
(60000, 28, 28)
```

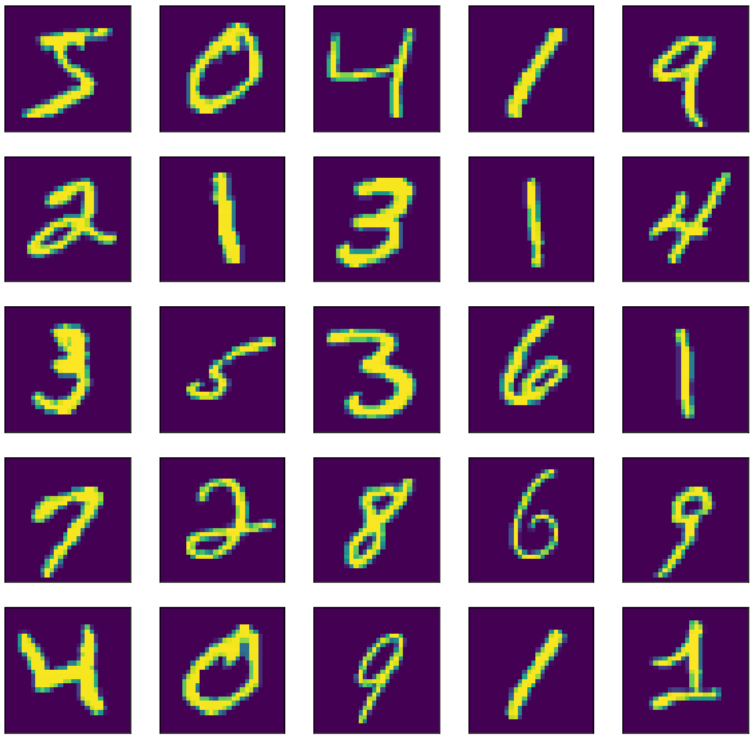
```
X_train = X_train.reshape(-1,28,28,1)
X_train.shape
```

```
(60000, 28, 28, 1)
```

```
X_test = X_test.reshape(-1,28,28,1)
X_test.shape
```

```
(10000, 28, 28, 1)
```

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(X_train[i])
plt.show()
```



Creating and Training a Convolutional Neural Network

```
convolutional_neural_network = models.Sequential([
    layers.Conv2D(filters=25, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
convolutional_neural_network.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 25)        250

 max_pooling2d_3 (MaxPooling  (None, 13, 13, 25)       0
 2D)

 conv2d_4 (Conv2D)           (None, 11, 11, 64)        14464

 max_pooling2d_4 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_5 (Conv2D)           (None, 3, 3, 64)          36928

 max_pooling2d_5 (MaxPooling  (None, 1, 1, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 64)                0

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 56,452
Trainable params: 56,452
Non-trainable params: 0
_________________________________________________________________
```

```
convolutional_neural_network.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history=convolutional_neural_network.fit(X_train, y_train, epochs=10,validation_data=(X_test, y_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 61s 32ms/step - loss: 0.0110 - accuracy: 0.9964 - val_loss: 0.0628 - val_accuracy: 0.9
Epoch 2/10
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0101 - accuracy: 0.9967 - val_loss: 0.0662 - val_accuracy: 0.9
Epoch 3/10
1875/1875 [==============================] - 52s 28ms/step - loss: 0.0095 - accuracy: 0.9968 - val_loss: 0.0691 - val_accuracy: 0.9
Epoch 4/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0075 - accuracy: 0.9975 - val_loss: 0.0680 - val_accuracy: 0.9
Epoch 5/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0083 - accuracy: 0.9973 - val_loss: 0.0653 - val_accuracy: 0.9
Epoch 6/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0085 - accuracy: 0.9974 - val_loss: 0.0637 - val_accuracy: 0.9
Epoch 7/10
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0071 - accuracy: 0.9977 - val_loss: 0.0614 - val_accuracy: 0.9
Epoch 8/10
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0057 - accuracy: 0.9982 - val_loss: 0.0772 - val_accuracy: 0.9
Epoch 9/10
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0083 - accuracy: 0.9974 - val_loss: 0.0688 - val_accuracy: 0.9
Epoch 10/10
1875/1875 [==============================] - 50s 27ms/step - loss: 0.0079 - accuracy: 0.9977 - val_loss: 0.0571 - val_accuracy: 0.9
```

Evaluating the CNN model

```
convolutional_neural_network.evaluate(X_test, y_test)
```

```
313/313 [==============================] - 2s 7ms/step - loss: 0.0571 - accuracy: 0.9900
[0.05712764710187912, 0.9900000095367432]
```

Making Predictions

```
y_predicted_by_model = convolutional_neural_network.predict(X_test)
y_predicted_by_model[0]   #getting probability score for each class digits
```

```
313/313 [==============================] - 3s 10ms/step
array([8.8857193e-13, 2.1868127e-09, 8.5338634e-09, 1.3332598e-10,
       2.4687311e-10, 2.1022958e-12, 6.8526713e-19, 9.9999994e-01,
       2.5362282e-10, 1.6787450e-09], dtype=float32)
```

```
np.argmax(y_predicted_by_model[0])
```

```
7
```

```
y_predicted_labels = [np.argmax(i) for i in y_predicted_by_model]
```

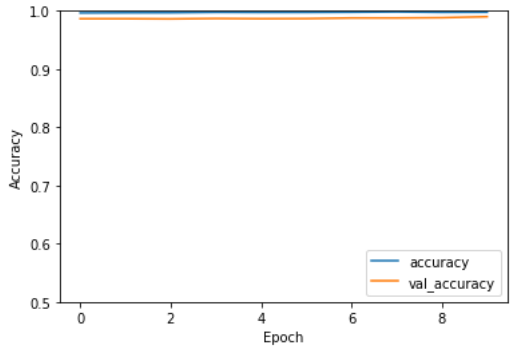
```
y_predicted_labels[:5]
```

```
[7, 2, 1, 0, 4]
```

```
plt.plot(history.history['accuracy'],label='accuracy')
plt.plot(history.history['val_accuracy'],label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = convolutional_neural_network.evaluate(X_test,
                                    y_test,
                                    verbose=2)
```

```
313/313 - 3s - loss: 0.0571 - accuracy: 0.9900 - 3s/epoch - 10ms/step
```




```
print('Test Accuracy is',test_acc)
```

```
Test Accuracy is 0.9900000095367432
```