```
#Importing packages
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img,
img_to_array
import cv2
np.random.seed(42)
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam, SGD

#Load Dataset
 (X_train, y_train), (X_test, y_test) =
tf.keras.datasets.mnist.load_data()
```

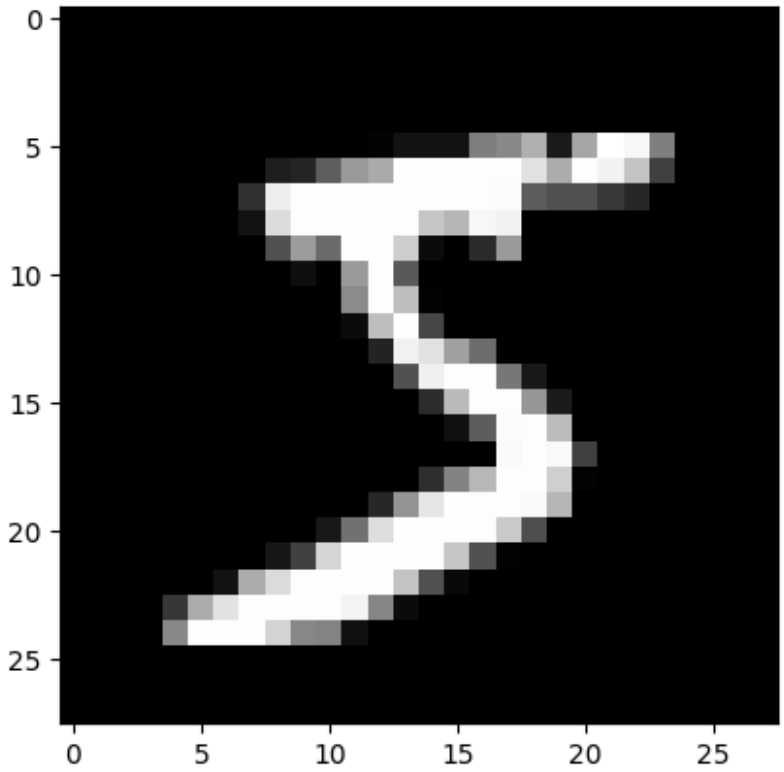
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
print('Total no of Images: ',X_train.shape[0])
print('Size of Image:', X_train.shape[1:])
print('Total no of labels:', y_train.shape)
```

```
Total no of Images:  60000
Size of Image: (28, 28)
Total no of labels: (60000,)
```

```
plt.imshow(X_train[0], cmap = plt.get_cmap('gray'))
print('Label:', y_train[0])
```

```
Label: 5
```




```
X_train = X_train.reshape((X_train.shape[0],-1))
X_test = X_test.reshape((X_test.shape[0], -1))

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

print(X_train.shape, X_test.shape)
```

```
(60000, 784) (10000, 784)
```

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

# print(X_train[0])
X_train.shape
```

```
(60000, 784)
```

```
from tensorflow.keras.utils import to_categorical

# One-hot encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print(y_train.shape)
```

```
(60000, 10)
```

```
num_classes = y_test.shape[1]
num_pixels = 784

def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(256, input_dim=num_pixels, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))

    return model

# build the model
model = baseline_model()
model.summary()
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/
dense.py:93: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                   │ (None, 256)            │       200,960 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 64)             │        16,448 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 10)             │           650 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
```

```
 Total params: 218,058 (851.79 KB)

 Trainable params: 218,058 (851.79 KB)

 Non-trainable params: 0 (0.00 B)
```

