e-ISSN: 2582-5208

International Research Journal of Modernization in Engineering Technology and Science

(Peer-Reviewed, Open Access, Fully Refereed International Journal)

Volume:04/Issue:03/March-2022 Impact Factor- 6.752 www.irjmets.com

IMAGE CLASSIFICATION OF CIFAR10 USING CNN

V Seetharamarao[*1], P V S N Deepak[*2], N Srihari[*3], P Sai Kumar[*4]

[*1]Assistant Professor, ECE Department, SNIST, Hyderabad, India.

[*2,3,4]UG Scholar, ECE Department, SNIST, Hyderabad, India.

ABSTRACT

Image classification is a very important issue in digital image analysis. CNN is a type of Deep Neural Networks (DNN) that contains multiple layers such as Conv layers, integration layer, and fully integrated layer. Convolutional neural networks (CNNs) are widely used in pattern- and picture- recognition problems as they have a number of advantages compared to others strategies. CIFAR-10 is a very popular computer vision database. This database is well read in it many types of in-depth study of object recognition. This database contains 60,000 images separated by 10 target classes, each a section containing 6000 images of 32 * 32 shapes. This database contains images of low-resolution (32 * 32), which allows researchers to experiment with new algorithms. The different categories of this database are:

1. Flight 2. Car 3. Bird 4. Ikati 5. Deer 6. Dog 7. Frog 8. Horse 9. The ship 10.Iloli

The CIFAR-10 database is already available in the Keras data sets module. We do not know you need to download it; we can import it directly from keras datasets.

Keywords: CNN, Deep Learning, CIFAR10, Tensorflow.

I. INTRODUCTION

The image classification is the ability to capture a picture and categorise it successfully in one of the other classes. The goal of our work will be to build a model will be able to identify and classify images accurately. This project uses Convolutional Neural Networks to train effectively model. The database used is the Cifar10 dataset. The Cifar10 database imported from cameras has 10 classes and we have built a model so much so that our model predicts a given image with excellent external accuracy any problems with excessive dressing or inadequacy.

Scope and Proposed Model

This provides a straightforward way to identify different images from the database. The proposed system has been improved by the accuracy of the training and checking accuracy without overfilling. The proposed program is well done with Adam optimizer for better results. Images can be predicted in a database using the Cnn algorithm in a great relief. Improves image quality separation. The proposed system can be easily changed to different fields, science and technology.

II. LITERATURE SURVEY

Tensorflow

TensorFlow is a machine learning software library that is free and open-source. It can be used for a variety of applications, but it focuses on deep neural network training and inference. Tensorflow is a dataflow and differentiable programming-based symbolic math toolkit. At Google, it's used for both research and manufacturing. The Google Brain team created TensorFlow for internal Google use. In 2015, it was distributed under the Apache License 2.0. Google Brain's second-generation system is called TensorFlow. On February 11, 2017, version 1.0.0 was released. TensorFlow can run on many CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units) while the reference implementation runs on single devices. TensorFlow is available for 64-bit Linux, macOS, Windows, and Android and iOS mobile computing platforms.

Keras

Keras is a Python-based deep learning API that runs on top of TensorFlow. It was created with the goal of allowing users to explore quickly. To perform successful research, you must be able to get from concept to outcome as quickly as feasible. Keras includes a number of implementations of basic neural-network building elements including layers, goals, activation functions, optimizers, and a number of other tools to make dealing with picture and text data easier and to reduce the amount of coding required to write deep neural network code. The source is hosted on GitHub, and there is a Slack channel for community support. Keras offers support for convolutional and recurrent neural networks in addition to regular neural networks. Other popular utility layers supported include dropout, batch normalisation, and pooling.

Convolutional Neural Networks (CNN)

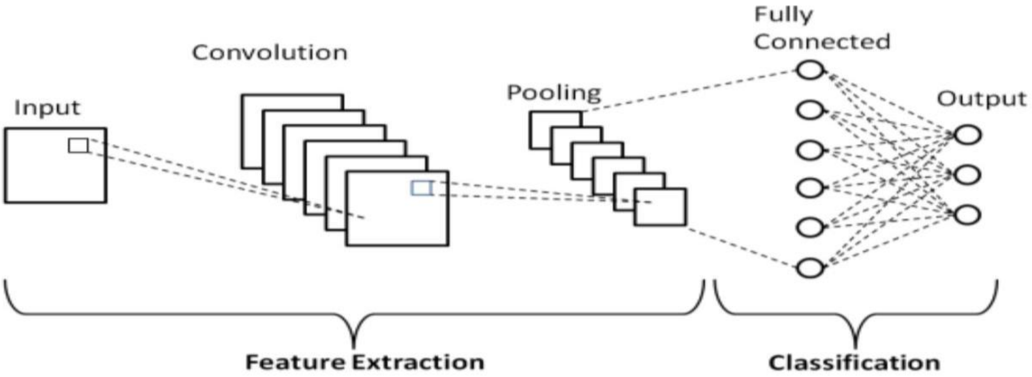
Deep Learning has proven to be a particularly useful technique in recent decades due to its capacity to manage massive volumes of data. Hidden layers have eclipsed traditional techniques in popularity, particularly in pattern recognition. Convolutional Neural Networks are one of the most widely used deep neural networks. Researchers have sought to create a system since the 1950s, when AI was in its infancy. that is capable of comprehending visual data This field grew in popularity over the years. Computer Vision is the term for this type of technology. When a breakthrough in computer vision was made in 2012, it was a quantum leap. A group of University of Toronto researchers created an AI model that by a significant margin, it outperformed the best picture recognition systems. AlexNet (named after its developer) was the name given to the AI system. Alex Krizhevsky took first place in the 2012 ImageNet computer vision competition. 85 percent accuracy is incredible. The runner-up received a respectable 74 percent on the test. Convolutional Neural Networks, an unique sort of neural network, were at the heart of AlexNet. A neural network that tries to replicate human eyesight as closely as possible. CNNs have evolved over time. Many Computer Vision applications now include it as a critical component.

Working of CNN Algorithm

The Convolutional Neural Networks contains various layers included in it and the input image has to pass through these layers to have a desired output.

