ASSIGNMENT

- Name : Achal Rajesh Mate
- Roll No : 2203541
- Enroll No : MITU20BTCSD001
- Branch : CSE
- Class : TY CSE Is - 3
- Guided By : Prof Nagesh Jadhav Sir

Title:- : Implement CNN for MNIST/CIFAR10 dataset using tensorflow

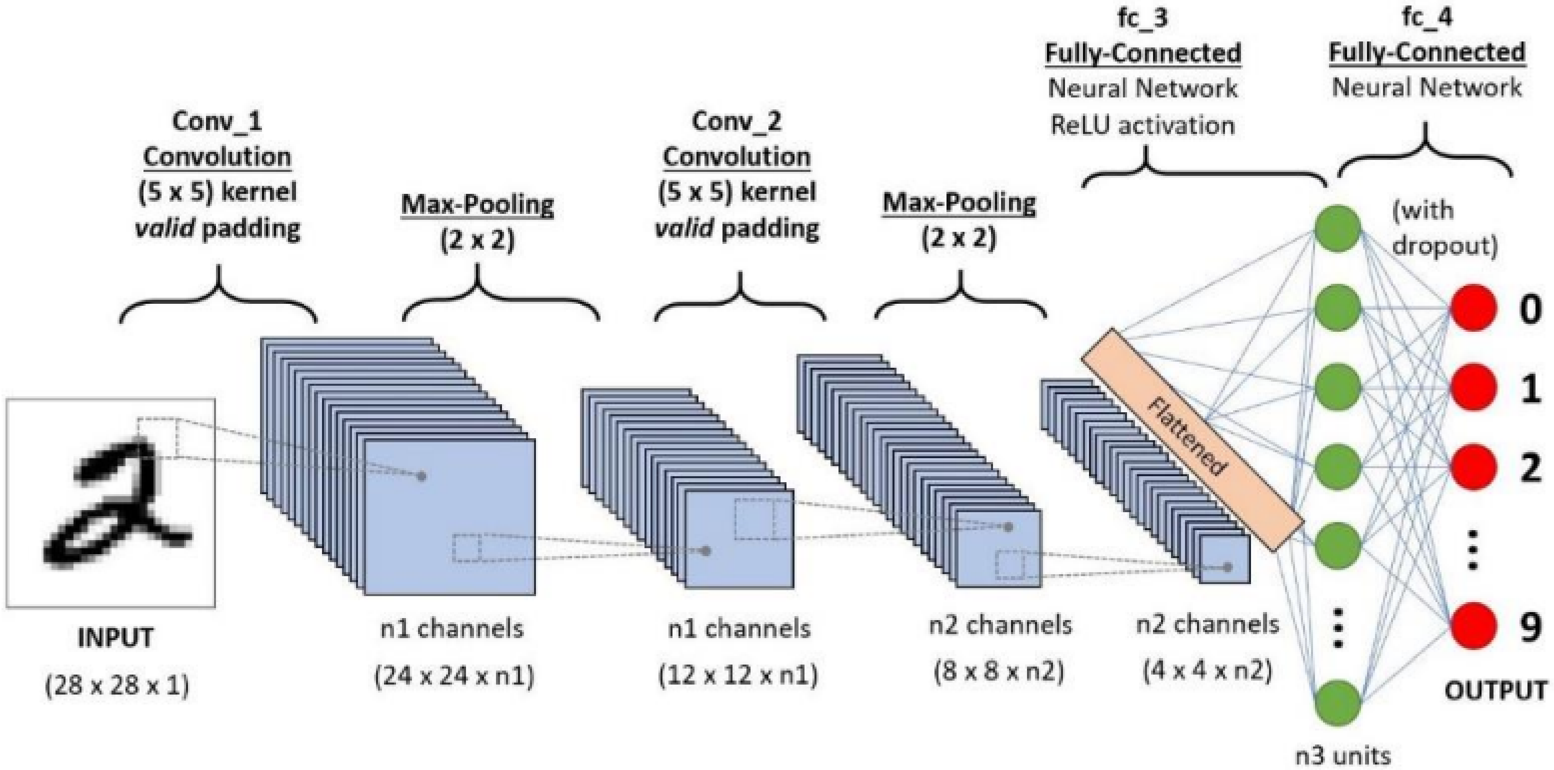
Objectives:-
1. To learn basics of deep learning
2. To learn and implement CNN

Theory:

Deep Learning

- Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making.
- Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled.
- Also known as deep neural learning or deep neural network.
- Deep learning, a subset of machine learning, utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning.
- The artificial neural networks are built like the human brain, with neuron nodes connected together like a web.
- ile traditional programs build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

CNN Architecture



CNN WORKING

A Convolutional Neural Networks Introduction so to speak.