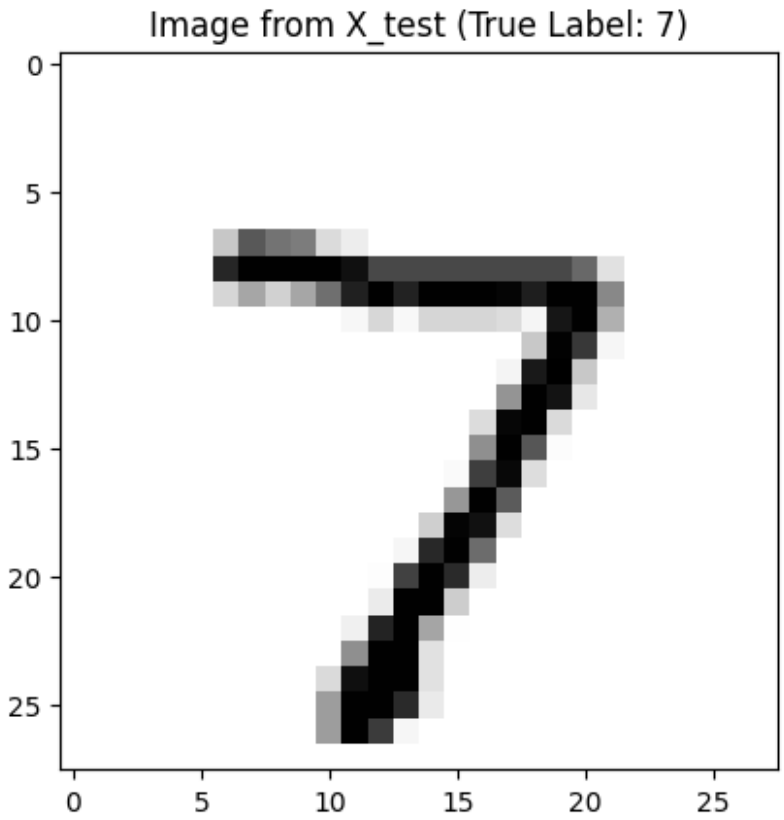
```
opt = SGD(learning_rate = 0.001)
model.compile(loss='categorical_crossentropy', optimizer= opt,
metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5, batch_size=32, verbose=1)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.3431 - loss:
2.0312
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7976 - loss:
0.9828
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8522 - loss:
0.6162
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8692 - loss:
0.4971
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8859 - loss:
0.4248

<keras.src.callbacks.history.History at 0x7a52165c0500>
```

```
scores = model.evaluate(X_test, y_test, verbose=1)
print("Error: %.2f%%" % (100-scores[1]*100))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8835 - loss:
0.4341
Error: 9.83%
```

```
img_width, img_height = 28, 28


image_to_predict_flat = X_test[0]

# Reshape the flattened image
image_to_predict_2d = image_to_predict_flat.reshape(img_width,
img_height)

# Get the true label for this image from y_test for comparison
true_label = np.argmax(y_test[0])

# Display the image
plt.imshow(image_to_predict_2d, cmap='Greys')
plt.title(f"Image from X_test (True Label: {true_label})")
plt.show()

# Prepare the image for model prediction
x = np.expand_dims(image_to_predict_flat, axis=0)

print(f"Prepared image shape for prediction: {x.shape}")
```




```
Prepared image shape for prediction: (1, 784)
```

```
import pickle


with open('model.pkl', 'wb') as file:

    pickle.dump(model, file)

print("Model saved successfully to model.pkl")
```

```
Model saved successfully to model.pkl
```

```
import os

file_path = 'model.pkl'
file_size_bytes = os.path.getsize(file_path)
file_size_mb = file_size_bytes / (1024 * 1024)


print(f"The size of 'model.pkl' is {file_size_mb:.2f} MB.")
```

```
The size of 'model.pkl' is 0.85 MB.
```

```
import pickle

with open('model.pkl', 'rb') as file:
    loaded_model = pickle.load(file)

print("Model loaded successfully.")
```

```
Model loaded successfully.
```

```
scores = loaded_model.evaluate(X_test, y_test, verbose=0)
print(f"Accuracy of the loaded model: {scores[1]*100:.2f}%")
```

```
Accuracy of the loaded model: 90.17%
```