The layers included are:

1.Convolutional layer 2.Pooling layer 3.Fully connected layer

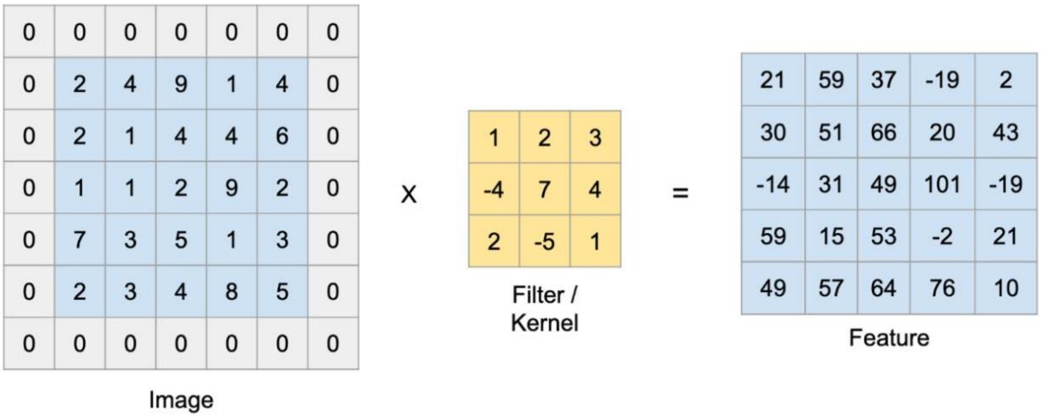


1. The Convolutional layer:

It is the place where the convolution operation takes place.

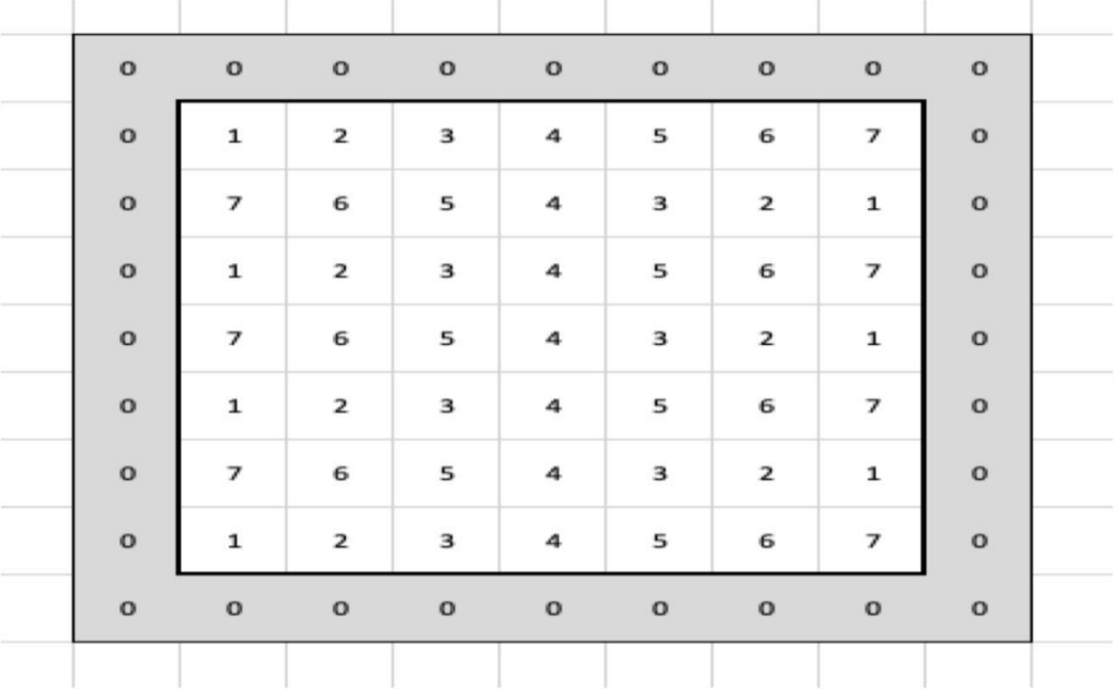
Filter / kernel : A matrix which contains weights , which are learn through back propagation , used to detect the edges and features from a given input image.

Feature Map: output generated when a filter is convolved over input image.



$$\underset{\text{Image}}{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 4 & 9 & 1 & 4 & 0 \\ 0 & 2 & 1 & 4 & 4 & 6 & 0 \\ 0 & 1 & 1 & 2 & 9 & 2 & 0 \\ 0 & 7 & 3 & 5 & 1 & 3 & 0 \\ 0 & 2 & 3 & 4 & 8 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}} \; X \; \underset{\text{Filter / Kernel}}{\begin{bmatrix} 1 & 2 & 3 \\ -4 & 7 & 4 \\ 2 & -5 & 1 \end{bmatrix}} = \underset{\text{Feature}}{\begin{bmatrix} 21 & 59 & 37 & -19 & 2 \\ 30 & 51 & 66 & 20 & 43 \\ -14 & 31 & 49 & 101 & -19 \\ 59 & 15 & 53 & -2 & 21 \\ 49 & 57 & 64 & 76 & 10 \end{bmatrix}}$$

Convolution operation: This technique involves the process of convolving a given filter over an input image to attain feature map. This is what happens in the convolutional layer.



0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	0
0	7	6	5	4	3	2	1	0
0	1	2	3	4	5	6	7	0
0	7	6	5	4	3	2	1	0
0	1	2	3	4	5	6	7	0
0	7	6	5	4	3	2	1	0
0	1	2	3	4	5	6	7	0
0	0	0	0	0	0	0	0	0

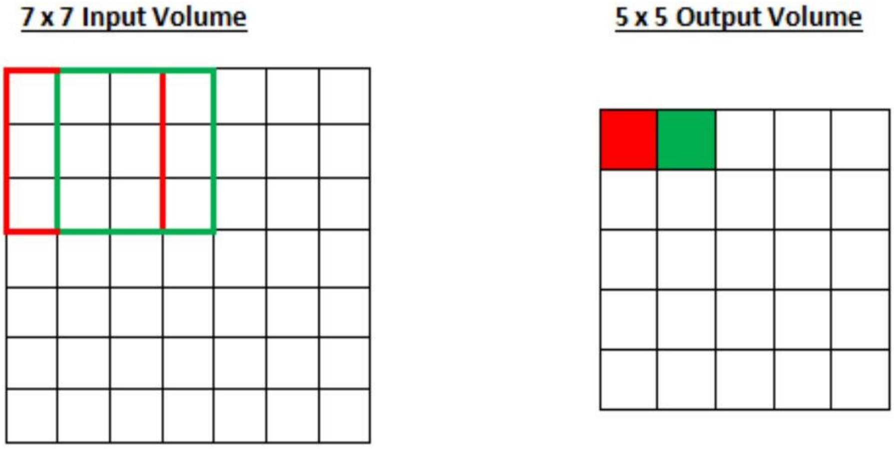
Padding: Adding an additional layer around the perimeter of our original picture to keep its features without being shrunk, we pad with zeros by convention. It's a hyperparameter that has to be fine-tuned.

The following are some of the reasons why we use padding:

1. Because the convolution procedure is repeated, our input picture will be shrunk, and its original size will be lost until it reaches the final step.

2. When a filter is applied to an input image, the top left corner appears just once, whereas pixels in the centre appear numerous times, resulting in information being thrown away from the corners.

