CNN : CONVENTIONAL NEURAL NETWORKS IN TENSOR FLOW :

PAGE 140

In [ ]:

In [ ]:

1. STAGE 1 : INSTALLING DEPENDENCIES AND NOTEBOOK GPU SETUP:

In [1]:

```
# "pip install tensorflow"  - in ANACONDA POWERSHELL PROMPT :
# pip install matplotlib-venn(successfully installed)
# !apt-get -qq install -y libfluidsynth1(it didn't worked)
```

2. STAGE 2 : IMPORTING DEPENDENCIES FOR THE PROJECT :

PAGE 154 BOOK 4

In [41]:
```
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras.datasets import cifar10


from tensorflow.keras.layers import Conv2D, Dense, Flatten # i personally added this syntax

%matplotlib inline
tf.__version__
```

Out[41]:
```
'2.12.0'
```

STAGE 3 : DATA PREPROCESSING :

LOADING THE 'cifar10' DATASET :

PAGE 155

SETTING CLASS NAMES FOR THE DATASET :

In [42]:
```
class_names=['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']
```

LOADING THE DATASET :

PAGE 155

In [43]:
```
(X_train,y_train),(X_test,y_test)=cifar10.load_data()
```

IMAGE NORMALISATION :

PAGE 155 B0OK 4

In [44]:
```
X_train = X_train/255.0
```

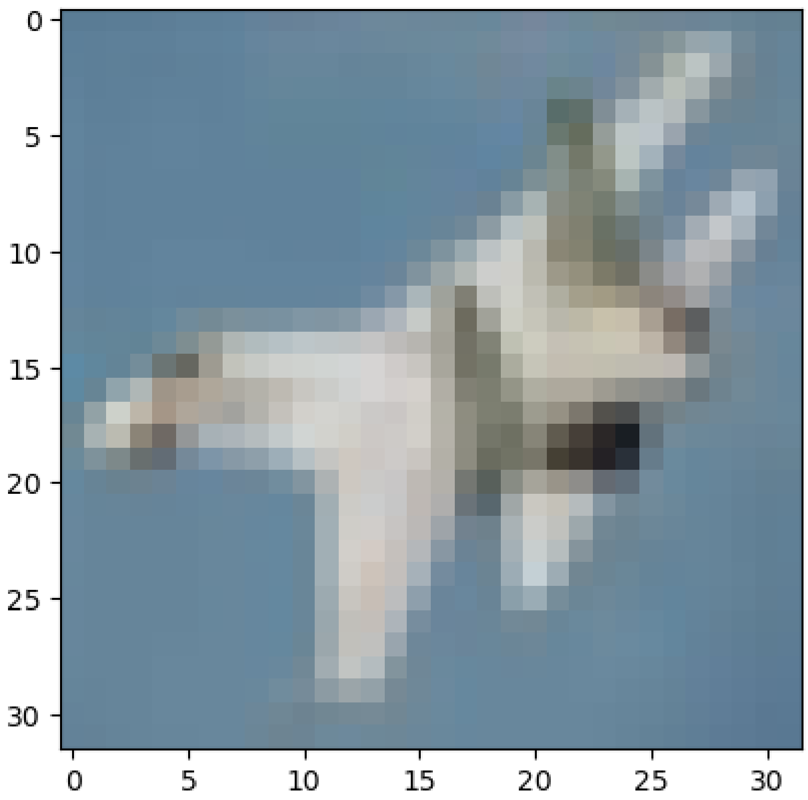
In [45]:
```
X_train.shape
```

Out[45]:
```
(50000, 32, 32, 3)
```

In [46]:
```
X_test = X_test/255.0
```

In [47]:
```
plt.imshow(X_test[10])
```

Out[47]: <matplotlib.image.AxesImage at 0x25508886880>



STAGE 4 : BUILDING A CONVOLUTIONAL NEURAL NETWORK :

DEFINING THE MODEL :

Sequential - 'S' Capital.

In [48]:
```
model = tf.keras.models.Sequential()
```

ADDING THE FIRST 'CNN' LAYER:

PAGE 157

#'CNN' layer hyper-parameters: filters : 32

kernel_size : 3

padding : same

activation : relu

input-shape : (32,32,3)

Conv2D - 'C' and 'D' Capitals

In [49]:
```
model.add(tf.keras.layers.Conv2D(filters = 32, kernel_size = 3, padding = "same", activation="relu",input_shape=[32,3
```

ADDING THE SECOND 'CNN' LAYER AND 'MAX POOL' LAYER :

PAGE 157

#'CNN' layer hyper-parameters:

filters : 32

kernel_size : 3

padding : same

activation : relu

Conv2D - 'C' and 'D' Capitals

#'Max Pool' layer hyper-parameters:

pool_size : 2

strides : 2

padding : valid

In [50]:
```
model.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding="same", activation="relu"))
```

In [51]:
```
model.add(tf.keras.layers.MaxPool2D(pool_size=2,strides=2, padding='valid'))
```

ADDING THE THIRD 'CNN' LAYER:

PAGE 158

#'CNN' layer hyper-parameters:

filters : 64

kernel_size : 3

padding : same

activation : relu

input-shape : (32,32,3)

Conv2D - 'C' and 'D' Capitals

In [52]:
```
model.add(tf.keras.layers.Conv2D(filters = 64, kernel_size= 3, padding = "same", activation = "relu"))
```

ADDING THE FOURTH 'CNN' LAYER AND 'MAX POOL' LAYER :

PAGE 158

#'CNN' layer hyper-parameters:

filters : 64

kernel_size : 3

padding : same

activation : relu

Conv2D - 'C' and 'D' Capitals

#'Max Pool' layer hyper-parameters:

pool_size : 2

strides : 2

padding : valid

In [53]:
```
model.add(tf.keras.layers.Conv2D(filters=64, kernel_size = 3, padding = "same", activation = "relu"))
```

