MNIST

Modified National Institute of Standards and Technology which large database of handwritten digits that is commonly used for training various image processing systems.

```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
#used in jupyter notebook to display plots in the output itself rather than a separate wi
import numpy as np
import seaborn as sns
import pandas as pd
```

Data Collection

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

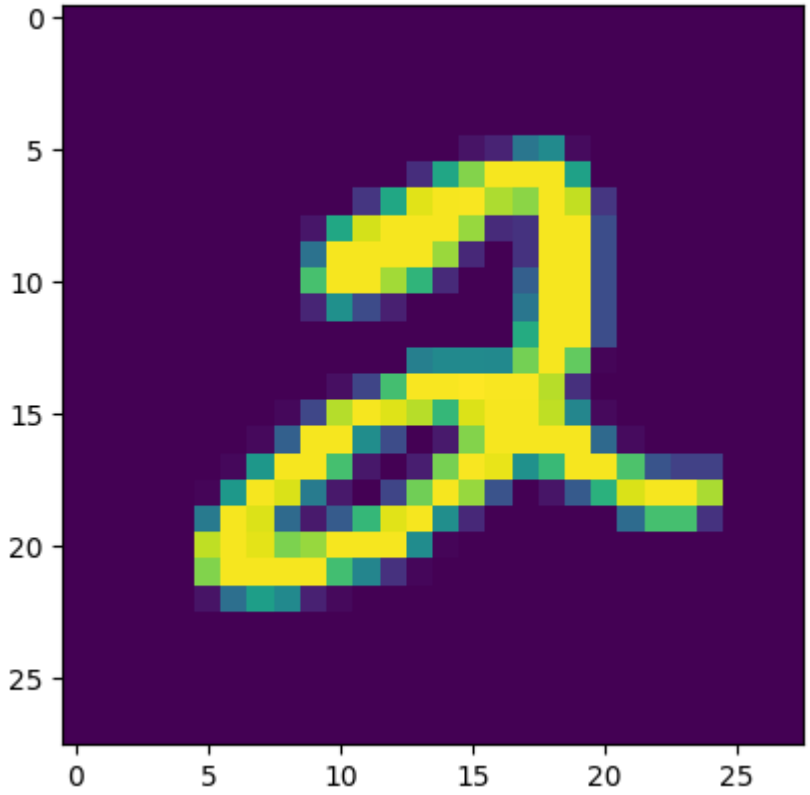
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mni
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

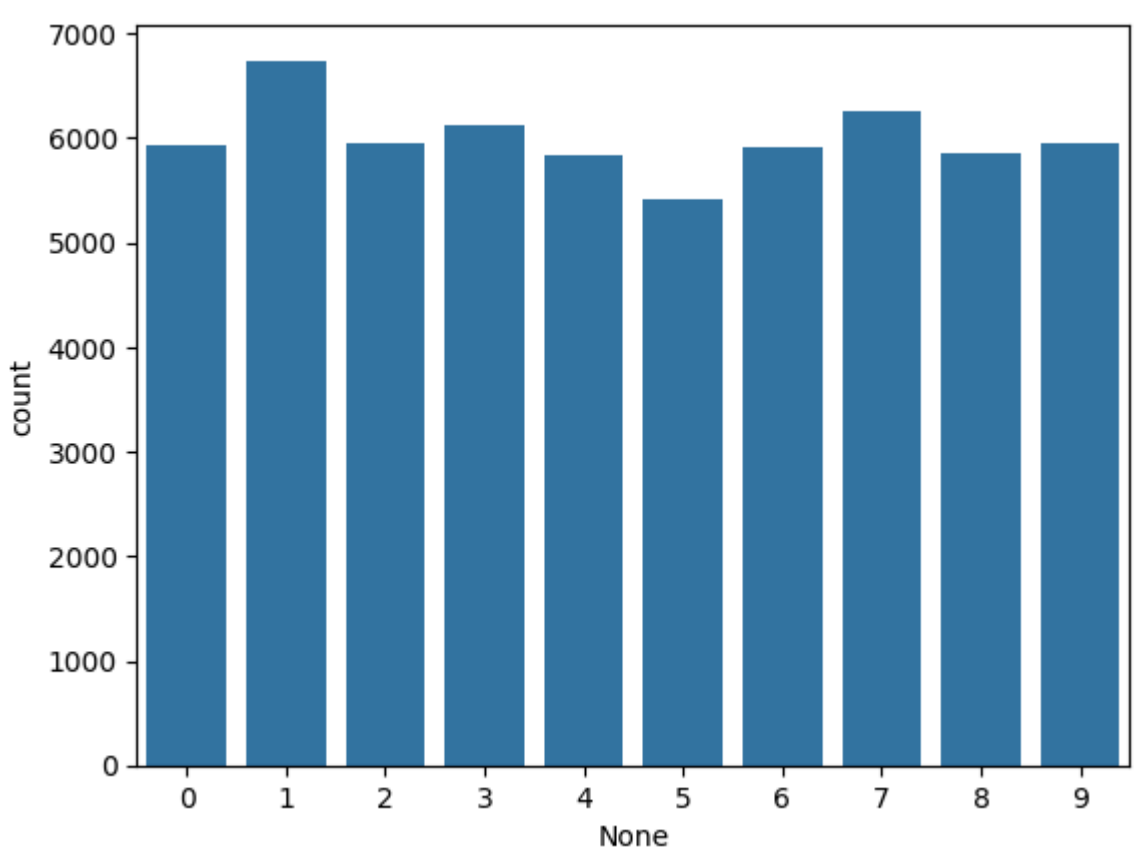
```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