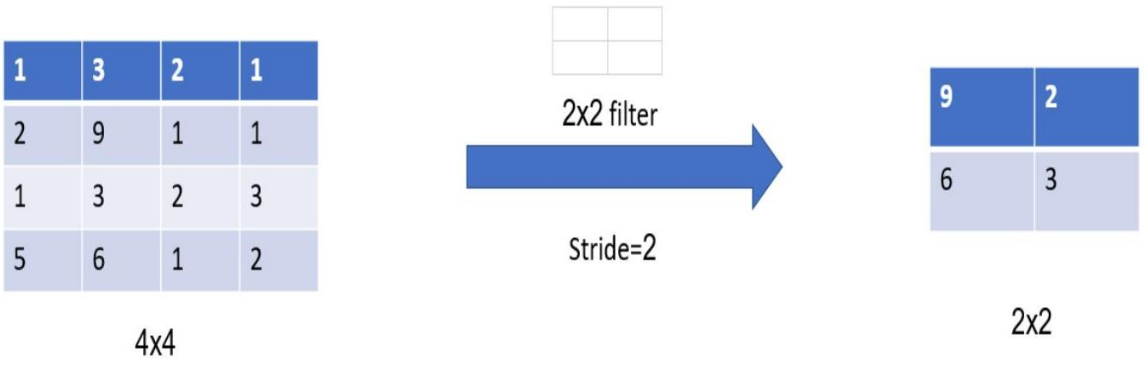
Stride: The amount by which the kernel moves as it passes across the picture is referred to as the stride. Stride refers to the step of the convolution operation and is also a hyperparameter in this context.



2. The Pooling Layer :

ConvNets frequently include pooling layers in addition to convolutional layers to make the depiction smaller , to make the computation go faster, also to make some of the detection characteristics a little more robust. There are several varieties of pooling, but two of them are the most well-known: **Maximum pooling** and **average pooling** are two different types of pooling.

Example for max pooling:



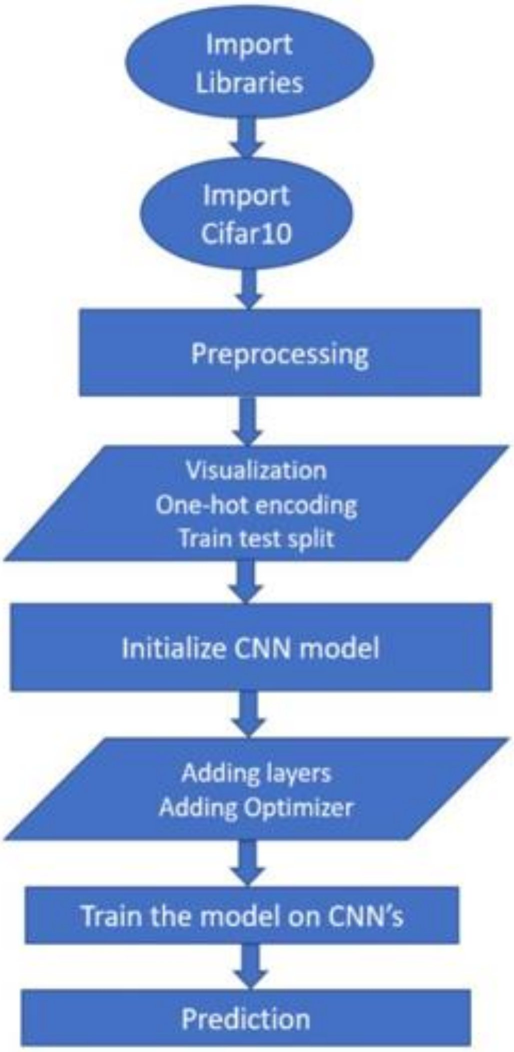
1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

9	2
6	3

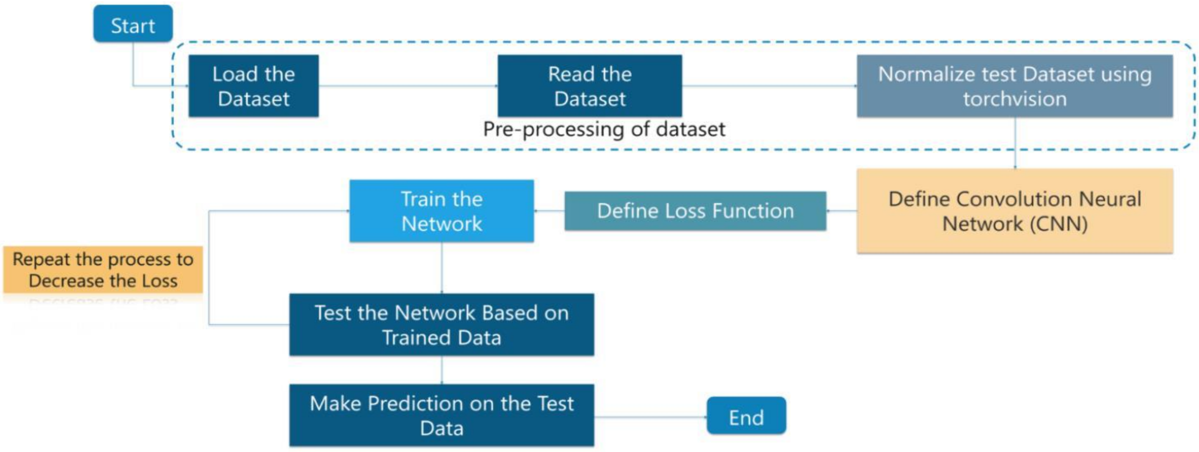
3. Fully Connected Layer/ Dense Layer :

The dense layer is a fully connected layer that feeds the outputs of the convolutional layers via one or more neural layers to create a prediction. In the case of a completely linked layer, all of the elements from all of the preceding layer's features are included in the computation of each element of each output feature, which also contains an activation function depending on our business needs. There is a 'Flatten' layer between the convolutional and fully linked layers. A two-dimensional matrix of characteristics is flattened into a vector that may be input into a fully connected neural network classifier.

