PROJECT REPORT

on

Customized CNN

Submitted by

Md Gulam Nabi 11713122

Mayank Pratap Singh 11713175

Section

KM038

Program

B.Tech (Computer Science & Engineering)

Under the Guidance of

Mr. Sanjay Kumar

School of Computer Science and Engineering

Lovely Professional University, Phagwara

(October, 2020)

Abstract

Artificial Intelligence has been witnessing a monumental growth in bridging the gap between the capabilities of humans and machines. One of many such areas is the domain of Computer Vision.
The advancements in Computer Vision with Deep Learning has been constructed and perfected with time, primarily over one particular algorithm — a **Convolutional Neural Network (CNN)**.

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories for image recognition, image classifications, Objects detections, recognizing faces etc., are some of the areas where CNNs are widely used.

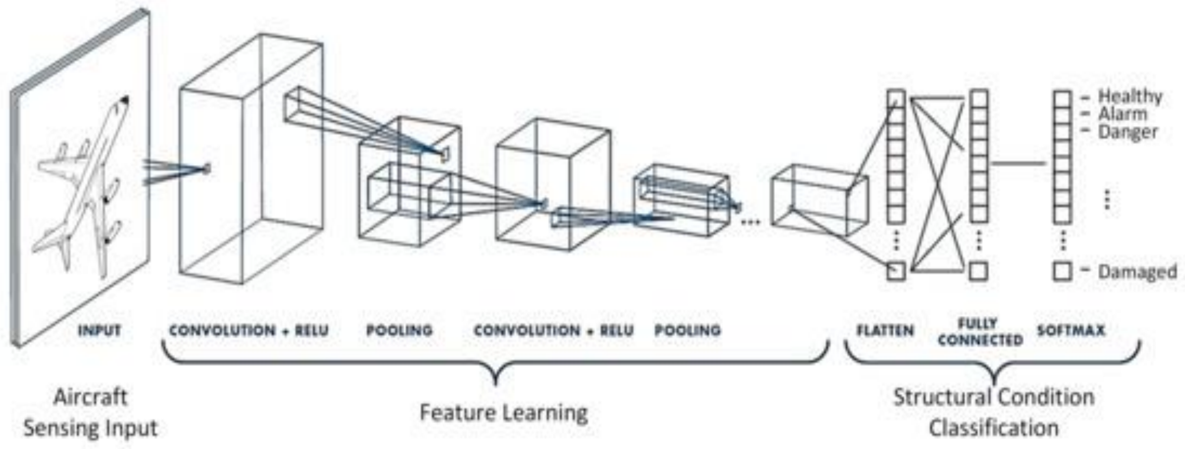
The CIFAR-10 dataset consists of 60000x32 x 32 colour images divided in 10 classes, with 6000 images in each class. There are 50000 training images and 10000 test images.
We will use Keras with TensorFlow during the training of the model.
We will then output a random set of images in the form of 2 rows and 5 column with their corresponding predicted class name, probability and the true value of that image.

The main focus of this project is on how to apply CNN in real life using python, to learn more about CNN.

Introduction

In neural networks, Convolutional neural network (CNNs) is one of the main categories for image recognition, image classifications, Objects detections, recognizing faces etc., are some of the areas where CNNs are widely used. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a CNN has just enough weights to look at a small patch of the image.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