In [54]:
```
model.add(tf.keras.layers.MaxPool2D(pool_size = 2, strides = 2, padding = 'valid'))
```

ADDING THE 'FLATTEN LAYER' :

PAGE 159

In [55]:
```
model.add(tf.keras.layers.Flatten())
```

ADDING THE FIRST DENSE LAYER :

#'DENSE' LAYER HYPER-PARAMETERS :

units/neurons : 128

activation : Softmax

In [56]:
```
model.add(tf.keras.layers.Dense(units = 128, activation = 'Softmax'))
```

ADDING THE SECOND DENSE LAYER : (OUTPUT LAYER)

#'DENSE' LAYER HYPER-PARAMETERS :

units/neurons : 10 (NUNBER OF CLASSES)

activation : Softmax

In [57]:
```
model.add(tf.keras.layers.Dense(units = 10, activation = 'Softmax'))
```

In [58]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 32, 32, 32)        896       
                                                                 
 conv2d_3 (Conv2D)           (None, 32, 32, 32)        9248      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (None, 16, 16, 64)        18496     
                                                                 
 conv2d_5 (Conv2D)           (None, 16, 16, 64)        36928     
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 8, 8, 64)         0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 4096)              0         
                                                                 
 dense_2 (Dense)             (None, 128)               524416    
                                                                 
 dense_3 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 591,274
Trainable params: 591,274
Non-trainable params: 0
_________________________________________________________________
```

STAGE 5 : COMPILING THE MODEL :

PAGE 160

#sparse_categorical_accuracy -> checks to see if the maximal true value is equal to the Index of the maximal predicted value.

In [59]:
```
model.compile(loss = "sparse_categorical_crossentropy", optimizer = "Adam", metrics = ["sparse_categorical_accuracy"]
```

STAGE 6 : TRAINING THE MODEL :

PAGE 160

In [60]:
```
model.fit(X_train, y_train,epochs = 5)
```

```
Epoch 1/5
1563/1563 [==============================] - 137s 86ms/step - loss: 2.2603 - sparse_categorical_accuracy: 0.1442
Epoch 2/5
1563/1563 [==============================] - 133s 85ms/step - loss: 2.2833 - sparse_categorical_accuracy: 0.1205
Epoch 3/5
1563/1563 [==============================] - 134s 86ms/step - loss: 2.3029 - sparse_categorical_accuracy: 0.0965
Epoch 4/5
1563/1563 [==============================] - 133s 85ms/step - loss: 2.3028 - sparse_categorical_accuracy: 0.0989
Epoch 5/5
1563/1563 [==============================] - 134s 86ms/step - loss: 2.3028 - sparse_categorical_accuracy: 0.0987
```

Out[60]:
```
<keras.callbacks.History at 0x25508a11580>
```

STAGE 7 : MODEL EVALUATION AND PREDICTION :

PAGE 161

In [61]:
```
test_loss, test_accuracy = model.evaluate(X_test,y_test)
```

```
313/313 [==============================] - 8s 25ms/step - loss: 2.3029 - sparse_categorical_accuracy: 0.1000
```

In [62]:
```
# Here we are identifying the Accurate test :

print("Test Accuracy : {}".format(test_accuracy))
```

```
Test Accuracy : 0.10000000149011612
```

In [ ]:

In [ ]:

HAND WRITTING DIGIT RECOGNIZER USING 'DEEP LEARNING' :

PAGE 162

In [63]:
```
# In this Project, we will try to identify Hand Writting digits by using the power of 'Deep Learning'.
# We will be Working on the 'MNIST' dataset to create a DEEP LEARNING CLASSIFICATION MODEL & See how our Model perfor
# in accurately Acurately Predicting Images with the Correct digit notation.

# 'MNIST' is a database of Hand Writting Digits made up of a Training Set of 60000 examples and a Testset of 10000 ex
# The Training examples are annotated by Humans wih the Correct Answer.
# For Instance, if the Handwritting digit is the number '3, then'3 is simply the label associated with that example.
```

In [64]:
```
# We will Train the model with the samples available in the training set and then use the test set to evaluate -
# how well our neural network has learned to recognise digits.

# Now let's create a Convolution Neural Network to Solve this Problem.
```

In [65]:
```
# The 'MNIST database' (Modified National Institute of Standards and Technology database)
# -> is a large database of handwritten digits that is commonly used for training various image processing systems.
```

1. 'import Libraries' :

page 163

In [66]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [67]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPool2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score
%matplotlib inline
```

In [68]:
```
# We can use the following code snippet to check which version of TensorFlow is installed in our system:
# This will print the version of TensorFlow that is currently installed on our system.
# For example, if we are running version 2.4.1 of TensorFlow, the below code will output "2.4.1".


tf.__version__
```

Out[68]:
```
'2.12.0'
```

In [ ]:
```

```

2. 'Import MNIST Dataset' :

page 164

In [90]:
```
from tensorflow.keras.datasets import mnist
```

In [70]:
```
# In this one also We have an a inbuilt dataset, Where the dataset shapes are 'X' 60000 of 10000 and 'y' 60000 of 100


(X_train,y_train),(X_test,y_test) = mnist.load_data()
print("shape of X_train:",X_train.shape)
print("shape of X_test:",X_test.shape)
print("shape of y_train:",y_train.shape)
print("shape of y_test:",y_test.shape)
```