III. SYSTEM ARCHITECTURE

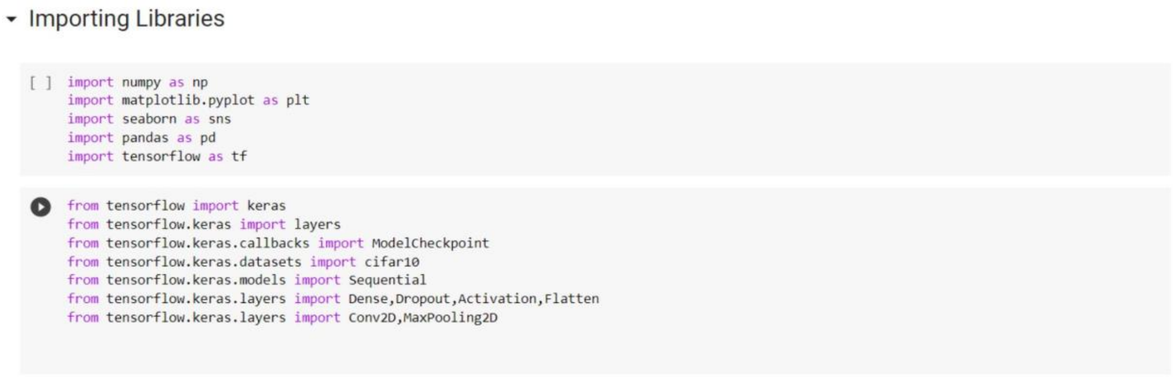


Data Flow Diagram:


Start
Load the Dataset
Read the Dataset
Normalize test Dataset using torchvision
Pre-processing of dataset
Define Convolution Neural Network (CNN)
Define Loss Function
Train the Network
Repeat the process to Decrease the Loss
Test the Network Based on Trained Data
Make Prediction on the Test Data
End

IV. IMPLEMENTATION OF MODEL

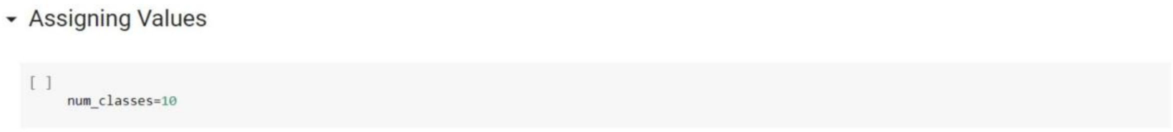
1. Importing Libraries:



Importing Libraries

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    import pandas as pd
    import tensorflow as tf
```

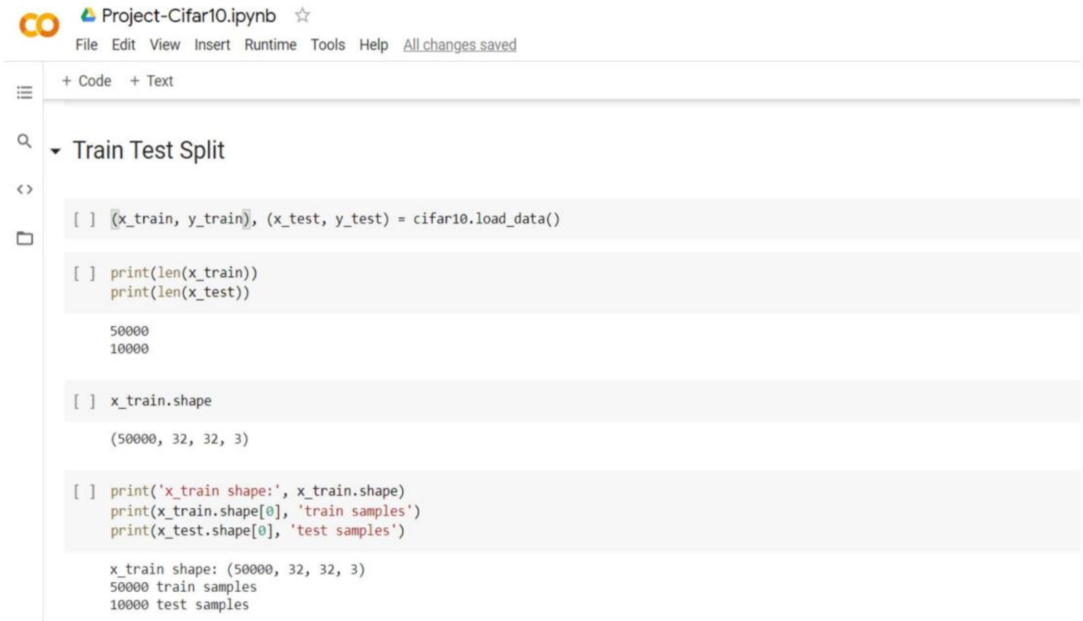
```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Activation,Flatten
from tensorflow.keras.layers import Conv2D,MaxPooling2D
```



Assigning Values

```
[ ]
    num_classes=10
```

2. Load Data and Split:



Train Test Split

```
[ ] (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
[ ] print(len(x_train))
    print(len(x_test))

    50000
    10000
```

```
[ ] x_train.shape

    (50000, 32, 32, 3)
```

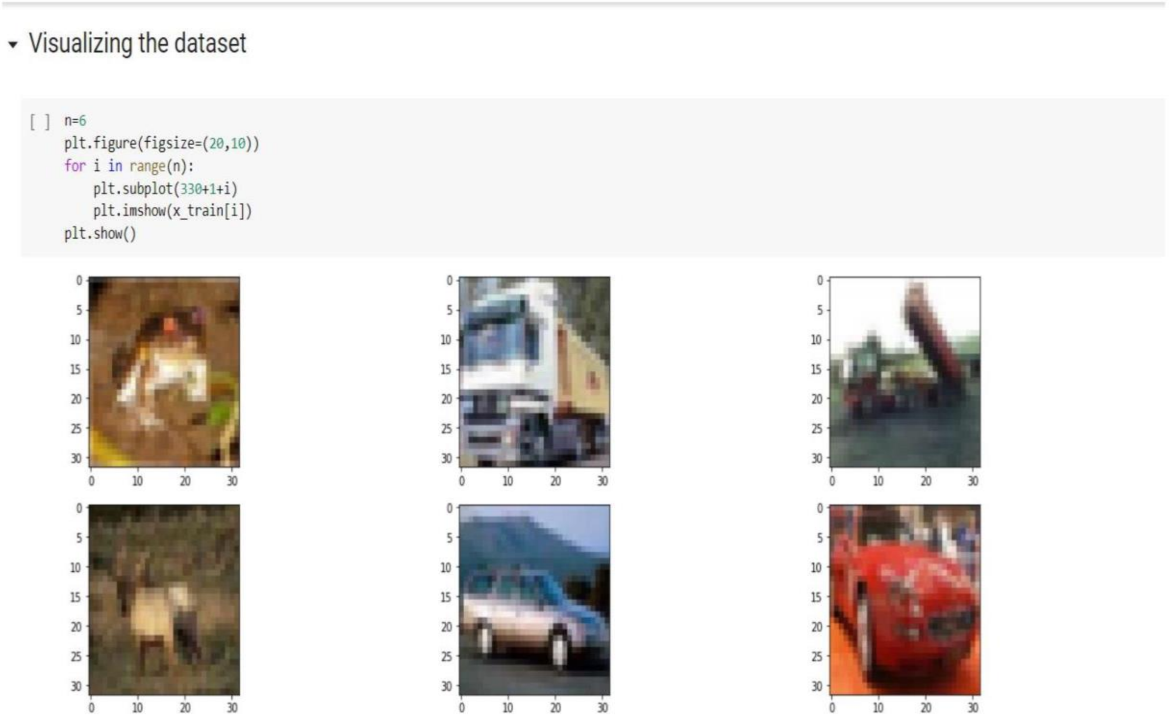
```
[ ] print('x_train shape:', x_train.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    x_train shape: (50000, 32, 32, 3)
    50000 train samples
    10000 test samples
```

We'll input the cifar10 data into our model and divide it into train and test sets in this stage. Because we have a total of 60000 photographs, the train set now has 50000 images and the test set has 10,000 images. Because the photos in the collection are coloured, their dimensions are (32,32,3).

3. Visualising the data:

Here we look at the images present in our data in a random manner to get the idea of our dataset. We can take any number of images that we intend to look at.

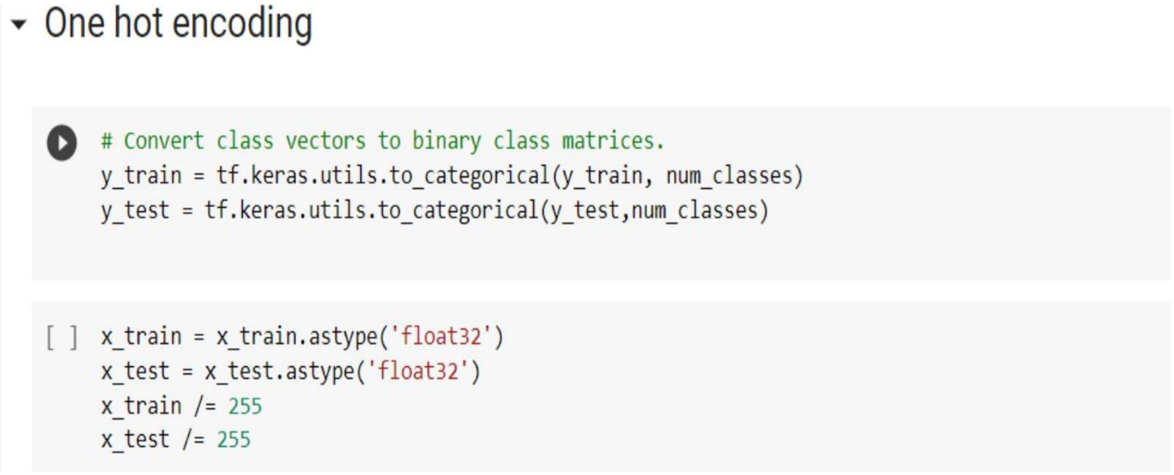


Visualizing the dataset