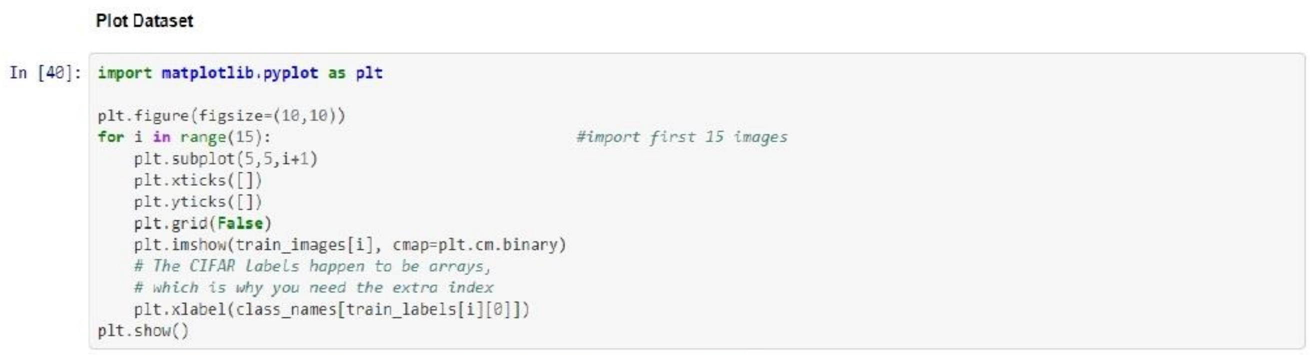


The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

Dataset

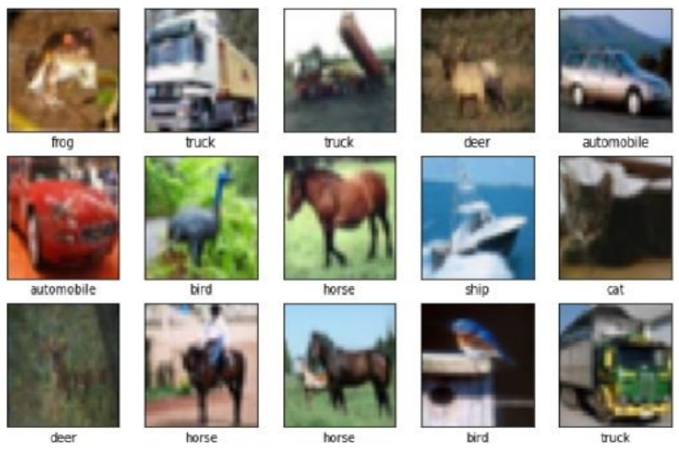
CIFAR is an acronym that stands for the Canadian Institute for Advanced Research and the CIFAR-10 dataset was developed along with the CIFAR-100 dataset by researchers at the CIFAR institute The classes are mutually exclusive and there is no overlap between them.
The dataset is comprised of 60,000 32×32-pixel color photographs of objects from 10 classes, such as frogs, birds, cats, ships, etc. The class labels and their standard associated integer values are listed below.

- 0: airplane
- 1: automobile
- 2: bird
- 3: cat
- 4: deer
- 5: dog
- 6: frog
- 7: horse
- 8: ship
- 9: truck



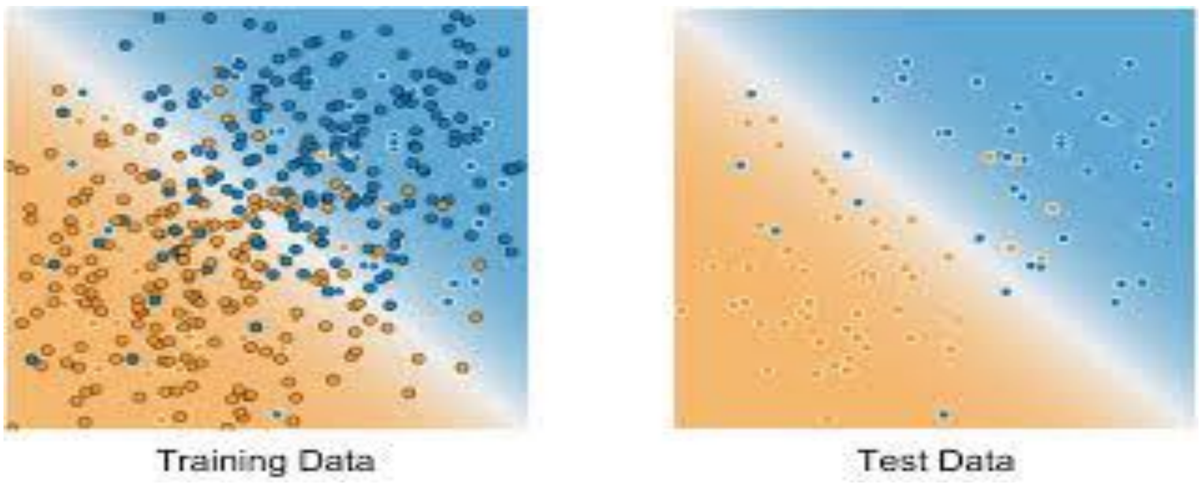
Plot Dataset

```
In [40]: import matplotlib.pyplot as plt

         plt.figure(figsize=(10,10))
         for i in range(15):                                  #import first 15 images
             plt.subplot(5,5,i+1)
             plt.xticks([])
             plt.yticks([])
             plt.grid(False)
             plt.imshow(train_images[i], cmap=plt.cm.binary)
             # The CIFAR labels happen to be arrays,
             # which is why you need the extra index
             plt.xlabel(class_names[train_labels[i][0]])
         plt.show()
```