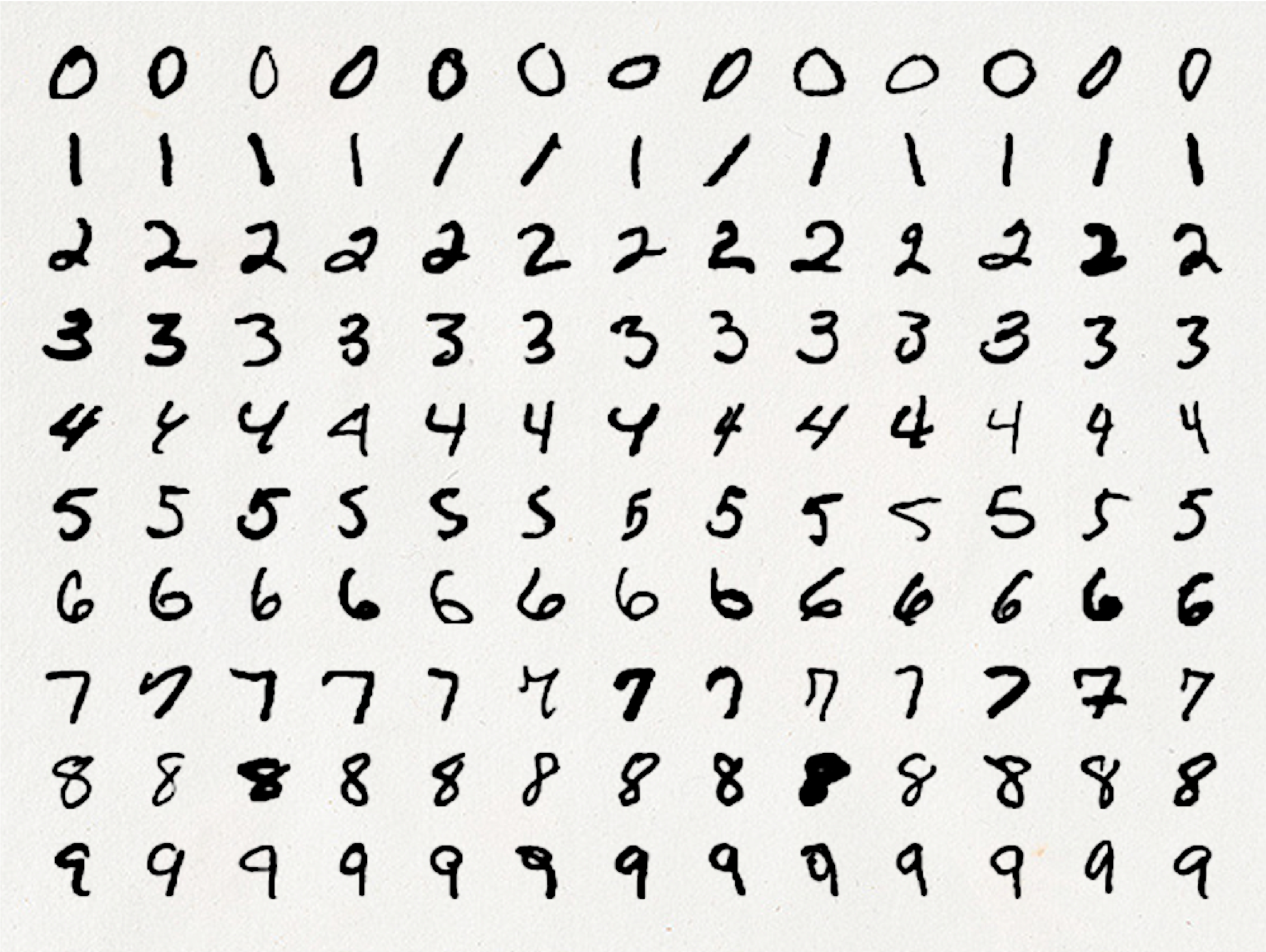
```
shape of X_train: (60000, 28, 28)
shape of X_test: (10000, 28, 28)
shape of y_train: (60000,)
shape of y_test: (10000,)
```

3. Each MNIST Image is in GrayScale and consists of 28*28 pixels :

page 164

In [71]:
```
from PIL import Image
Image.open('mnist.png')
```

Out[71]:



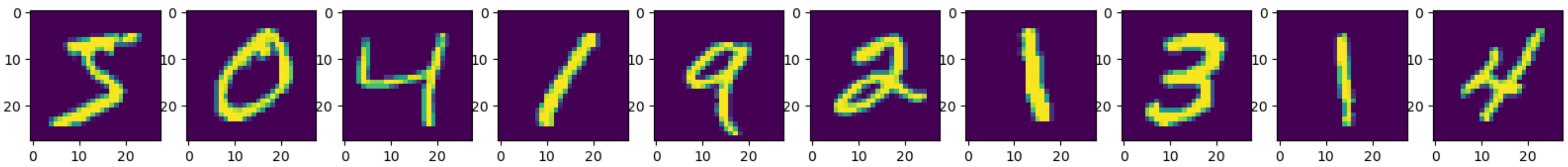
4. Load Sample Image from the 'MNIST' Dataset :

page 165

In [72]:
```
plt.figure(figsize=(20,20))

plt.subplot(1,10,1)
plt.imshow(X_train[0])
plt.subplot(1,10,2)
plt.imshow(X_train[1])
plt.subplot(1,10,3)
plt.imshow(X_train[2])
plt.subplot(1,10,4)
plt.imshow(X_train[3])
plt.subplot(1,10,5)
plt.imshow(X_train[4])
plt.subplot(1,10,6)
plt.imshow(X_train[5])
plt.subplot(1,10,7)
plt.imshow(X_train[6])
plt.subplot(1,10,8)
plt.imshow(X_train[7])
plt.subplot(1,10,9)
plt.imshow(X_train[8])
plt.subplot(1,10,10)
plt.imshow(X_train[9])
```