```
plt.imshow(X_train[5])
print(X_train[5].shape)
```

```
(28, 28)
```




```
sns.countplot(x=pd.Series(y_train))
plt.show()
```




```
X_train[0][25]
```

```
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], dtype=uint8)
```

Normalization

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

```
X_train[0][5]
```

```
array([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.        ])
```

```
X_train.shape
```

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

```
X_train[0].shape
```

```
(28, 28)
```

```
X_train_matrix=X_train.reshape(-1,28*28)
```

```
print(X_train_matrix.shape)
X_test_matrix=X_test.reshape(-1,28*28)
print(X_test_matrix[0])
```

```
       0.         0.         0.         0.         0.         0.
       0.         0.         0.01176471 0.79607843 0.99607843 0.85882353
       0.1372549  0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.14901961 0.99607843 0.99607843 0.30196078 0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.12156863 0.87843137 0.99607843
       0.45098039 0.00392157 0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.52156863 0.99607843 0.99607843 0.20392157 0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.23921569 0.94901961
       0.99607843 0.99607843 0.20392157 0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.4745098  0.99607843 0.99607843 0.85882353
       0.15686275 0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.         0.         0.         0.         0.         0.
       0.4745098  0.99607843 0.81176471 0.07058824 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.        ]
```

```
model=keras.Sequential([
    keras.layers.Dense(128,input_shape=(784,),activation='relu'),
    keras.layers.Dense(128,input_shape=(784,),activation='relu'),
    keras.layers.Dense(10,input_shape=(784,),activation='softmax'),
])
print(model.summary())

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:93: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	100,480
dense_1 (Dense)	(None, 128)	16,512
dense_2 (Dense)	(None, 10)	1,290

Total params: 118,282 (462.04 KB)
Trainable params: 118,282 (462.04 KB)
Non-trainable params: 0 (0.00 B)
None

```
model.input_shape
```

```
(None, 784)
```