These are very small images, much smaller than a typical photograph, and the dataset was intended for computer vision research.

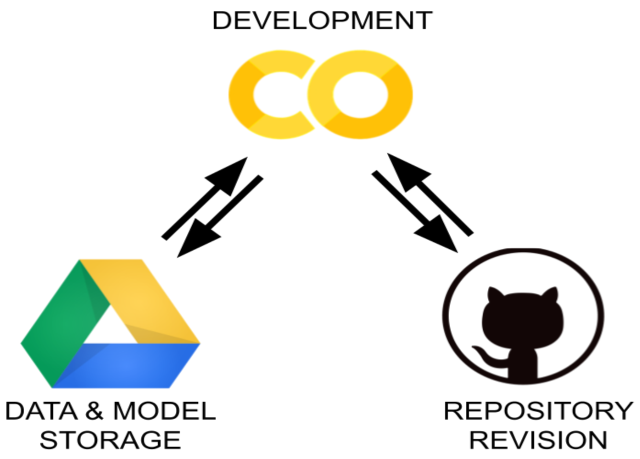
CIFAR-10 is a well-understood dataset and widely used for benchmarking computer vision algorithms in the field of machine learning. The problem is "solved." It is relatively straightforward to achieve 70% classification accuracy.



The CIFAR10 dataset contains 60,000 colour images in 10 classes, with 6,000 images in each class. The dataset is divided into five training batches of 50,000 training images and one test batch with 10,000 testing images. The test batch contains exactly 1000 randomly-selected images from each class.