```
[ ] n=6
    plt.figure(figsize=(20,10))
    for i in range(n):
        plt.subplot(330+1+i)
        plt.imshow(x_train[i])
    plt.show()
```

4. One Hot Encoding:

One approach of transforming data to prepare it for an algorithm and improve prediction is hot encoding. With one-hot, each category value is converted into a new categorical column and given a binary value of 1 or 0. A binary vector is used to represent each integer value.

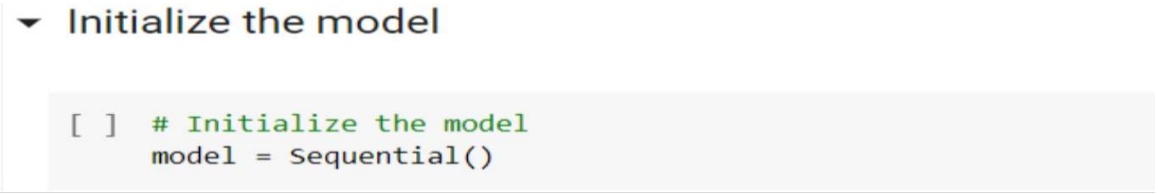


One hot encoding

```
# Convert class vectors to binary class matrices.
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test,num_classes)
```

```
[ ] x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255
```

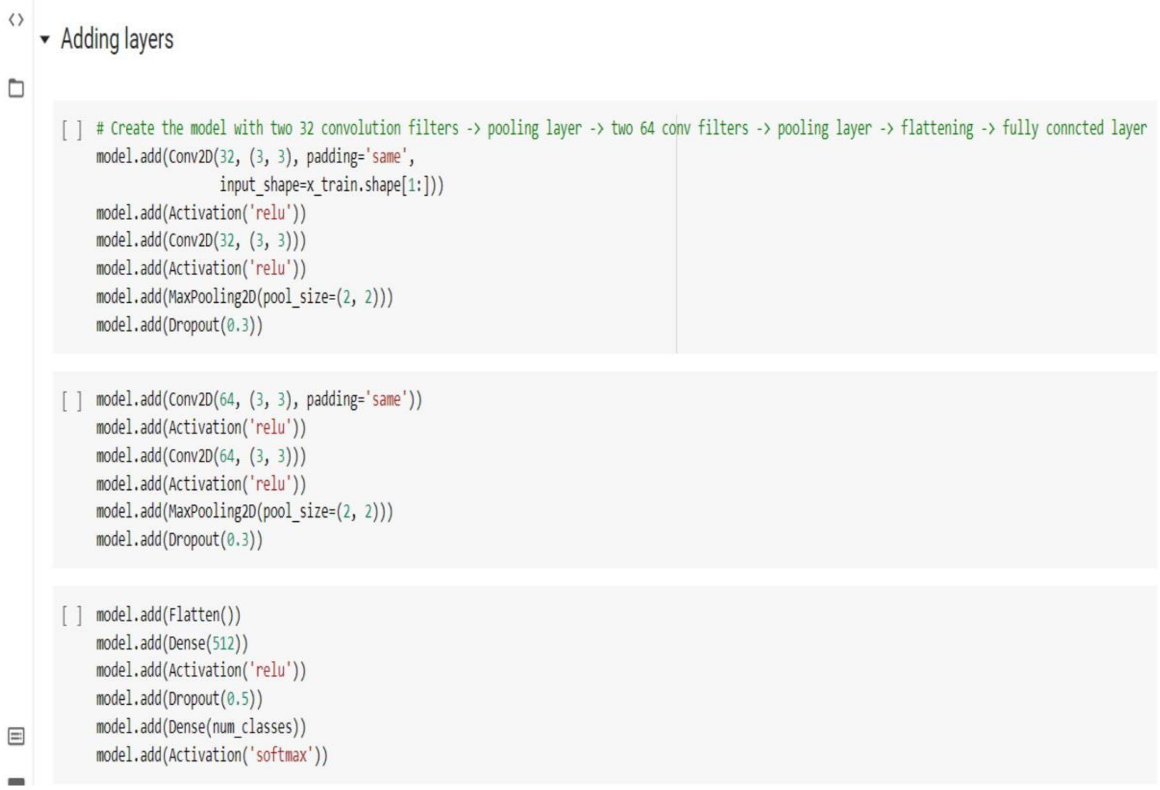
5. Initializing the Sequential Model:



Initialize the model

```
[ ]  # Initialize the model
     model = Sequential()
```

6. Adding Layers:



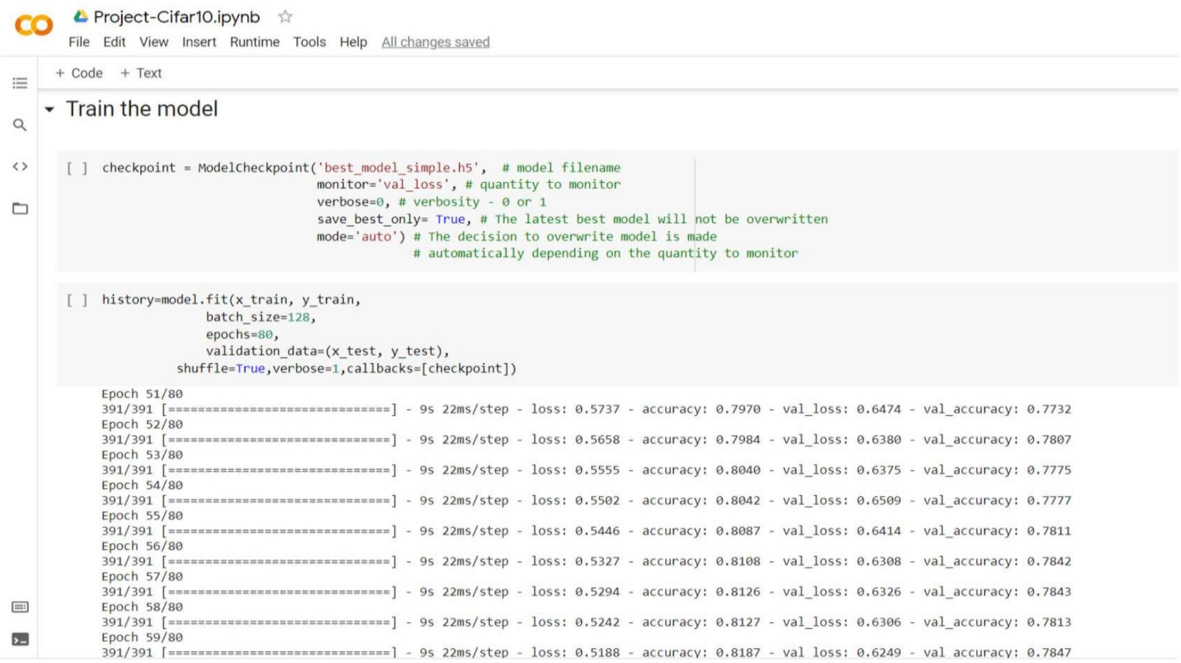
Adding layers

```
[ ]  # Create the model with two 32 convolution filters -> pooling layer -> two 64 conv filters -> pooling layer -> flattening -> fully conncted layer
     model.add(Conv2D(32, (3, 3), padding='same',
                      input_shape=x_train.shape[1:]))
     model.add(Activation('relu'))
     model.add(Conv2D(32, (3, 3)))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Dropout(0.3))
```

```
[ ]  model.add(Conv2D(64, (3, 3), padding='same'))
     model.add(Activation('relu'))
     model.add(Conv2D(64, (3, 3)))
     model.add(Activation('relu'))
     model.add(MaxPooling2D(pool_size=(2, 2)))
     model.add(Dropout(0.3))
```

```
[ ]  model.add(Flatten())
     model.add(Dense(512))
     model.add(Activation('relu'))
     model.add(Dropout(0.5))
     model.add(Dense(num_classes))
     model.add(Activation('softmax'))
```

7. Training the Model:

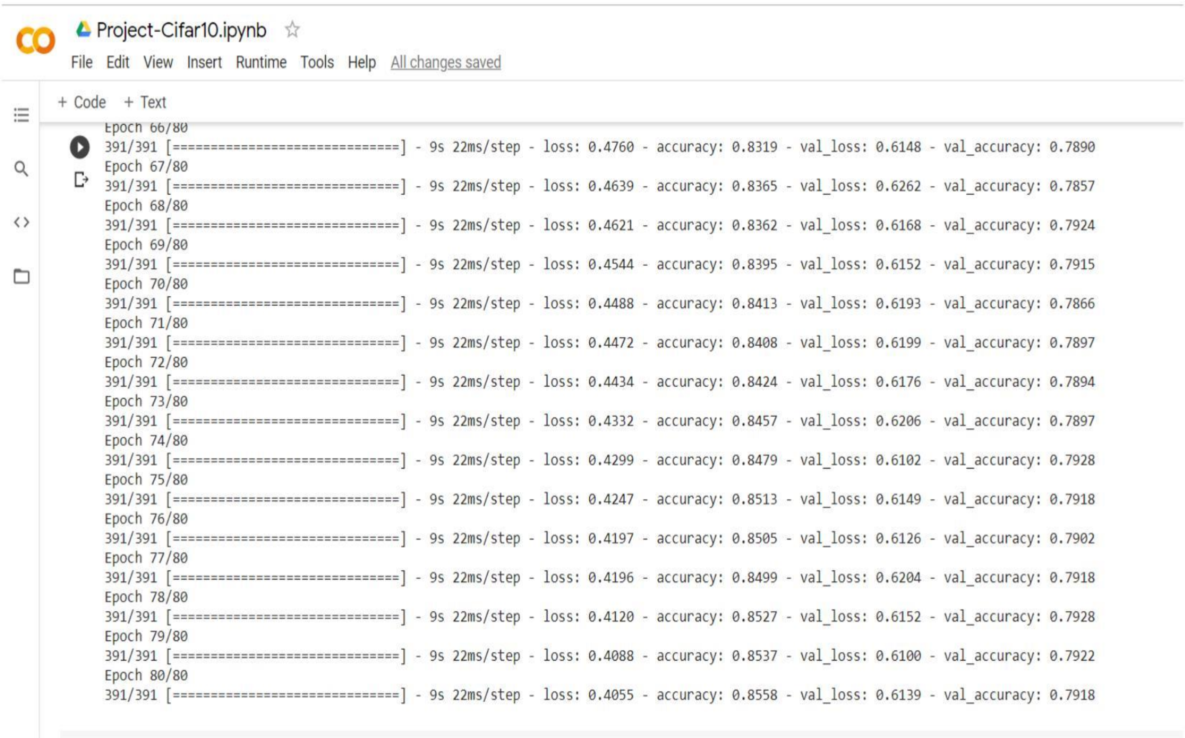


Train the model