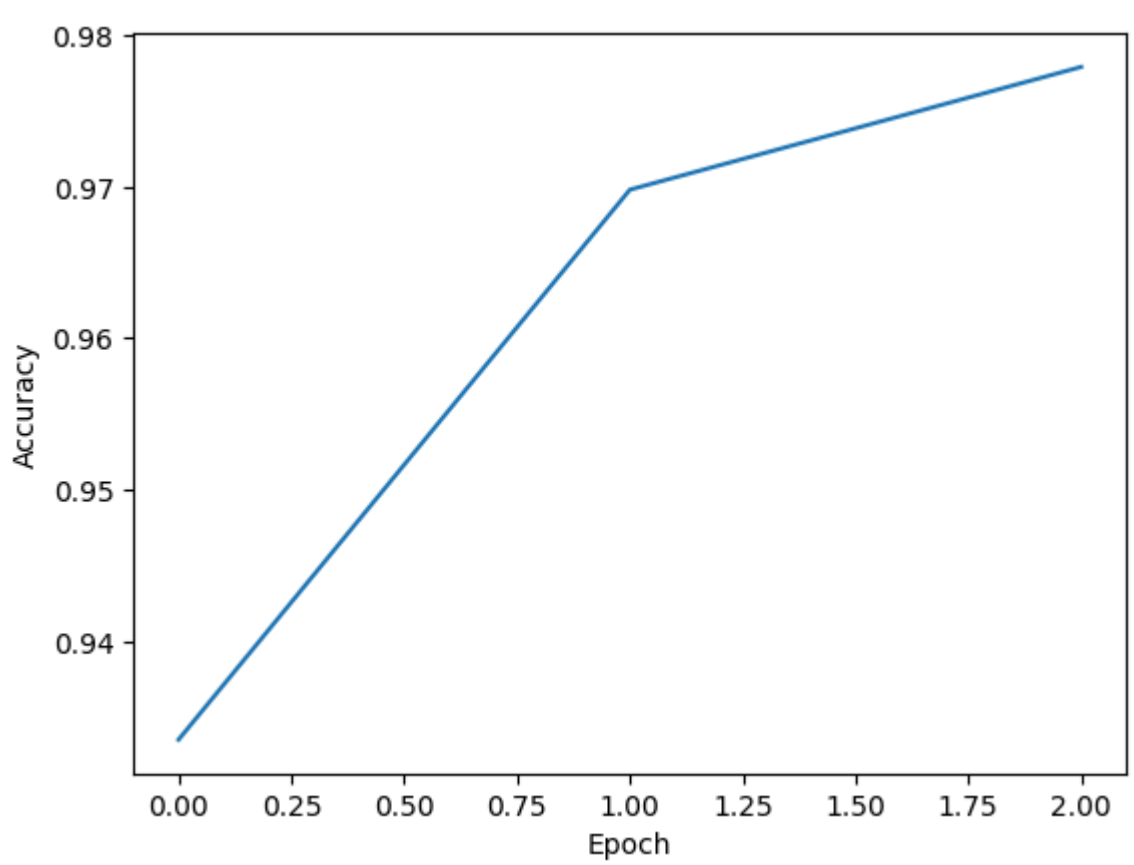
Double-click (or enter) to edit

```
history=model.fit(X_train_matrix,y_train,epochs=3,batch_size=32)
```

```
Epoch 1/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 12s 5ms/step - accuracy: 0.8848 - loss: 0.3909
Epoch 2/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9677 - loss: 0.1025
Epoch 3/3
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 3ms/step - accuracy: 0.9802 - loss: 0.0631
```

```
# model.fit(X_train_matrix,y_train,epochs=3,batch_size=64)
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
# plt.legend()
plt.show()
```




```
model.evaluate(X_test_matrix,y_test)
y_pred=model.predict(X_test_matrix)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9650 - loss: 0.1267
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step
```

```
print(y_test)
```

```
[7 2 1 ... 4 5 6]
```

```
print(y_pred[0])
```

```
[6.3993156e-08 1.9598531e-07 1.0310415e-06 2.1175309e-05 3.9872913e-10
 1.3407356e-07 1.2507753e-12 9.9964869e-01 5.4903957e-08 3.2858746e-04]
```