Platform

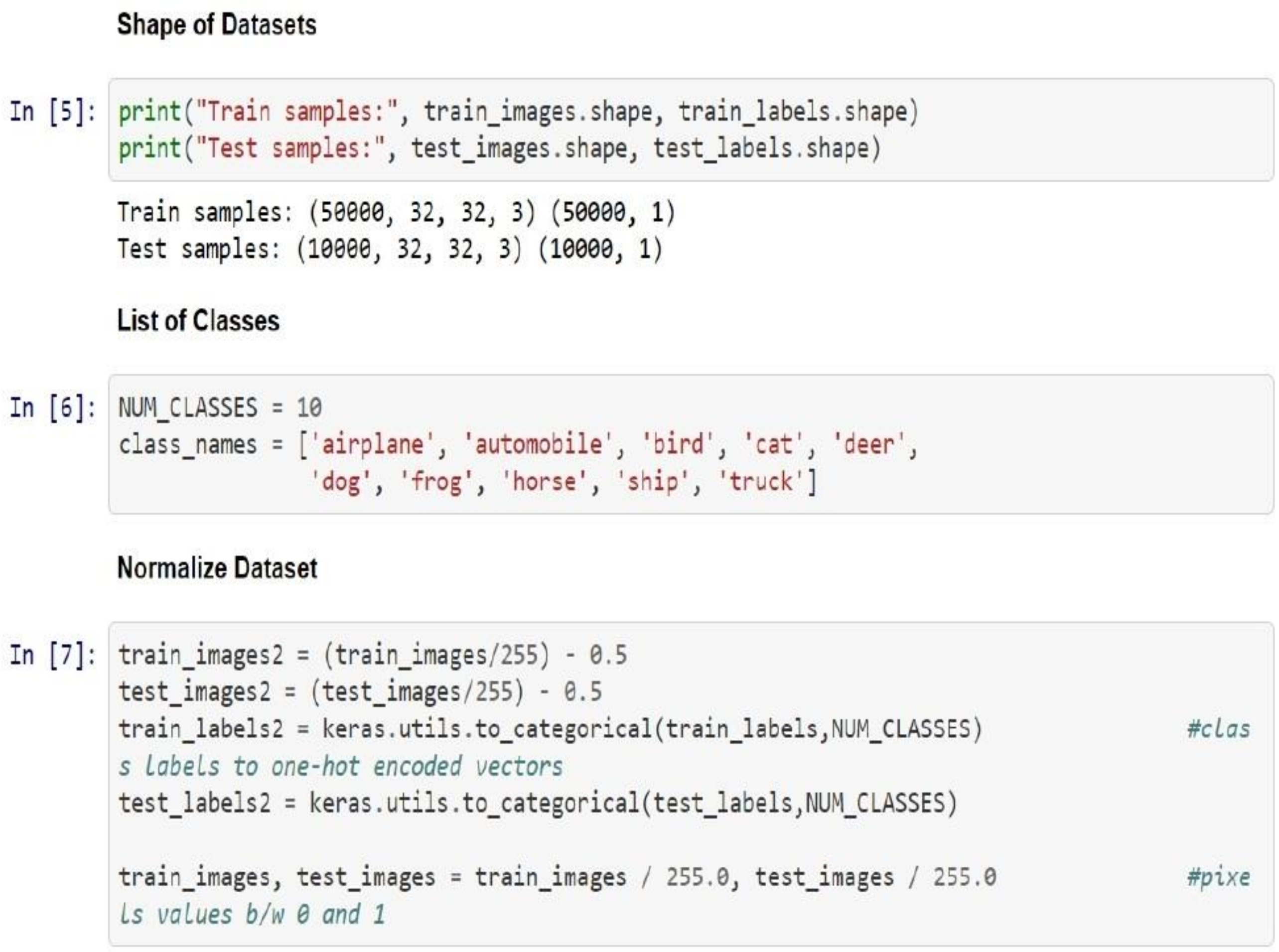
Colaboratory is a research tool for machine learning education and research. It's a Jupyter notebook environment that requires no setup to use. Colaboratory works with most major browsers, and is most thoroughly tested with latest versions of Chrome, Firefox and Safari. All Colaboratory notebooks are stored in Google Drive.



Colaboratory notebooks can be shared just as you would with Google Docs or Sheets. Simply click the Share button at the top right of any Colaboratory notebook, or follow these Google Drive file sharing instructions.

Data Pre-processing

Now check shape of dataset and make a list of classes.
Next normalize the training dataset’s pixel values to be between 0 and 1. Now convert class labels to one-hot encoded vectors.



Shape of Datasets

```
In [5]: print("Train samples:", train_images.shape, train_labels.shape)
        print("Test samples:", test_images.shape, test_labels.shape)

        Train samples: (50000, 32, 32, 3) (50000, 1)
        Test samples: (10000, 32, 32, 3) (10000, 1)
```

List of Classes

```
In [6]: NUM_CLASSES = 10
        class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
                       'dog', 'frog', 'horse', 'ship', 'truck']
```

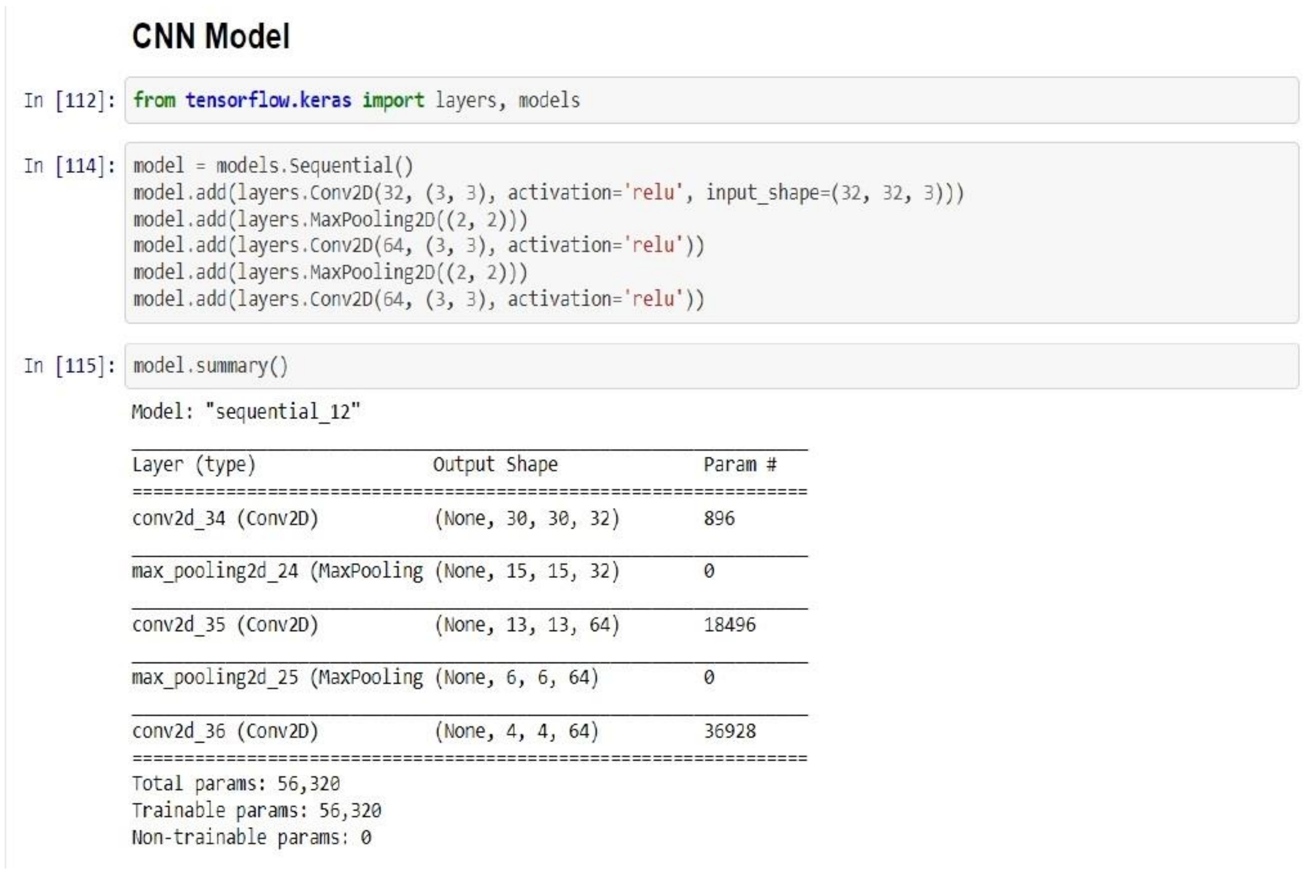
Normalize Dataset

```
In [7]: train_images2 = (train_images/255) - 0.5
        test_images2 = (test_images/255) - 0.5
        train_labels2 = keras.utils.to_categorical(train_labels,NUM_CLASSES)              #clas
        s labels to one-hot encoded vectors
        test_labels2 = keras.utils.to_categorical(test_labels,NUM_CLASSES)

        train_images, test_images = train_images / 255.0, test_images / 255.0             #pixe
        ls values b/w 0 and 1
```

CNN Model

Our input image is a tensor whose width is 32 pixels and height is 32 pixels with 3 channels representing RGB(red, green, blue) color intensities. Thus we need to define a model which takes (None, 32, 32, 3**)** input shape and predicts (None, 10) output with probabilities for all classes. None in shapes stands for batch. We can do this by passing the input shape argument to our first layer.

We will Stack **3** convolutional layers with kernel size (3, 3) with growing number of filters (32, 32, 64) with padding size of same so input and output images will have same dimensions. We will add 2x2 pooling layer after every convolutional layer (conv-pool scheme).