Out[72]:
```
<matplotlib.image.AxesImage at 0x2550a00c3d0>
```



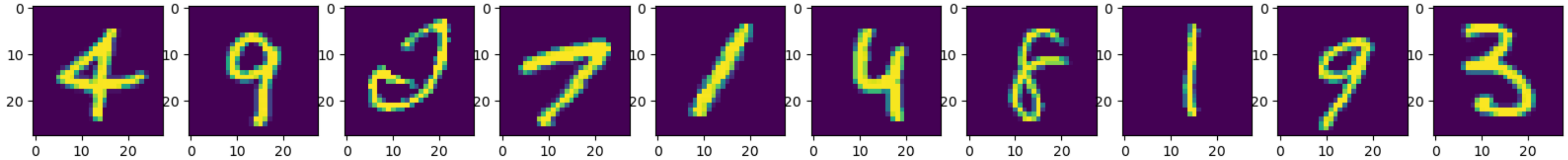
In [73]:
```
# Here,This particular data, in the image pattern,we are changing in to an a pixel.

# The image data cannot be fed directly into the model.
# So, we need to perform some operations and process the data to make it ready for our Neural Networks.
# The Dimensions of the Training data is (60000,28,28).
# The CNN Model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).
# This extra dimension is for the Color Channel for Grayscale images like MNIST, it's value is 1.
# For Color images, the Channel Value is '3' Corresponding to 'Red,Green & Blue(RGB)'.
```

In [74]:
```
plt.figure(figsize=(20,20))

plt.subplot(1,10,1)
plt.imshow(X_train[150])
plt.subplot(1,10,2)
plt.imshow(X_train[162])
plt.subplot(1,10,3)
plt.imshow(X_train[178])
plt.subplot(1,10,4)
plt.imshow(X_train[193])
plt.subplot(1,10,5)
plt.imshow(X_train[205])
plt.subplot(1,10,6)
plt.imshow(X_train[3978])
plt.subplot(1,10,7)
plt.imshow(X_train[456])
plt.subplot(1,10,8)
plt.imshow(X_train[7896])
plt.subplot(1,10,9)
plt.imshow(X_train[57])
plt.subplot(1,10,10)
plt.imshow(X_train[31897])
```

Out[74]: <matplotlib.image.AxesImage at 0x2550e4363d0>



5. DATA PREPROCESSING :

PAGE 166

In [75]:
```
print("Shape X_train: ", X_train.shape)
print("Shape X_test: ", X_test.shape)
```

```
Shape X_train:  (60000, 28, 28)
Shape X_test:  (10000, 28, 28)
```

In [76]:
```
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)
print("Shape of X_train: ", X_train.shape)
print("Shape of X_test: ", X_test.shape)
```

```
Shape of X_train:  (60000, 28, 28, 1)
Shape of X_test:  (10000, 28, 28, 1)
```

6. ONE HOT ENCODING OF TARGET LABELS :

PAGE 167

CATEGORICAL DATA : (NON-NUMERICAL)

In [77]:
```
# We are going to use 'OHE(NON-NUMERICAL)' as a Sample Tool to encode information used inside Neural Networks.
# In many applications it is Convinent to Transform Categorical(non-numerical) features into 'Numerical Variables'.
# For Instance, The Categorical Feature 'digit' with Value 'd' in [0 to 9] can be encoded into 'Binary Vector' with 10
# Which always has '0' Value, except the 'd'th Position, Where a '1' is present.

# For Example, the digit '3' can be encoded as [0,0,0,1,0,0,0,0,0,0],
# This type of Representation is called 'ONE-HOT ENCODING(OHE)', OR Sometimes Simply ONE-HOT, and is very Common in da
# -mining when the Learning Algorithm is Specialized in dealing with 'Numerical Functions'.
```

In [78]:
```
y_train[0:11]
```

Out[78]:
```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3], dtype=uint8)
```

In [79]:
```
y_cat_train = to_categorical(y_train,10)
y_cat_test = to_categorical(y_test,10)
y_cat_train[0:11]
```

Out[79]:
```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

7. SCALING FEATURE DATA :

PAGE 169

In [80]:
```
# NEURAL NETWORKS Works Well, When the Feature Values lie between '0 to 1'.

# Hence, We will Scale the dataset by Simply Dividing Each Value by 255.

# The Value for Each Pixel is case of 'Gray Scale Images range from '0(white)' to '255(Black)'.
```

In [81]:
```
X_train[0]
```

Out[81]:
```
array([[[  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
```

In [82]:
```
# And Here, We are 'Normalising the Data' :

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

X_train/=255
X_test/=255
```

In [83]:
```
X_train[0]
```

```
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.3137255 ],
        [0.6117647 ],
        [0.41960785],
        [0.99215686],
        [0.99215686],
        [0.8039216 ],
        [0.04313726],
        [0.        ],
        [0.16862746],
        [0.6039216 ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
```

8. MODEL CREATION :

PAGE 170

In [84]:
```
# Now we will Create our CNN Model.
# A CNN Model generally consists of Convolutional and Pooling layers.
# It Works better for Data that are represented as Grid Structures, this is the reason why CNN Works well,
# for Image Classification Problems.
```

In [95]:
```
model = Sequential()

model.add(Conv2D(32,kernel_size = (3,3), activation = 'relu', input_shape = input_shape))
model.add(Conv2D(64, (3,3), activation = 'relu' ))
model.add(MaxPool2D(pool_size = (2,2)))
model.add(Dropout(0.25))
model.add(Flatten())

model.add(Dense(256,activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(10,activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