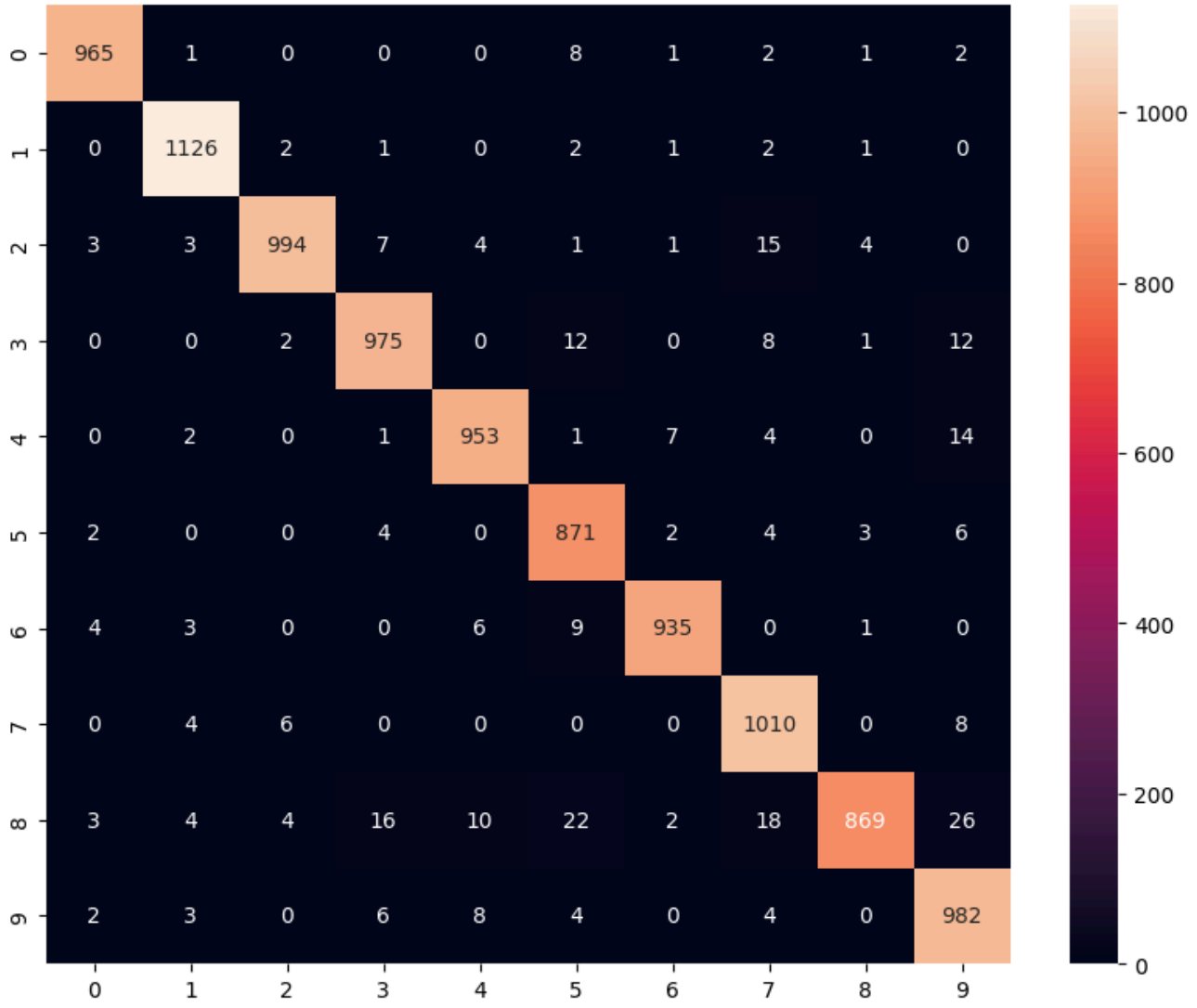
```
y_pred = np.argmax(y_pred, axis=1)
```

```
print(y_pred)
```

```
[7 2 1 ... 4 5 6]
```

```
confusion_mtx = tf.math.confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_mtx, annot=True, fmt='g')
```

```
<Axes: >
```




```
from PIL import Image as PILImage
import requests
from io import BytesIO
url = "https://drive.google.com/uc?export=view&id=1Km0zRlxQka6DBFzx0mdWsEhVpgJjv0pB"
response = requests.get(url)
img = PILImage.open(BytesIO(response.content))
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7cb00a389850>
```


```
img1 = img.convert('L').resize((28, 28))
plt.imshow(img1)
img2=255-np.array(img1)
plt.imshow(img2)
img_array = img2 / 255.0
img_array = img_array.reshape(1, 28*28)
```