```
[ ]  checkpoint = ModelCheckpoint('best_model_simple.h5',  # model filename
                                 monitor='val_loss', # quantity to monitor
                                 verbose=0, # verbosity - 0 or 1
                                 save_best_only= True, # The latest best model will not be overwritten
                                 mode='auto') # The decision to overwrite model is made
                                              # automatically depending on the quantity to monitor
```

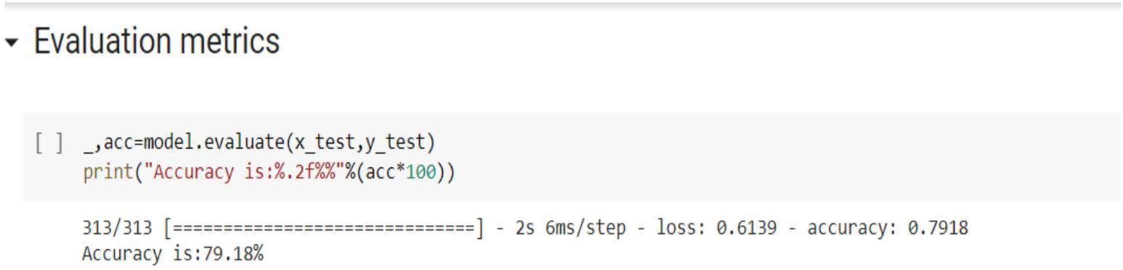
```
[ ]  history=model.fit(x_train, y_train,
                   batch_size=128,
                   epochs=80,
                   validation_data=(x_test, y_test),
                shuffle=True,verbose=1,callbacks=[checkpoint])
```

```
Epoch 51/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5737 - accuracy: 0.7970 - val_loss: 0.6474 - val_accuracy: 0.7732
Epoch 52/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5658 - accuracy: 0.7984 - val_loss: 0.6380 - val_accuracy: 0.7807
Epoch 53/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5555 - accuracy: 0.8040 - val_loss: 0.6375 - val_accuracy: 0.7775
Epoch 54/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5502 - accuracy: 0.8042 - val_loss: 0.6509 - val_accuracy: 0.7777
Epoch 55/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5446 - accuracy: 0.8087 - val_loss: 0.6414 - val_accuracy: 0.7811
Epoch 56/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5327 - accuracy: 0.8108 - val_loss: 0.6308 - val_accuracy: 0.7842
Epoch 57/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5294 - accuracy: 0.8126 - val_loss: 0.6326 - val_accuracy: 0.7843
Epoch 58/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5242 - accuracy: 0.8127 - val_loss: 0.6306 - val_accuracy: 0.7813
Epoch 59/80
391/391 [==============================] - 9s 22ms/step - loss: 0.5188 - accuracy: 0.8187 - val_loss: 0.6249 - val_accuracy: 0.7847
```



```
Epoch 66/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4760 - accuracy: 0.8319 - val_loss: 0.6148 - val_accuracy: 0.7890
Epoch 67/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4639 - accuracy: 0.8365 - val_loss: 0.6262 - val_accuracy: 0.7857
Epoch 68/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4621 - accuracy: 0.8362 - val_loss: 0.6168 - val_accuracy: 0.7924
Epoch 69/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4544 - accuracy: 0.8395 - val_loss: 0.6152 - val_accuracy: 0.7915
Epoch 70/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4488 - accuracy: 0.8413 - val_loss: 0.6193 - val_accuracy: 0.7866
Epoch 71/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4472 - accuracy: 0.8408 - val_loss: 0.6199 - val_accuracy: 0.7897
Epoch 72/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4434 - accuracy: 0.8424 - val_loss: 0.6176 - val_accuracy: 0.7894
Epoch 73/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4332 - accuracy: 0.8457 - val_loss: 0.6206 - val_accuracy: 0.7897
Epoch 74/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4299 - accuracy: 0.8479 - val_loss: 0.6102 - val_accuracy: 0.7928
Epoch 75/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4247 - accuracy: 0.8513 - val_loss: 0.6149 - val_accuracy: 0.7918
Epoch 76/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4197 - accuracy: 0.8505 - val_loss: 0.6126 - val_accuracy: 0.7902
Epoch 77/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4196 - accuracy: 0.8499 - val_loss: 0.6204 - val_accuracy: 0.7918
Epoch 78/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4120 - accuracy: 0.8527 - val_loss: 0.6152 - val_accuracy: 0.7928
Epoch 79/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4088 - accuracy: 0.8537 - val_loss: 0.6100 - val_accuracy: 0.7922
Epoch 80/80
391/391 [==============================] - 9s 22ms/step - loss: 0.4055 - accuracy: 0.8558 - val_loss: 0.6139 - val_accuracy: 0.7918
```

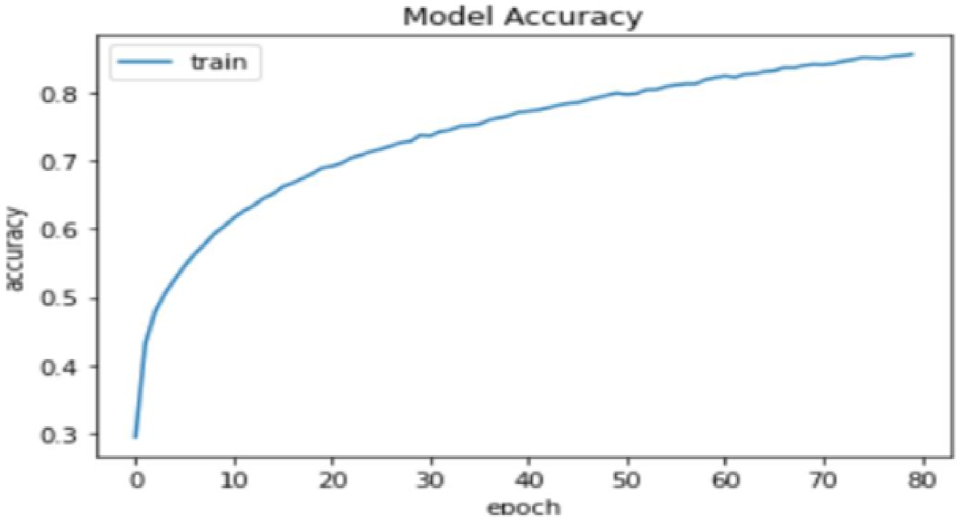
V. RESULTS AND GRAPHS

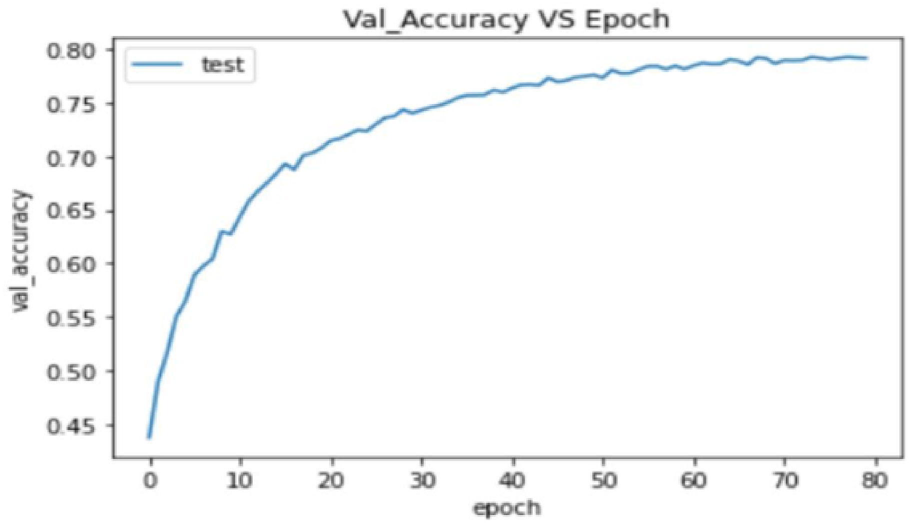