9. Model Summary :

page 171

In [96]:
```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_8 (Conv2D)           (None, 26, 26, 32)        320       
                                                                 
 conv2d_9 (Conv2D)           (None, 24, 24, 64)        18496     
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 12, 12, 64)       0         
 2D)                                                             
                                                                 
 dropout_6 (Dropout)         (None, 12, 12, 64)        0         
                                                                 
 flatten_4 (Flatten)         (None, 9216)              0         
                                                                 
 dense_8 (Dense)             (None, 256)               2359552   
                                                                 
 dropout_7 (Dropout)         (None, 256)               0         
                                                                 
 dense_9 (Dense)             (None, 10)                2570      
                                                                 
=================================================================
Total params: 2,380,938
Trainable params: 2,380,938
Non-trainable params: 0
_________________________________________________________________
```

10. ADDING 'EARLY STOPPING' :

PAGE 171

In [97]:
```
early_stop = EarlyStopping(monitor = 'val_loss', patience = 2)
```

11. MODEL TRAINING :

PAGE 172

In [98]:
```
# And if we observe cross check, it looks like, Model training what we have seen in the 'ANN'.
```

In [99]:
```
model.fit(X_train, y_cat_train, epochs = 50,callbacks = [early_stop], validation_data = (X_test, y_cat_test))

print("The Model has Successfully Trained ")
model.save('mnist.h5')
print("Saving the Model as mnist.h5 ")
```

```
Epoch 1/50
1875/1875 [==============================] - 102s 54ms/step - loss: 0.1270 - accuracy: 0.9601 - val_loss: 0.0402 - v
al_accuracy: 0.9869
Epoch 2/50
1875/1875 [==============================] - 106s 57ms/step - loss: 0.0476 - accuracy: 0.9850 - val_loss: 0.0349 - v
al_accuracy: 0.9889
Epoch 3/50
1875/1875 [==============================] - 112s 60ms/step - loss: 0.0325 - accuracy: 0.9898 - val_loss: 0.0296 - v
al_accuracy: 0.9889
Epoch 4/50
1875/1875 [==============================] - 107s 57ms/step - loss: 0.0246 - accuracy: 0.9921 - val_loss: 0.0256 - v
al_accuracy: 0.9922
Epoch 5/50
1875/1875 [==============================] - 122s 65ms/step - loss: 0.0186 - accuracy: 0.9935 - val_loss: 0.0399 - v
al_accuracy: 0.9887
Epoch 6/50
1875/1875 [==============================] - 139s 74ms/step - loss: 0.0170 - accuracy: 0.9944 - val_loss: 0.0323 - v
al_accuracy: 0.9908
The Model has Successfully Trained
Saving the Model as mnist.h5
```

12. MODEL PERFORMANCE DURING TRAINING & VALIDATION :

PAGE 172

In [104]:
```
training_metrics = pd.DataFrame(model.history.history)
training_metrics.columns
```

Out[104]:
```
Index(['loss', 'accuracy', 'val_loss', 'val_accuracy'], dtype='object')
```

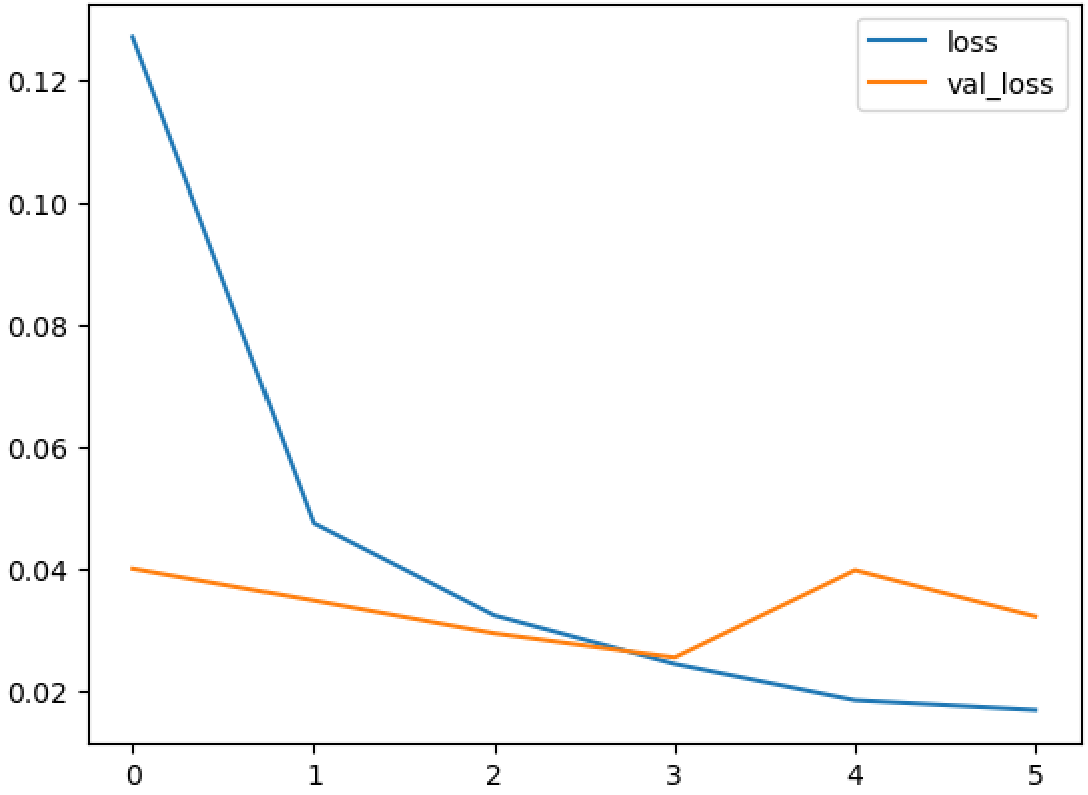
In [105]:
```
training_metrics.head()
```

Out[105]:

	loss	accuracy	val_loss	val_accuracy
0	0.127006	0.960133	0.040150	0.9869
1	0.047592	0.985017	0.034949	0.9889
2	0.032518	0.989833	0.029571	0.9889
3	0.024573	0.992100	0.025625	0.9922
4	0.018638	0.993533	0.039902	0.9887

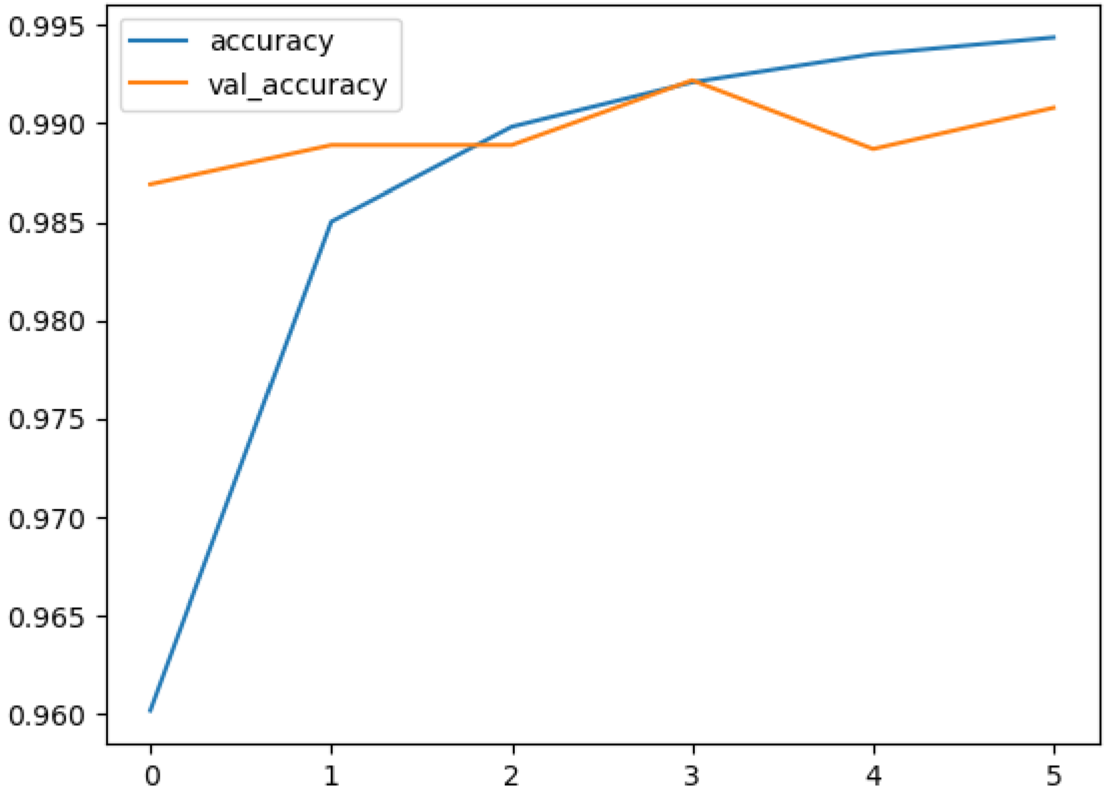
In [106]:
```
training_metrics[['loss','val_loss']].plot()
```

Out[106]: <AxesSubplot:>



In [107]:
```
training_metrics[['accuracy', 'val_accuracy']].plot()
```

Out[107]: <AxesSubplot:>



NOW ACCURACY TEST :

In [108]:
```
score = model.evaluate(X_test, y_cat_test, verbose = 0)
print('Test loss : ', score [0])
print('Test accuracy : ', score [1])
```

```
Test loss :  0.03234346956014633
Test accuracy :  0.9908000230789185
```

13. MODEL PREDICTIONS :

PAGE 174

In [113]:
```
predictions = np.argmax(model.predict(X_test), axis = -1)
print(classification_report(y_test, predictions))
```

```
313/313 [==============================] - 4s 13ms/step
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       1.00      0.99      1.00      1135
           2       1.00      0.99      0.99      1032
           3       0.98      1.00      0.99      1010
           4       0.99      0.98      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.98      0.99       958
           7       0.99      1.00      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.98      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

In [115]:
```
print(confusion_matrix(y_test,predictions))
```

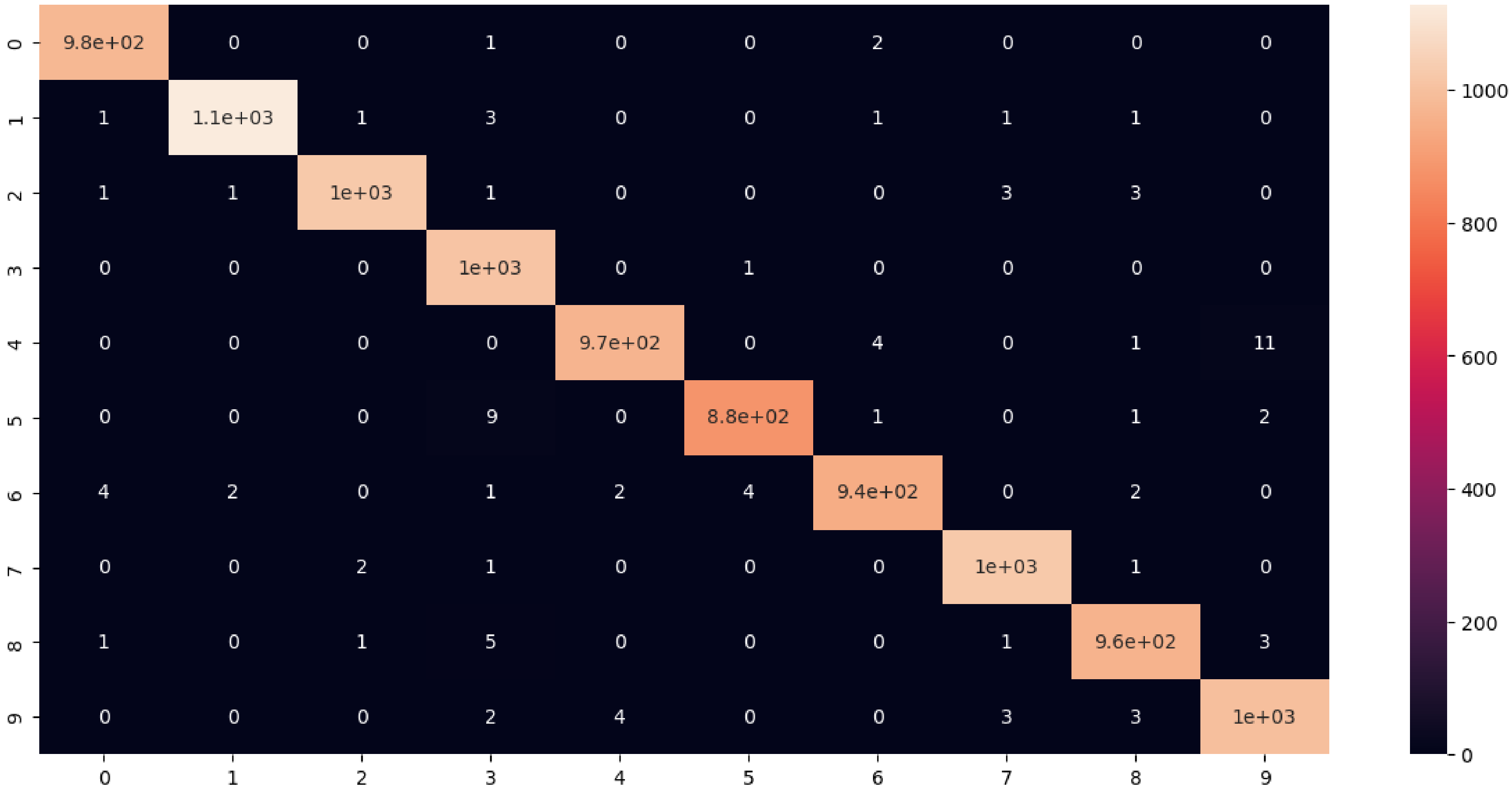
```
[[ 977    0    0    1    0    0    2    0    0    0]
 [   1 1127    1    3    0    0    1    1    1    0]
 [   1    1 1023    1    0    0    0    3    3    0]
 [   0    0    0 1009    0    1    0    0    0    0]
 [   0    0    0    0  966    0    4    0    1   11]
 [   0    0    0    9    0  879    1    0    1    2]
 [   4    2    0    1    2    4  943    0    2    0]
 [   0    0    2    1    0    0    0 1024    1    0]
 [   1    0    1    5    0    0    0    1  963    3]
 [   0    0    0    2    4    0    0    3    3  997]]
```

14. VISUALIZING CONFUSION MATRIX :

PAGE 175

In [118]:
```
plt.figure(figsize = (15,7))
sns.heatmap(confusion_matrix(y_test, predictions),annot=True)
```

Out[118]: <AxesSubplot:>



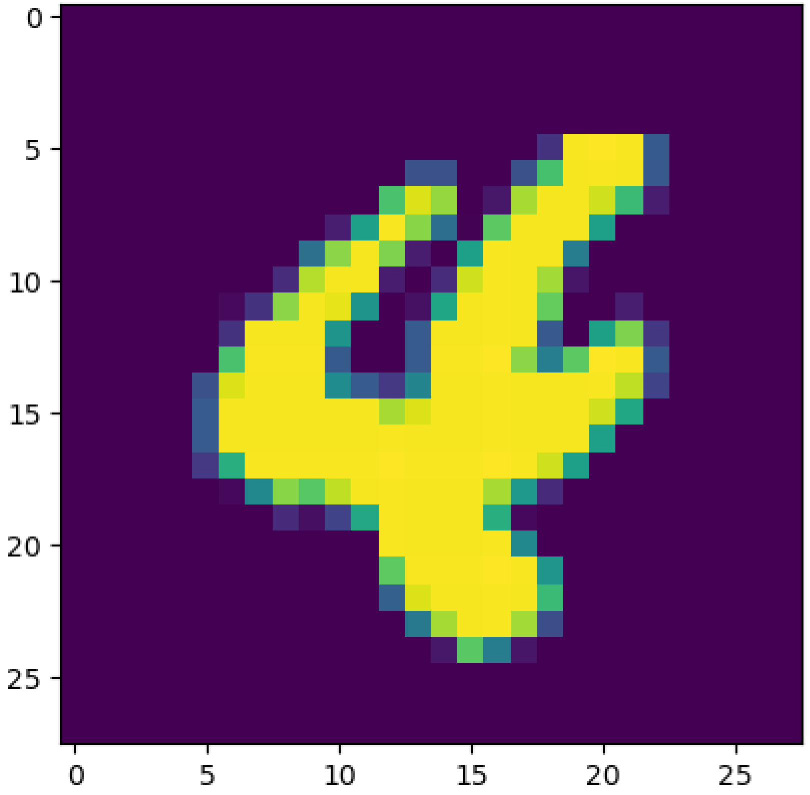
15. PREDICTING INDIVIDUAL IMAGES :

PAGE 175

In [120]:
```
# Here, This is predicting individually.
# It is showing the Image pattern.