* Step 1: Convolution Operation :-

The first building block in our plan of attack is convolution operation. In this step, we will touch on feature detectors, which basically serve as the neural network's filters. We will also discuss feature maps, learning the parameters of such maps, how patterns are detected, the layers of detection, and how the findings are mapped out.

* Step 1(b): ReLU Layer :-

The second part of this step will involve the Rectified Linear Unit or ReLU. We will cover ReLU layers and explore how linearity functions in the context of Convolutional Neural Networks. Not necessary for understanding CNN's, but there's no harm in a quick lesson to improve your skills.

* Step 2: Pooling :-

In this part, we'll cover pooling and will get to understand exactly how it generally works. Our nexus here, however, will be a specific type of pooling; max pooling. We'll cover various approaches, though, including mean (or sum) pooling. This part will end with a demonstration made using a visual interactive tool that will definitely sort the whole concept out for you.

* Step 3: Flattening :-

This will be a brief breakdown of the flattening process and how we move from pooled to flattened layers when working with Convolutional Neural Networks.

* Step 4: Full Connection :-

In this part, everything that we covered throughout the section will be merged together. By learning this, you'll get to envision a fuller picture of how Convolutional Neural Networks operate and how the "neurons" that are finally produced learn the classification of images.

Import All Necessary Files

In [2]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
```

In [27]:

```
import matplotlib.image as mpimg
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import load_model
import tensorflow as tf
import cv2 as cv
from google.colab.patches import cv2_imshow
```

Read the Dataset

In [3]:
```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

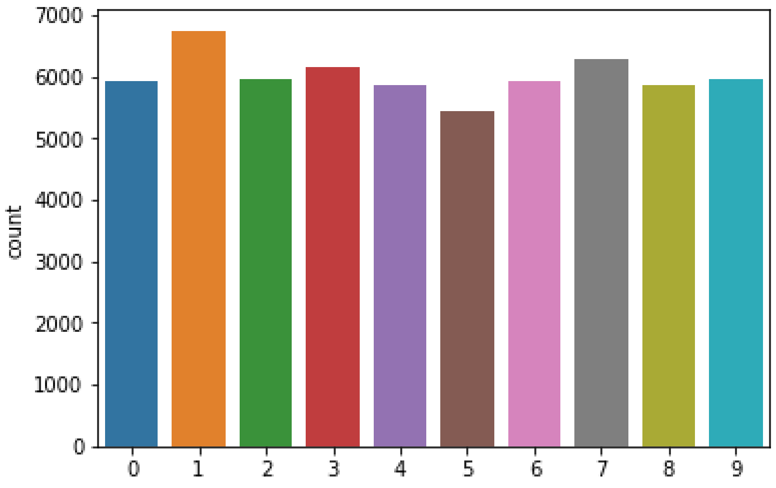
Display Dimensions of training and testing data

In [4]:
```
print('X Training shape: ',x_train.shape)
print('Y Training shape: ',y_train.shape)
print('X Testing shape: ',x_test.shape)
print('Y Testing shape: ',y_test.shape)
```

```
X Training shape:  (60000, 28, 28)
Y Training shape:  (60000,)
X Testing shape:  (10000, 28, 28)
Y Testing shape:  (10000,)
```