```
_,acc=model.evaluate(x_test,y_test)
print("Accuracy is:%.2f%%"%(acc*100))
```

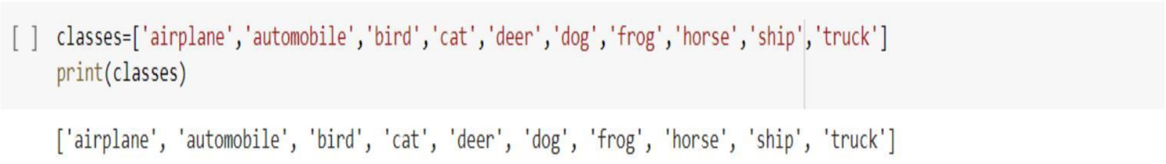
```
313/313 [==============================] - 2s 6ms/step - loss: 0.6139 - accuracy: 0.7918
Accuracy is:79.18%
```

The model gave an accuracy of 79.18%. This can be seen in the figure below.



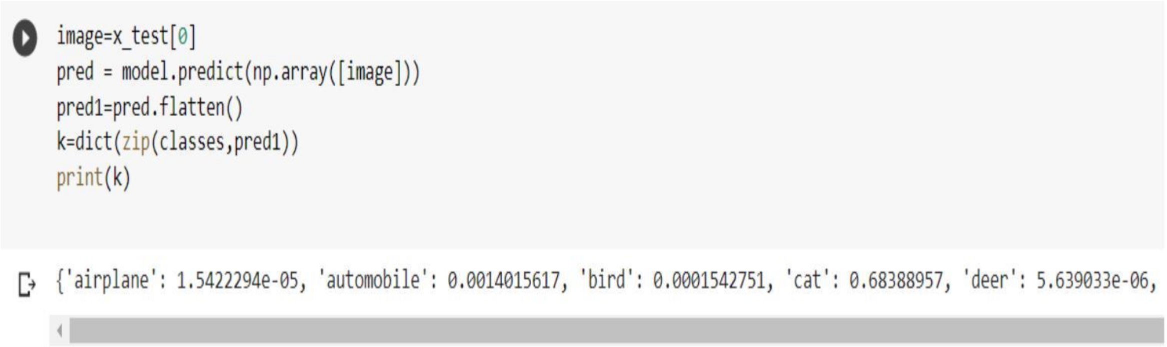


Predictions:



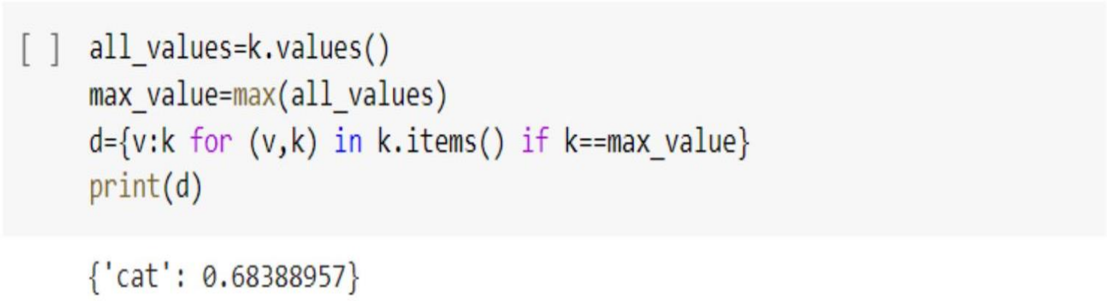
```
[ ] classes=['airplane','automobile','bird','cat','deer','dog','frog','horse','ship','truck']
    print(classes)

    ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```



```
image=x_test[0]
pred = model.predict(np.array([image]))
pred1=pred.flatten()
k=dict(zip(classes,pred1))
print(k)

{'airplane': 1.5422294e-05, 'automobile': 0.0014015617, 'bird': 0.0001542751, 'cat': 0.68388957, 'deer': 5.639033e-06,
```



```
[ ] all_values=k.values()
    max_value=max(all_values)
    d={v:k for (v,k) in k.items() if k==max_value}
    print(d)

    {'cat': 0.68388957}
```

VI. CONCLUSION

CNNs are Deep Neural Networks (DNNs) that include several layers, such as convolutional layers, pooling layers, and fully connected layers. Convolutional neural networks (CNNs) are frequently employed in pattern and image recognition challenges because they provide several benefits over other approaches. CNNs are important in the field of computer vision and have a wide range of applications. The Cifar10 dataset, which we imported from Keras, comprises ten classes, and we built a model that accurately predicts the supplied image. Image classification is one of the most important applications of computer vision, and it has aided in the understanding of the convolution process and the many layers of Convolutional neural networks. Because the future will be more automated and digitalized, these technologies will undoubtedly aid us in comprehending what occurs behind the scenes.

VII. REFERENCES

[1] Lillesand, T.M. and Kiefer, R.W. and Chipman, J.W., in “Remote Sensing and Image Interpretation” 5th ed. Wiley, 2004

[2] Li Deng and Dong Yu “Deep Learning: methods and applications” by Microsoft research [Online] available at: http://research.microsoft.com/pubs/209355/NOW-Book-Revised- Feb2014-online.pdf

[3] McCulloch, Warren; Walter Pitts, "A Logical Calculus of Ideas Im- manent in Nervous Activity”, Bulletin of Mathematical Biophysics 5 (4): 115–133(1943)

[4] Hubel, D. and Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology (Lon- don), 195, 215–243C. J. Kaufman, Rocky Mountain Research Laboratories, Boulder, Colo., personal communication, 1992. (Personal communication)

[5] Yann LeCun, Leon Bottou, Yodhua Bengio and Patrick Haffner, “Gradient -Based Learning Applied to Document Recognition”, Proc. Of IEEE, November 1998.

[6] S. L. Phung and A. Bouzerdoum,”MATLAB library for convolutional neural network,” Technical Report, ICT Research Institute, Visual and Audio Signal Processing Laboratory, University of Wollongong. Available at: http://www.uow.edu.au/˜phung

[7] Tutorial on deep learning [Online] available at : http://deeplearning.net/tutorial/lenet.html

[8] Adelson, Edward H., Charles H. Anderson, James R. Bergen, Peter J. Burt, and Joan M. Ogden. "Pyramid methods in image processing." RCA engineer 29, no. 6 (1984): 33-41.