new_img = X_test[95]
plt.imshow(new_img)
```

Out[120]: <matplotlib.image.AxesImage at 0x25509f71d90>



In [121]:
```
y_test[95]
```

Out[121]: 4

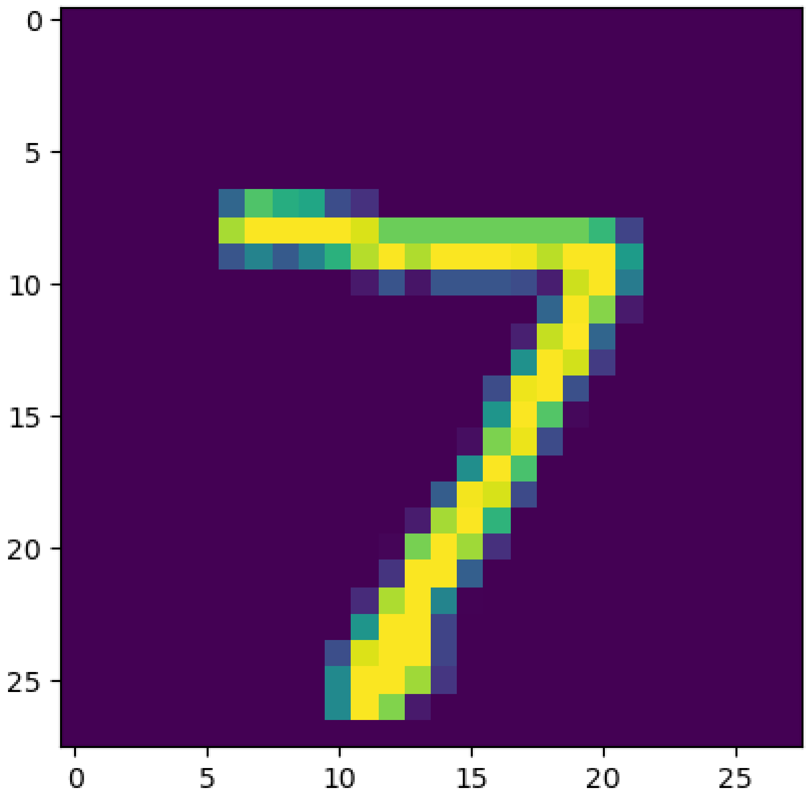
In [123]:
```
np.argmax(model.predict(new_img.reshape(1,28,28,1)),  axis = 1)
```

```
1/1 [==============================] - 0s 20ms/step
```

Out[123]:
```
array([4], dtype=int64)
```

In [124]:
```
new_img2 = X_test[0]
plt.imshow(new_img2)
```

Out[124]:
```
<matplotlib.image.AxesImage at 0x2550e529f10>
```



In [125]:
```
y_test[0]
```

Out[125]: 7

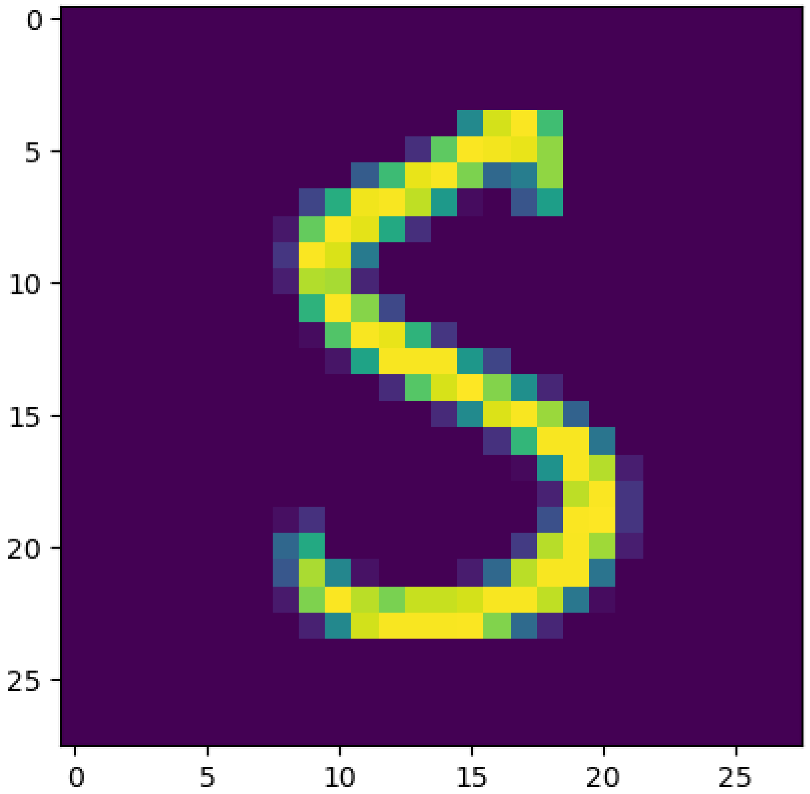
In [128]:
```
np.argmax(model.predict(new_img2.reshape(1,28,28,1)), axis = -1)
```

```
1/1 [==============================] - 0s 29ms/step
```

Out[128]: array([7], dtype=int64)

In [129]:
```
new_img3 = X_test[397]
plt.imshow(new_img3)
```

Out[129]: <matplotlib.image.AxesImage at 0x2550e57c160>



In [130]:
```
np.argmax(model.predict(new_img3.reshape(1,28,28,1)), axis = 1)
```

```
1/1 [==============================] - 0s 40ms/step
```

Out[130]:
```
array([5], dtype=int64)
```

In [ ]:
```
```