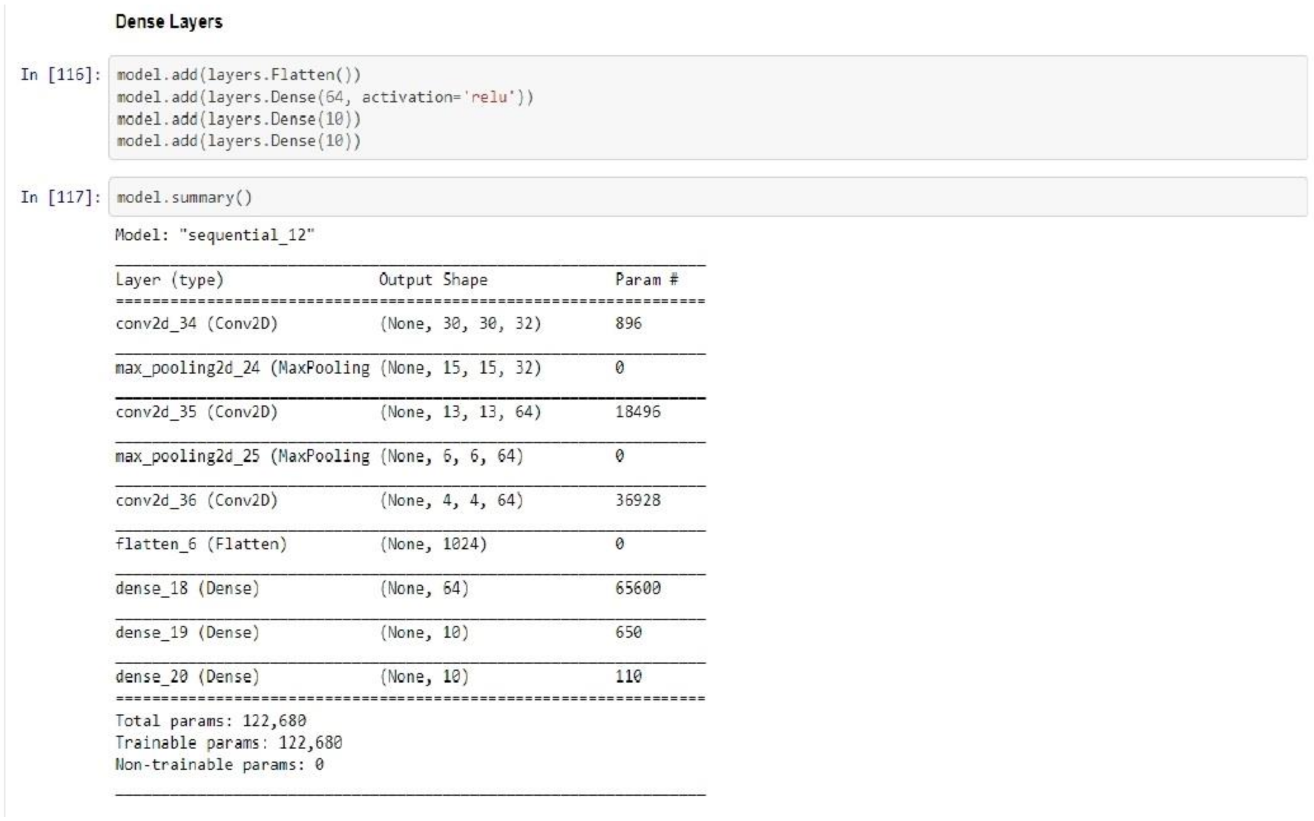
CNN Model

```
In [112]: from tensorflow.keras import layers, models
```

```
In [114]: model = models.Sequential()
          model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
          model.add(layers.MaxPooling2D((2, 2)))
          model.add(layers.Conv2D(64, (3, 3), activation='relu'))
          model.add(layers.MaxPooling2D((2, 2)))
          model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
In [115]: model.summary()

Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_34 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 4, 4, 64)          36928
=================================================================
Total params: 56,320
Trainable params: 56,320
Non-trainable params: 0
```

To complete our model, you will feed the last output tensor from the convolutional base (of shape (4, 4, 64)) into one or more Dense layers to perform classification. But we have to use Flatten layer before first dense layer to reshape input volume into a flat vector.