In [5]:
```
sns.countplot(y_train)
```

Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb770a55410>



In [6]:
```
x_train
```

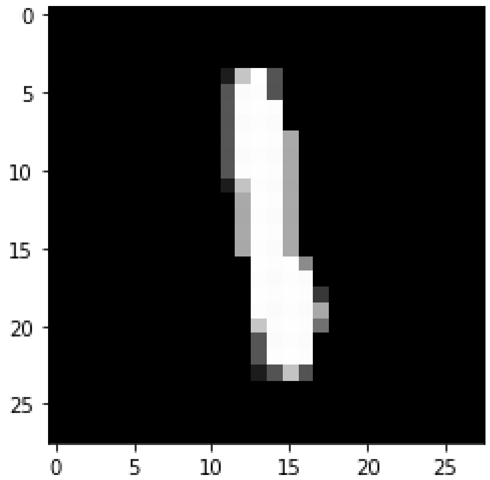
Out[6]:
```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[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, ..., 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]]], dtype=uint8)
```

Display any one image

In [7]:

```
plt.imshow(x_train[200], cmap='gray')
plt.show()
print("Digit in image is ",y_train[200])
```



```
Digit in image is  1
```

In [8]:

```
input_shape = (28,28,1)
```

View Dimension of image

In [9]:

```
x_train[200].shape
```

Out[9]:

```
(28, 28)
```

In [10]:

```
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
```

In [11]:

```
x_train = np.expand_dims(x_train,-1)
x_test = np.expand_dims(x_test,-1)
```

In [12]:

```
batch_size = 128
num_classes = 10
epochs = 5
```

Converts a class vector (integers) to binary class matrix

In [13]:

```
y_train = keras.utils.to_categorical(y_train, num_classes) .
y_test = keras.utils.to_categorical(y_test, num_classes)
```

In [14]:

```
# layer instances to the constructor:
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Dropout(0.5),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )
```

```
 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout_1 (Dropout)         (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

In [15]:

```
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

In [17]:

```
batch_size = 130
num_classes = 10
epochs = 10
```

In [18]:

```
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Epoch 1/10
416/416 [==============================] - 68s 163ms/step - loss: 0.0821 - accuracy: 0.9747 - val_loss: 0.0416 - val_accuracy: 0.9
887
Epoch 2/10
416/416 [==============================] - 64s 153ms/step - loss: 0.0781 - accuracy: 0.9753 - val_loss: 0.0386 - val_accuracy: 0.9
883
Epoch 3/10
416/416 [==============================] - 49s 118ms/step - loss: 0.0728 - accuracy: 0.9772 - val_loss: 0.0380 - val_accuracy: 0.9
897
Epoch 4/10
416/416 [==============================] - 46s 111ms/step - loss: 0.0693 - accuracy: 0.9781 - val_loss: 0.0355 - val_accuracy: 0.9
908
Epoch 5/10
416/416 [==============================] - 45s 109ms/step - loss: 0.0660 - accuracy: 0.9791 - val_loss: 0.0358 - val_accuracy: 0.9
902
Epoch 6/10
```

```
416/416 [==============================] - 47s 112ms/step - loss: 0.0673 - accuracy: 0.9791 - val_loss: 0.0343 - val_accuracy: 0.9
915
Epoch 7/10
416/416 [==============================] - 45s 109ms/step - loss: 0.0616 - accuracy: 0.9805 - val_loss: 0.0317 - val_accuracy: 0.9
912
Epoch 8/10
416/416 [==============================] - 47s 112ms/step - loss: 0.0599 - accuracy: 0.9809 - val_loss: 0.0313 - val_accuracy: 0.9
912
Epoch 9/10
416/416 [==============================] - 46s 111ms/step - loss: 0.0564 - accuracy: 0.9821 - val_loss: 0.0318 - val_accuracy: 0.9
915
Epoch 10/10
416/416 [==============================] - 46s 111ms/step - loss: 0.0565 - accuracy: 0.9820 - val_loss: 0.0285 - val_accuracy: 0.9
920
```

In [19]:

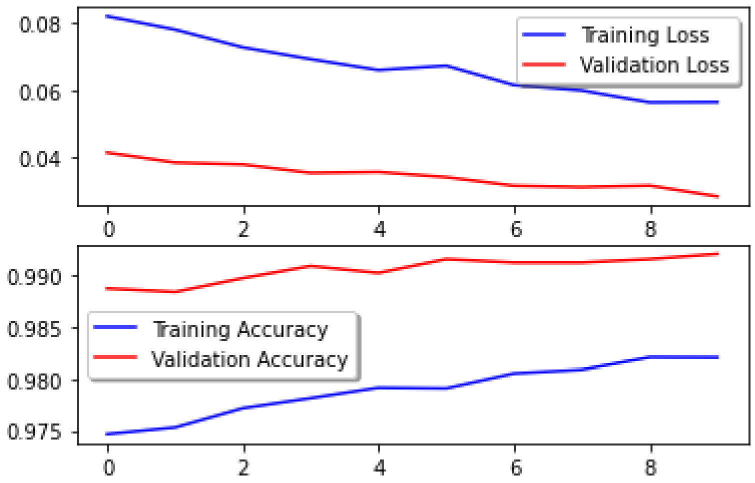
```python
print(history.history)
```

```
{'loss': [0.08208952844142914, 0.07807175070047379, 0.07279050350189209, 0.06928857415914536, 0.06604007631540298, 0.0673191398382
1869, 0.06156732887029648, 0.05994338542222977, 0.05640175938606262, 0.056481070816516876], 'accuracy': [0.9746666550636292, 0.975
3147959709167, 0.9771666526794434, 0.9781296253204346, 0.9791296124458313, 0.9790740609169006, 0.9804999828338623, 0.9808518290519
714, 0.9820555448532104, 0.9820370078086853], 'val_loss': [0.04157629981637001, 0.03859942778944969, 0.038046903908252716, 0.03553
558140993118, 0.03579990193247795, 0.03426143527030945, 0.03171133995056152, 0.03130418062210083, 0.03175002709031105, 0.028525330
126285553], 'val_accuracy': [0.9886666536331177, 0.9883333444595337, 0.9896666407585144, 0.9908333420753479, 0.9901666641235352,
0.9915000200271606, 0.9911666512489319, 0.9911666512489319, 0.9915000200271606, 0.9919999837875366]}
```

View Model Performace

In [20]:

```python
fig, ax = plt.subplots(2,1)
ax[0].plot(history.history['loss'], color='b', label="Training Loss")
ax[0].plot(history.history['val_loss'], color='r', label="Validation Loss",axes =ax[0])
legend = ax[0].legend(loc='best', shadow=True)

ax[1].plot(history.history['accuracy'], color='b', label="Training Accuracy")
ax[1].plot(history.history['val_accuracy'], color='r',label="Validation Accuracy")
legend = ax[1].legend(loc='best', shadow=True)
```



In [21]:

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 3s 10ms/step - loss: 0.0291 - accuracy: 0.9904
```

In [22]:

```
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert testing observations to one hot vectors
Y_true = np.argmax(y_test,axis = 1)
# compute the confusion matrix
confusion_mtx = tf.math.confusion_matrix(Y_true, Y_pred_classes)
```

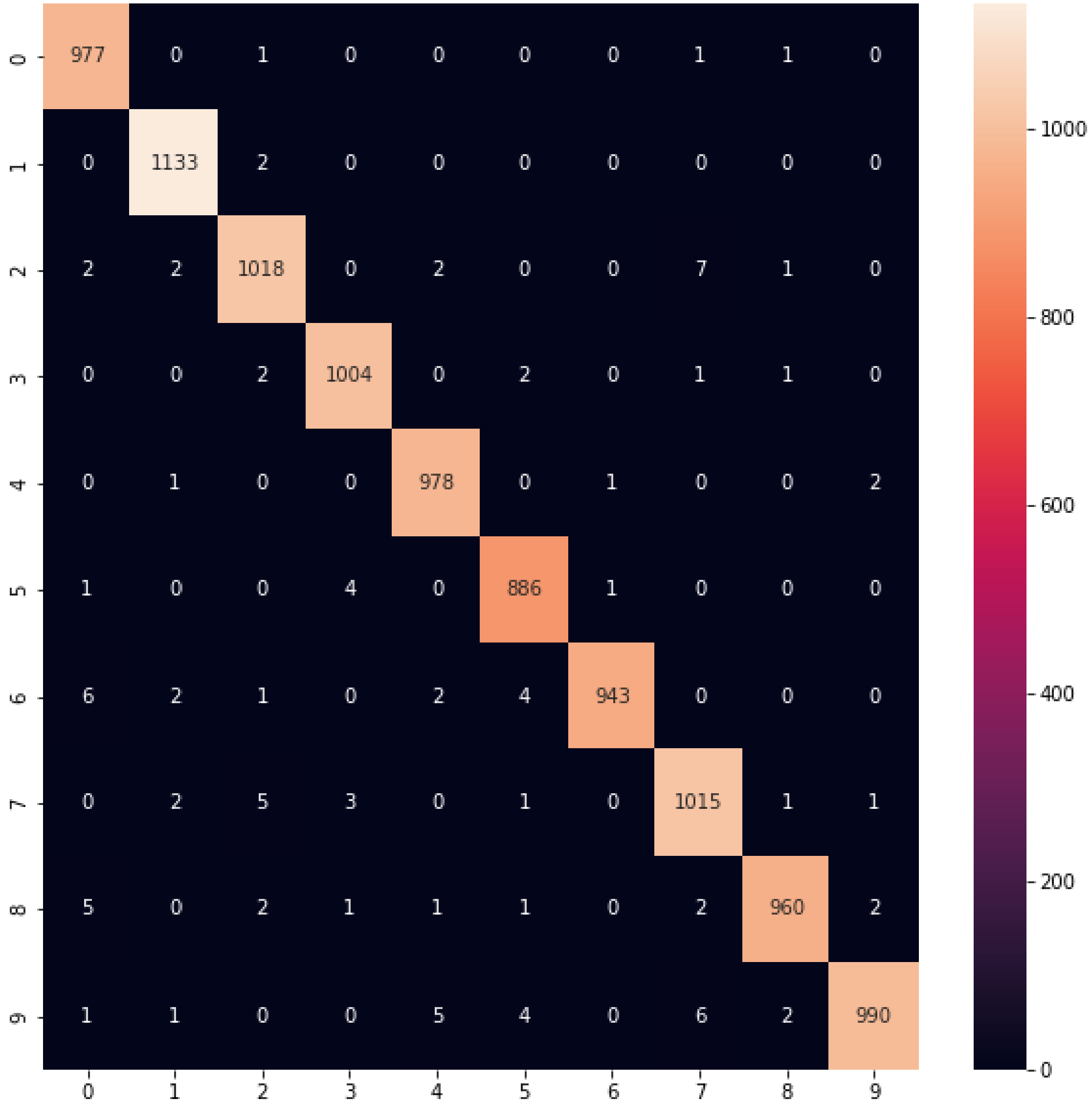
Plot Confusion Matrix

In [25]:

```
plt.figure(figsize=(10,10))
sns.heatmap(confusion_mtx, annot=True, fmt='g')
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb76b83e090>
```



In [31]:

```
import cv2 as cv
from google.colab.patches import cv2_imshow
```

Read new image for testing

In [34]:
```
img = cv.imread('five.jfif')
cv2_imshow(img)
```

View Dimension of image

In [35]:
```
img.shape
```

Out[35]:
```
(233, 216, 3)
```

In [37]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Convert image into black grey

In [38]:
```
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

In [39]:
```
gray.shape
```

Out[39]:
```
(233, 216)
```

Change dimension of image

In [40]:

```
img_rs = cv.resize(gray, (28, 28))
```

In [41]:

```
img_rs.shape
```

Out[41]: (28, 28)

Display the image

In [42]:

```
cv2_imshow(img_rs)
```

In [43]:

```
img_rs = np.expand_dims(img_rs,0)
```

In [44]:

```
img_rs.shape
```

Out[44]: (1, 28, 28)

In [45]:

```
img_rs = np.expand_dims(img_rs,-1)
```

In [46]:

```
img_rs.shape
```

Out[46]: (1, 28, 28, 1)

make prediction

In [47]:

```
num = model.predict(img_rs)
num
```

Out[47]:
```
array([[0.96098435, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.03901567, 0.        ]],
      dtype=float32)
```

In [48]:
```
rs = [0,1,2,3,4,5,6,7,8,9]
```

In [49]:
```
from numpy.core.fromnumeric import argmax
result = rs[argmax(num)]
```

In [50]:
```
result
```

Out[50]:
```
0
```

In [51]:
```
model.save('mnist.h5')
```

In [52]:
```
model = load_model('mnist.h5')
```

In [53]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout_1 (Dropout)         (None, 1600)              0
```

```
 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

Conclusion

Thus I have learn basics of deep learning and Successfully implement CNN

In [ ]: