S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 6

Aim: Implement Convolution Neural Network for Machine Learning.

Name of Student	: ______________________
Roll No.	: ______________________
Semester/Year	: ______________________
Academic Session	: ______________________
Date of Performance	: ______________________
Date of Submission	: ______________________

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

It is computationally ineffective right. So here comes Convolutional Neural Network or CNN. In simple word what CNN does is, it extract the feature of image and convert it into lower dimension without loosing its characteristics. In the following example you can see that initial the size of the image is 224 x 224 x

THEORY:

An introductory look at Convolutional Neural Network with theory and code example. I want to write about one of the most important neural networks used in the field of deep learning, especially for image recognition and natural language processing: convolutional neural network, also called "CNN" or "ConvNet".

What is Convolutional neural network?

Convolutional neural networks. Sounds like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. 2012 was the first year that neural nets grew to prominence as Alex Krizhevsky used them to win that year's ImageNet competition (basically, the annual Olympics of computer vision), dropping the classification error record from 26% to 15%, an astounding improvement at the time. Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural nets for their automatic tagging algorithms, Google for their photo search, Amazon for their product recommendations, Pinterest for their home feed personalization, and Instagram for their search infrastructure.

Architecture of a Traditional CNN:

A convolutional neural network is composed of at least 3 layers:

- A **convolution layer** to perform convolution operations and to generate many feature maps from one image;
- A **pooling layer** to denoise the feature maps by shrinking non-overlapping submatrices into summary statistics (such as maximums);
- A **dense layer** which is a usual (shallow/deep) neural network that takes flattened inputs.

Working of Convolutional Neural Networks:

Convolutional neural networks are based on neuroscience findings. They are made of layers of artificial neurons called nodes. These nodes are functions that calculate the weighted sum of the inputs and return an activation map. This is the convolution part of the neural network.

Each node in a layer is defined by its weight values. When you give a layer some data, like an image, it takes the pixel values and picks out some of the visual features.

When you're working with data in a CNN, each layer returns activation maps. These maps point out important features in the data set. If you gave the CNN an image, it'll point out features based on pixel values, like colors, and give you an activation function.

Usually with images, a CNN will initially find the edges of the picture. Then this slight definition of the image will get passed to the next layer. Then that layer will start detecting things like corners and color groups. Then that image definition will get passed to the next layer and the cycle continues until a prediction is made.

As the layers get more defined, this is called max pooling. It only returns the most relevant features from the layer in the activation map. This is what gets passed to each successive layer until you get the final layer.

The last layer of a CNN is the classification layer which determines the predicted value based on the activation map. If you pass a handwriting sample to a CNN, the classification layer will tell you what letter is in the image. This is what autonomous vehicles use to determine whether an object is another car, a person, or some other obstacle.

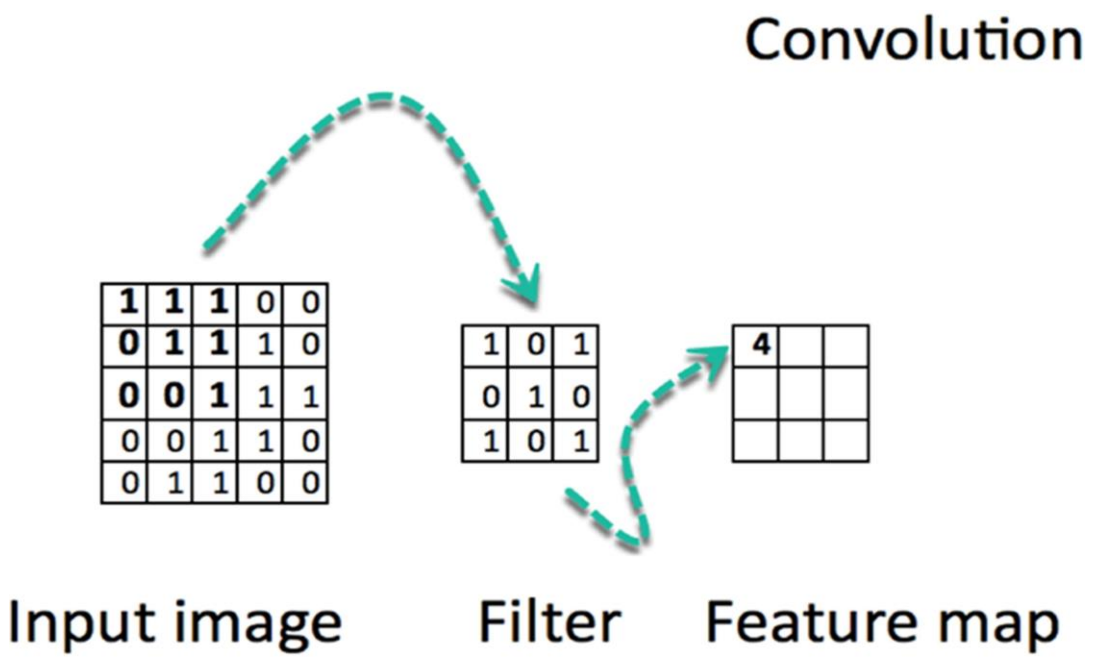
Training a CNN is similar to training many other machine learning algorithms. You'll start with some training data that is separate from your test data and you'll tune your weights based on the accuracy of the predicted values. Just be careful that you don't overfit your model.

Different types of CNN

1D CNN: With these, the CNN kernel moves in one direction. 1D CNNs are usually used on time-series data.

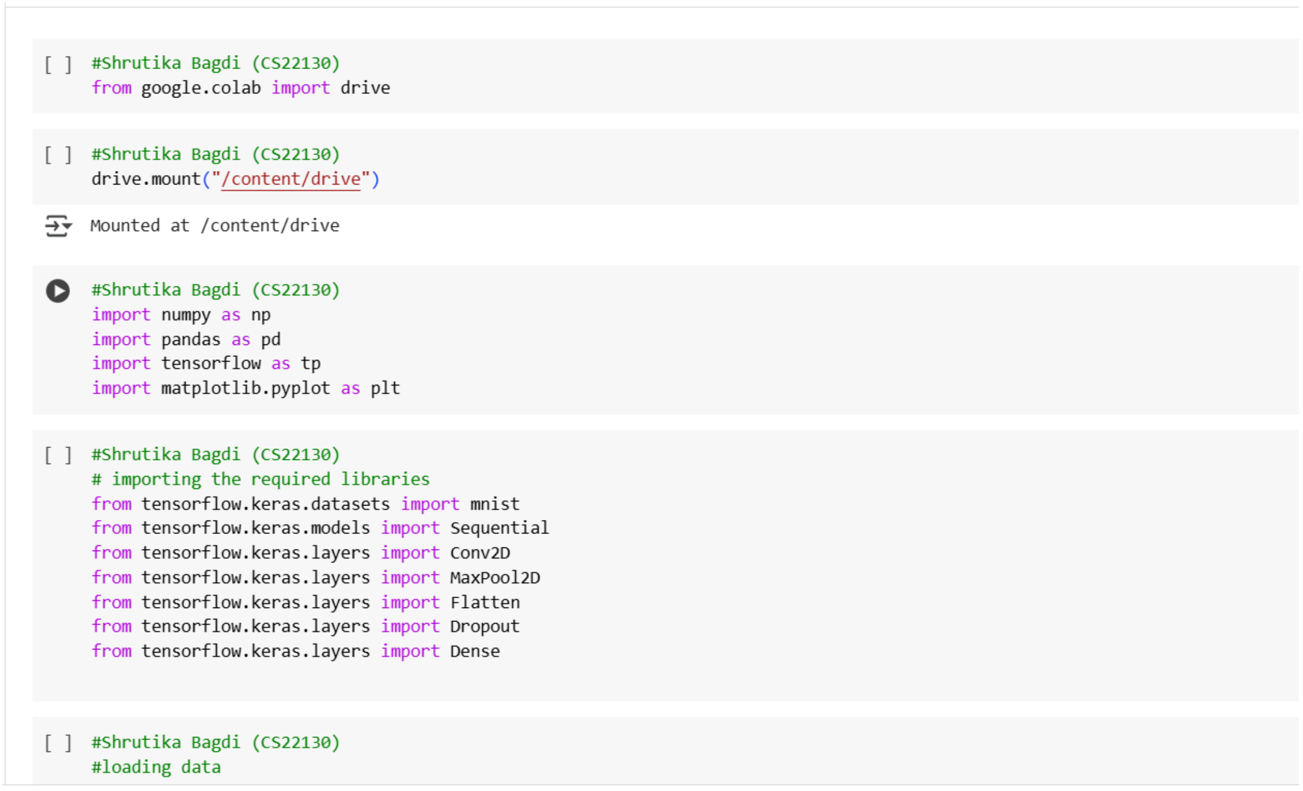
2D CNN: These kinds of CNN kernels move in two directions. You'll see these used with image labelling and processing.

3D CNN: This kind of CNN has a kernel that moves in three directions. With this type of CNN, researchers use them on 3D images like CT scans and MRIs.

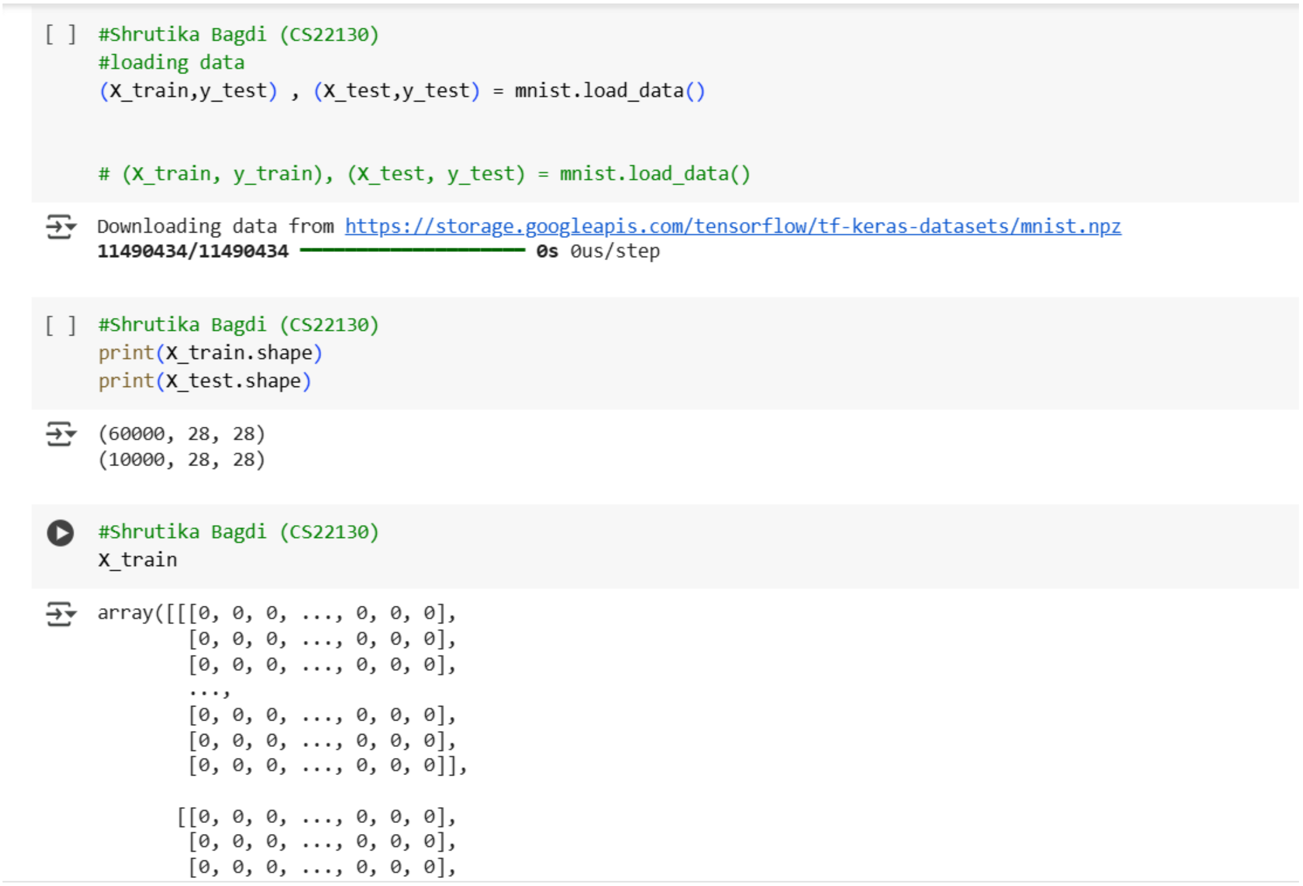


PROGRAM CODE:

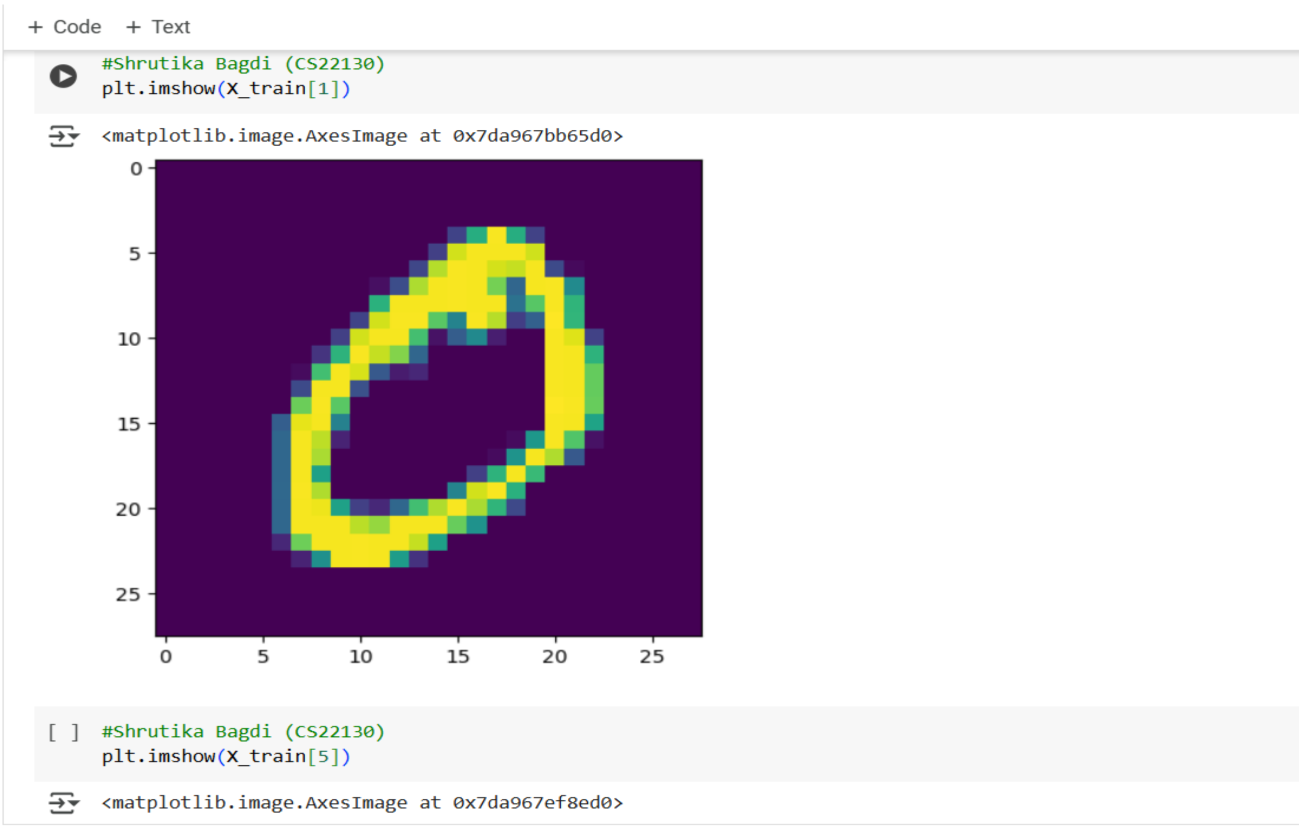
OUTPUT (SCREENSHOT):



```
[ ] #Shrutika Bagdi (CS22130)
    from google.colab import drive

[ ] #Shrutika Bagdi (CS22130)
    drive.mount("/content/drive")

    Mounted at /content/drive

    #Shrutika Bagdi (CS22130)
    import numpy as np
    import pandas as pd
    import tensorflow as tp
    import matplotlib.pyplot as plt

[ ] #Shrutika Bagdi (CS22130)
    # importing the required libraries
    from tensorflow.keras.datasets import mnist
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D
    from tensorflow.keras.layers import MaxPool2D
    from tensorflow.keras.layers import Flatten
    from tensorflow.keras.layers import Dropout
    from tensorflow.keras.layers import Dense

[ ] #Shrutika Bagdi (CS22130)
    #loading data
```

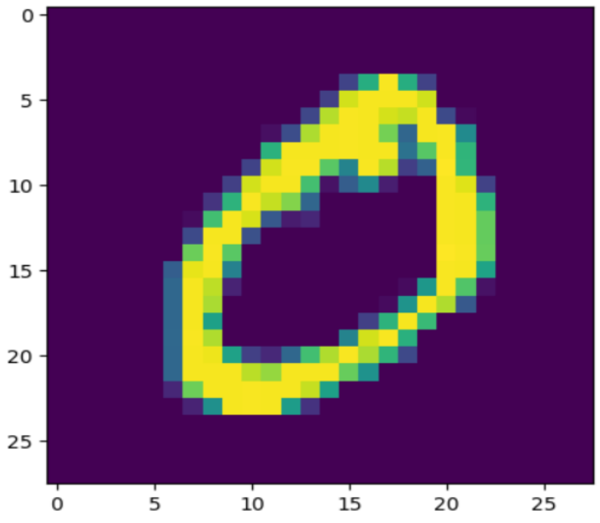


```
[ ] #Shrutika Bagdi (CS22130)
    #loading data
    (X_train,y_test) , (X_test,y_test) = mnist.load_data()


    # (X_train, y_train), (X_test, y_test) = mnist.load_data()

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

[ ] #Shrutika Bagdi (CS22130)
    print(X_train.shape)
    print(X_test.shape)

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

    #Shrutika Bagdi (CS22130)
    X_train

    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],
```



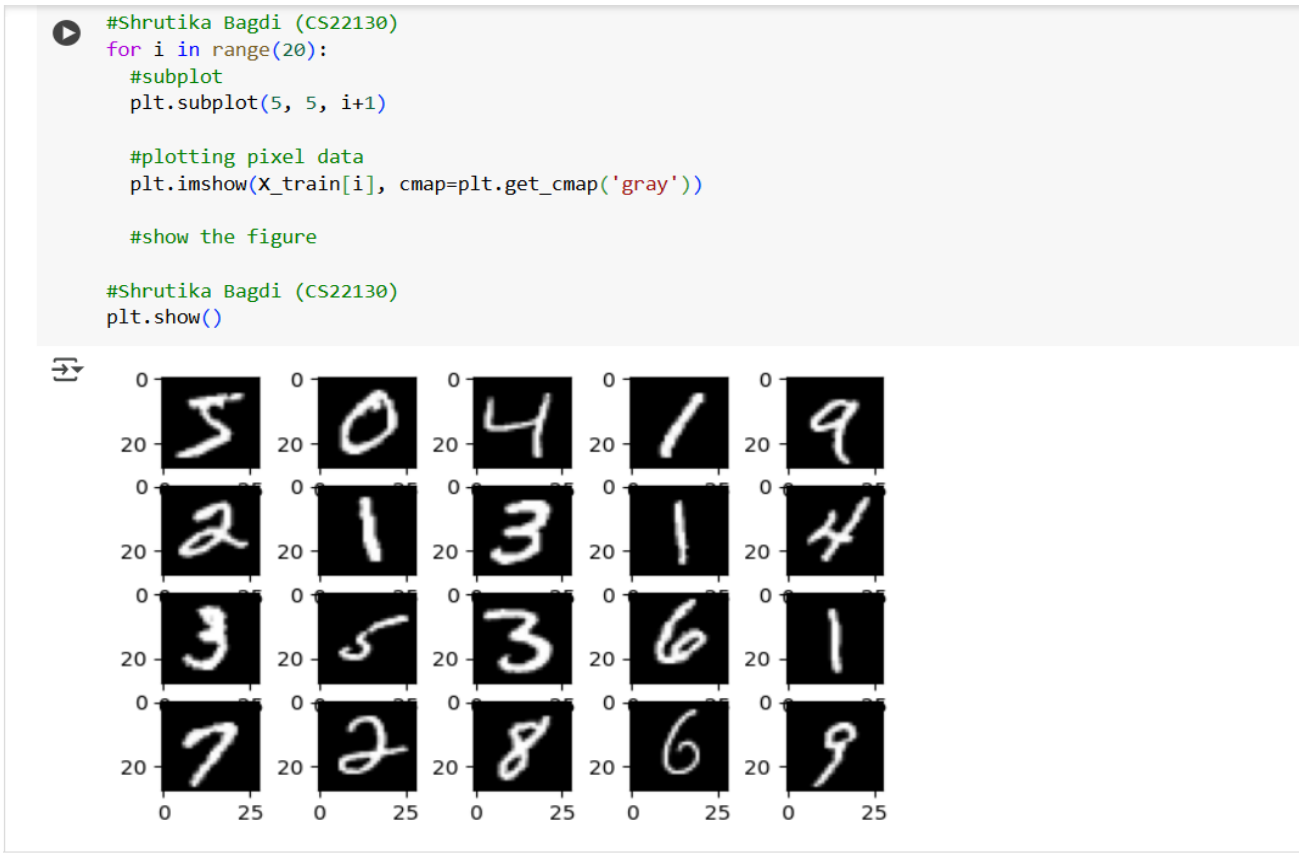
```
#Shrutika Bagdi (CS22130)
plt.imshow(X_train[1])
```

```
<matplotlib.image.AxesImage at 0x7da967bb65d0>
```

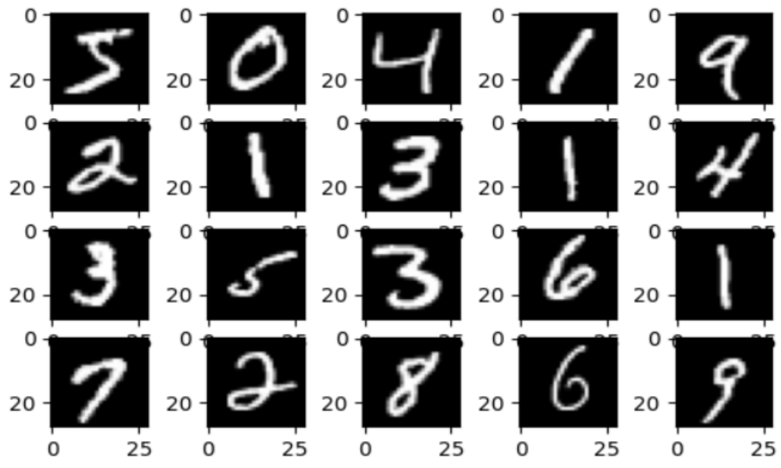


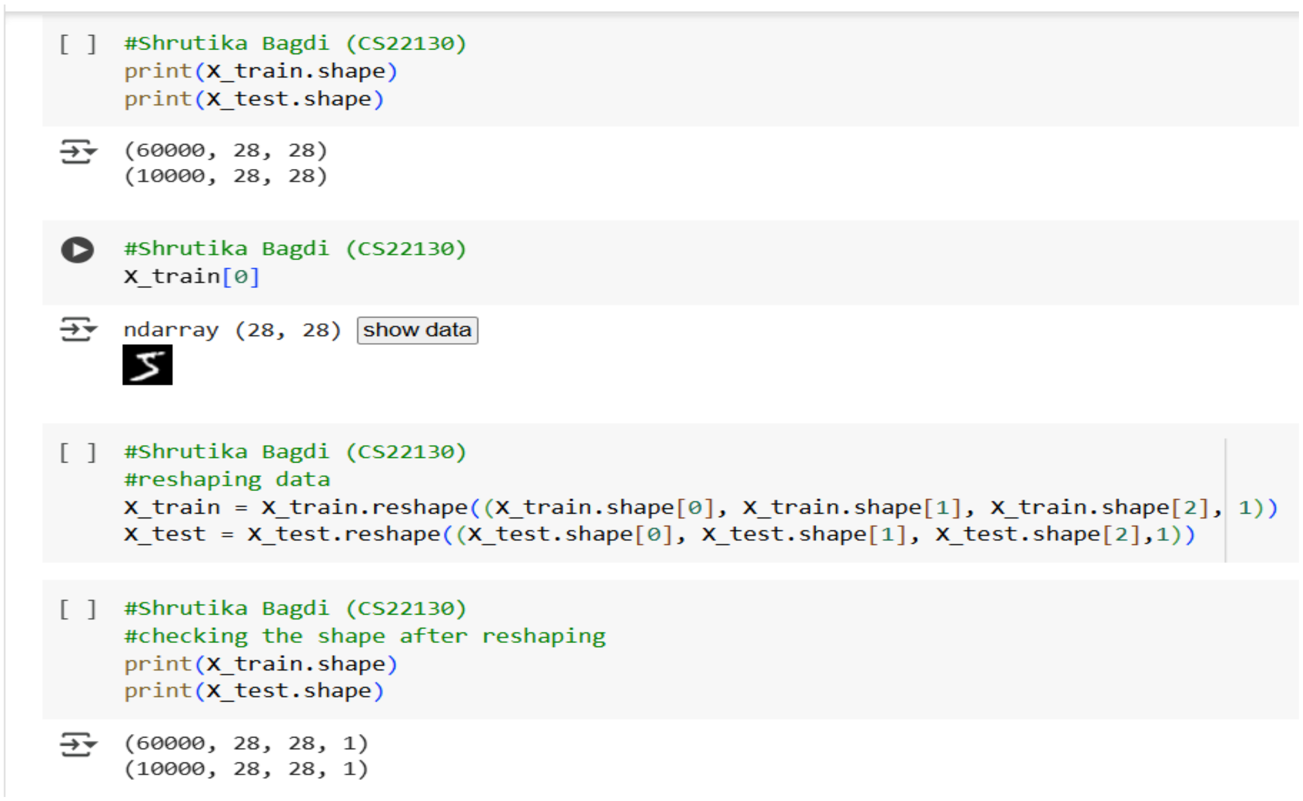
```
#Shrutika Bagdi (CS22130)
plt.imshow(X_train[5])
```

```
<matplotlib.image.AxesImage at 0x7da967ef8ed0>
```



```
#Shrutika Bagdi (CS22130)
for i in range(20):
  #subplot
  plt.subplot(5, 5, i+1)

  #plotting pixel data
  plt.imshow(X_train[i], cmap=plt.get_cmap('gray'))

  #show the figure

#Shrutika Bagdi (CS22130)
plt.show()
```



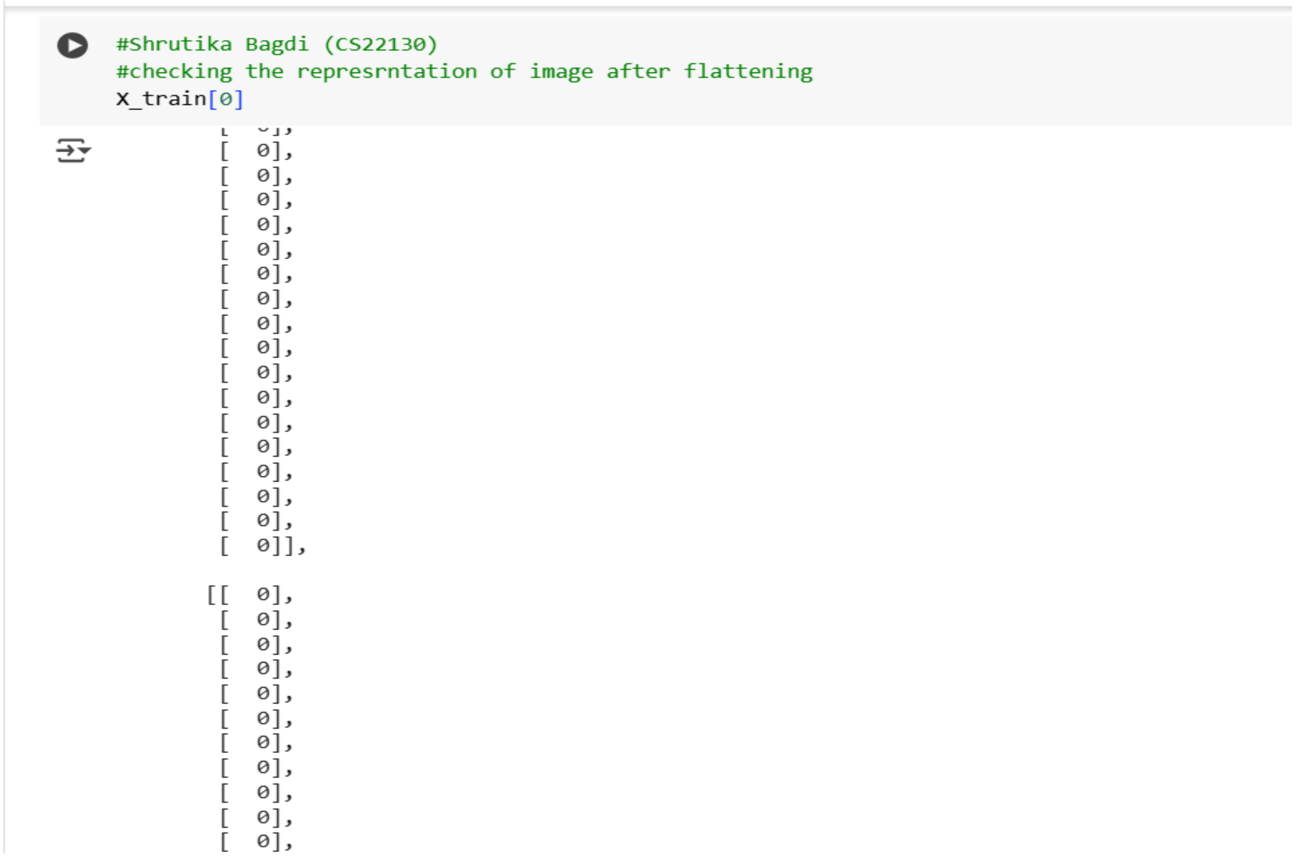


```
[ ] #Shrutika Bagdi (CS22130)
    print(X_train.shape)
    print(X_test.shape)

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

```
#Shrutika Bagdi (CS22130)
X_train[0]

ndarray (28, 28) show data
```

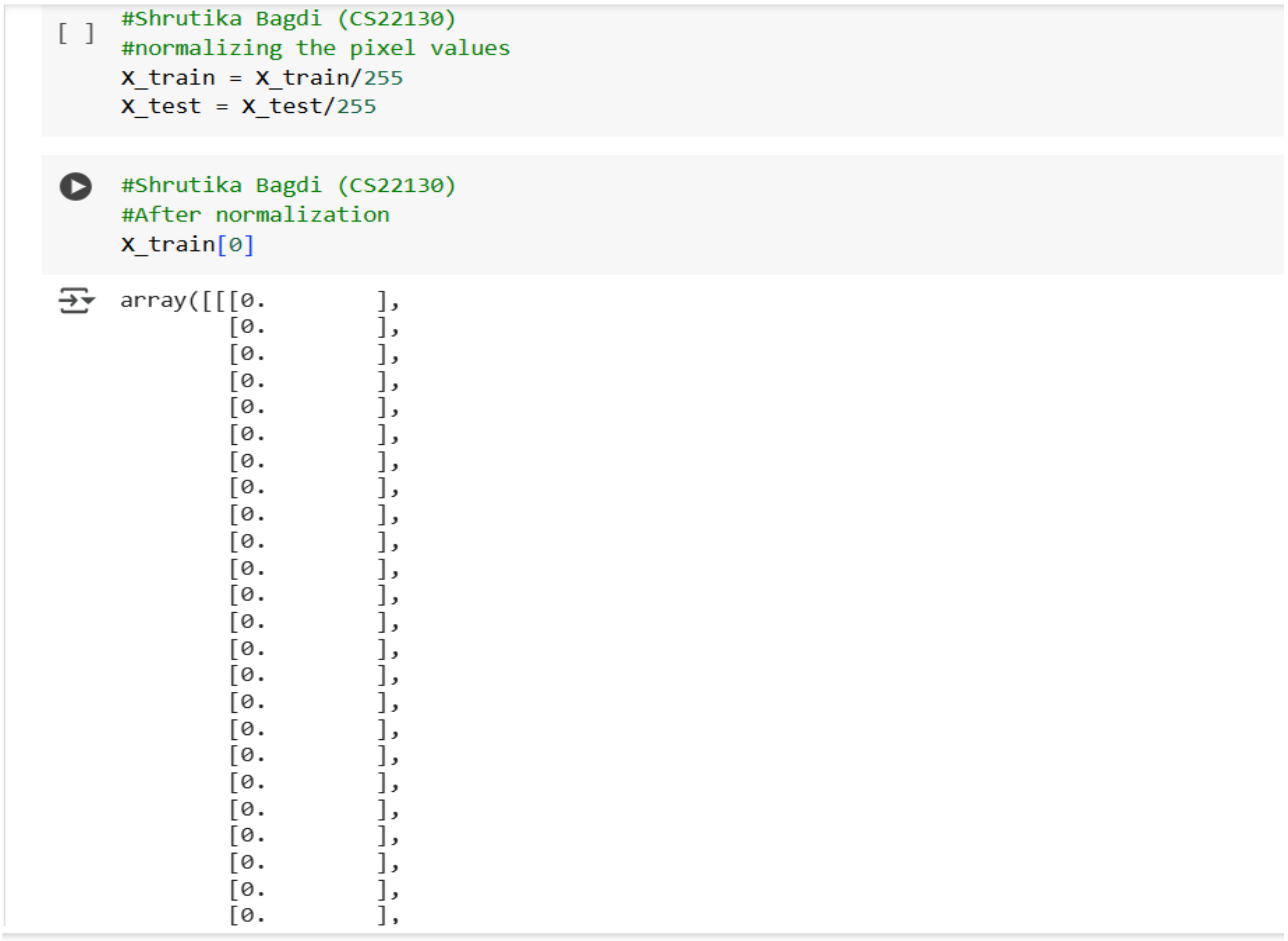
```
[ ] #Shrutika Bagdi (CS22130)
    #reshaping data
    X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], X_train.shape[2], 1))
    X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], X_test.shape[2],1))
```

```
[ ] #Shrutika Bagdi (CS22130)
    #checking the shape after reshaping
    print(X_train.shape)
    print(X_test.shape)

    (60000, 28, 28, 1)
    (10000, 28, 28, 1)
```



```
#Shrutika Bagdi (CS22130)
#checking the represrntation of image after flattening
X_train[0]
```

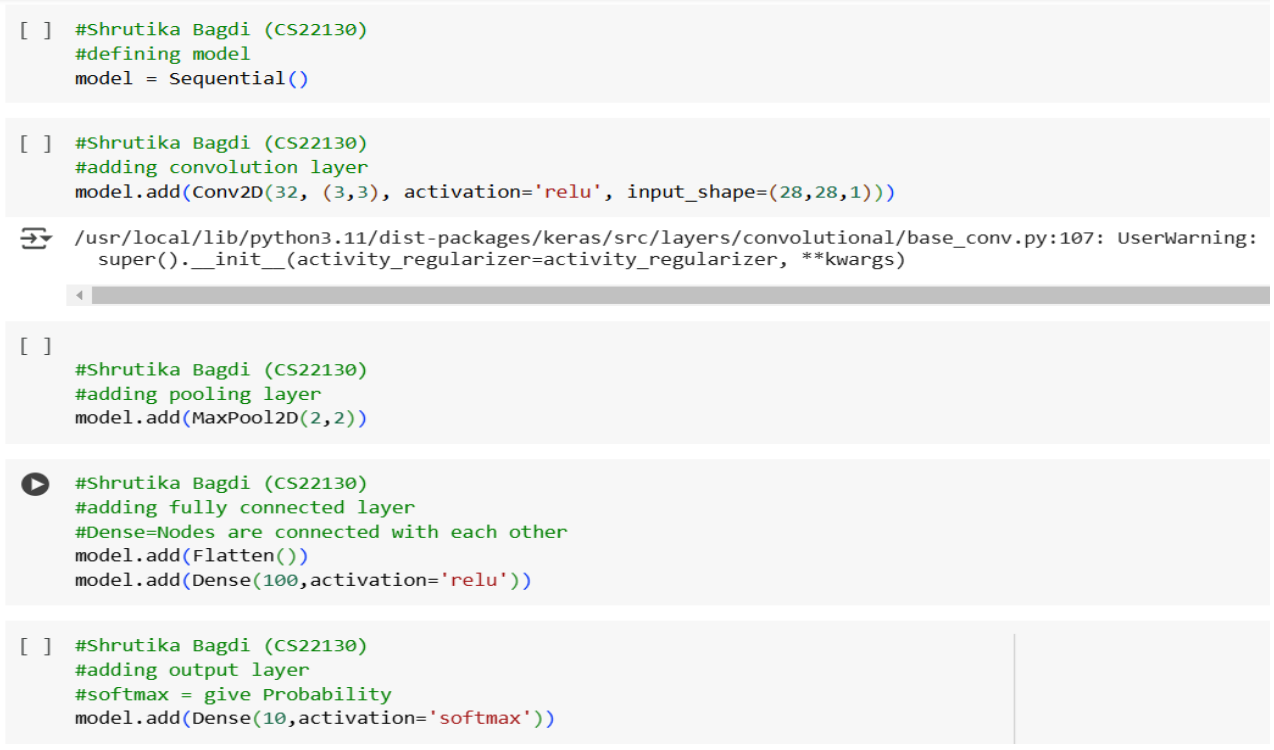
```
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0]],

       [[  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
        [  0],
```



```
[ ] #Shrutika Bagdi (CS22130)
    #normalizing the pixel values
    X_train = X_train/255
    X_test = X_test/255
```

```
#Shrutika Bagdi (CS22130)
#After normalization
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.        ],
```



```
[ ] #Shrutika Bagdi (CS22130)
    #defining model
    model = Sequential()
```

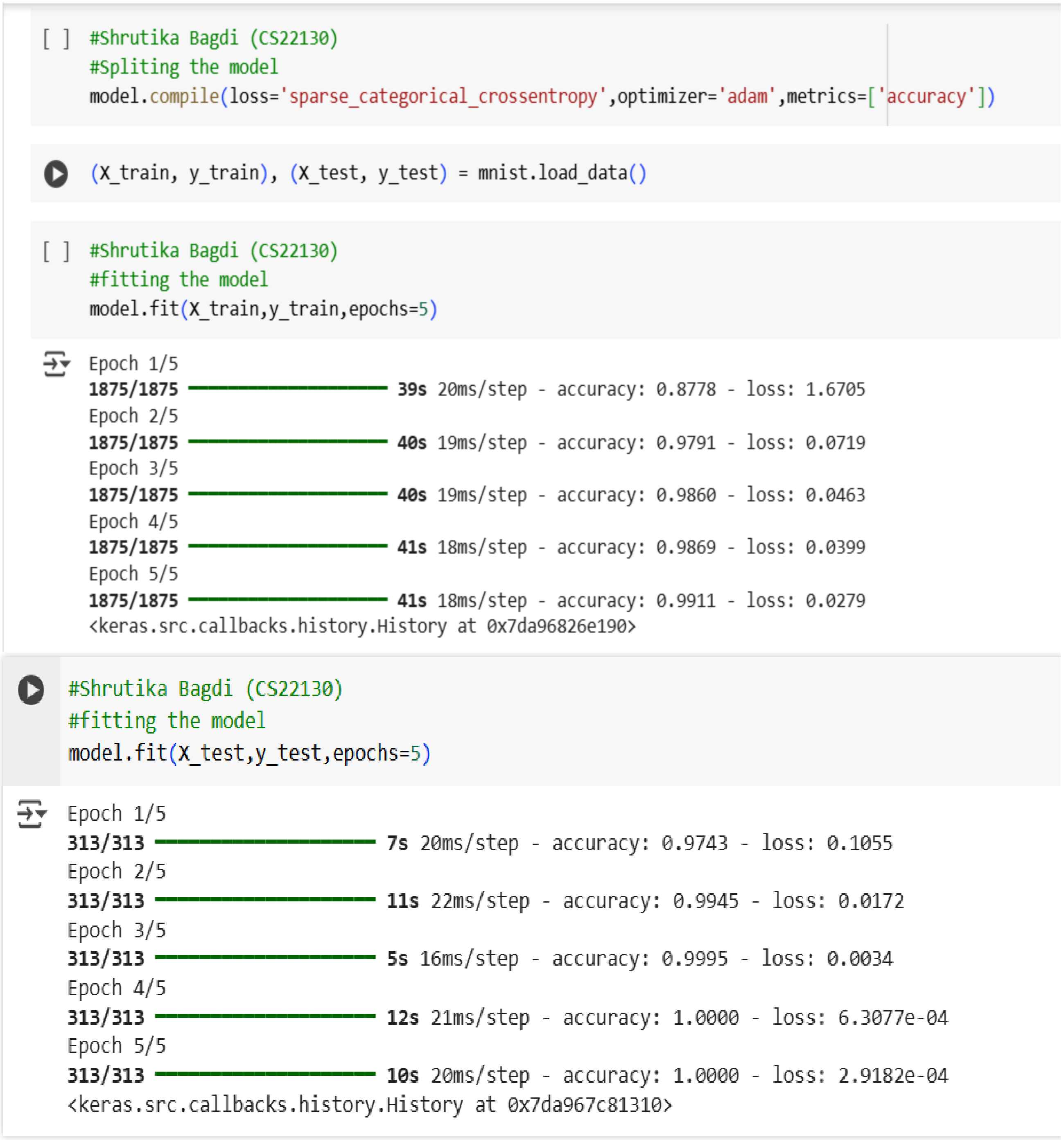
```
[ ] #Shrutika Bagdi (CS22130)
    #adding convolution layer
    model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
[ ]
    #Shrutika Bagdi (CS22130)
    #adding pooling layer
    model.add(MaxPool2D(2,2))
```

```
#Shrutika Bagdi (CS22130)
#adding fully connected layer
#Dense=Nodes are connected with each other
model.add(Flatten())
model.add(Dense(100,activation='relu'))
```

```
[ ] #Shrutika Bagdi (CS22130)
    #adding output layer
    #softmax = give Probability
    model.add(Dense(10,activation='softmax'))
```



```
[ ] #Shrutika Bagdi (CS22130)
    #Spliting the model
    model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

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

```
[ ] #Shrutika Bagdi (CS22130)
    #fitting the model
    model.fit(X_train,y_train,epochs=5)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 39s 20ms/step - accuracy: 0.8778 - loss: 1.6705
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 40s 19ms/step - accuracy: 0.9791 - loss: 0.0719
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 40s 19ms/step - accuracy: 0.9860 - loss: 0.0463
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 18ms/step - accuracy: 0.9869 - loss: 0.0399
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 41s 18ms/step - accuracy: 0.9911 - loss: 0.0279
<keras.src.callbacks.history.History at 0x7da96826e190>
```

```
#Shrutika Bagdi (CS22130)
#fitting the model
model.fit(X_test,y_test,epochs=5)
```

```
Epoch 1/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 7s 20ms/step - accuracy: 0.9743 - loss: 0.1055
Epoch 2/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 11s 22ms/step - accuracy: 0.9945 - loss: 0.0172
Epoch 3/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - accuracy: 0.9995 - loss: 0.0034
Epoch 4/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 12s 21ms/step - accuracy: 1.0000 - loss: 6.3077e-04
Epoch 5/5
313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 20ms/step - accuracy: 1.0000 - loss: 2.9182e-04
<keras.src.callbacks.history.History at 0x7da967c81310>
```

CONCLUSION:

_ _

_ _

_ _

DISCUSSION AND VIVA VOCE:

Q1 What is CNN?

Q2: What are the layers of CNN ?

Q3 What is padding in CNN

Q4: How can CNN improve image classification and image recognition**?**

Q5: Explain the different layers of CNN.

REFERENCE

- https://towardsdatascience.com/convolutional-neural-networks-cnns-a-practical-perspective-c7b3b2091aa8
- https://en.wikipedia.org/wiki/Convolutional_neural_network#References