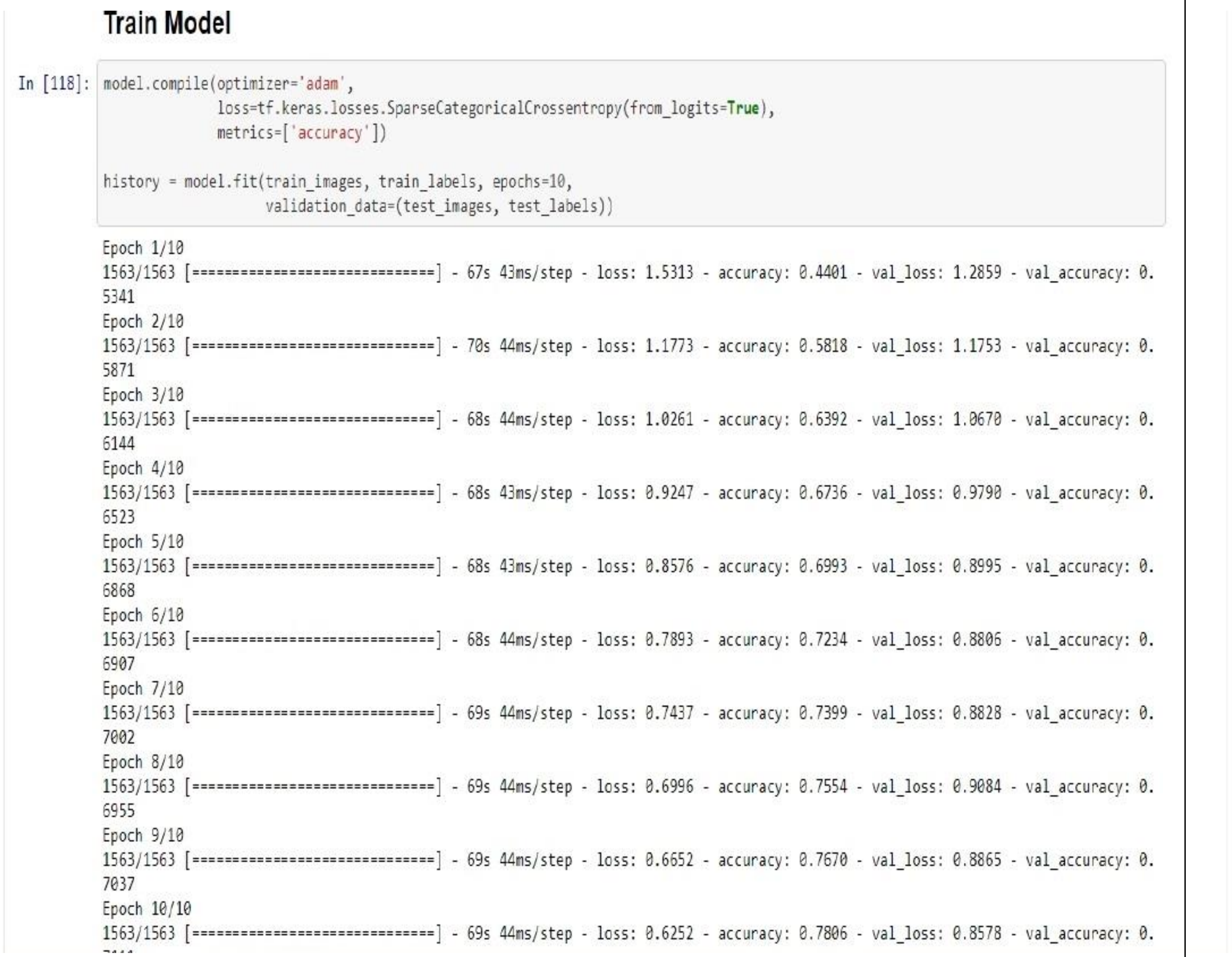
Dense Layers

```
In [116]: model.add(layers.Flatten())
          model.add(layers.Dense(64, activation='relu'))
          model.add(layers.Dense(10))
          model.add(layers.Dense(10))
```

```
In [117]: model.summary()

Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_34 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_36 (Conv2D)           (None, 4, 4, 64)          36928
_________________________________________________________________
flatten_6 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_18 (Dense)             (None, 64)                65600
_________________________________________________________________
dense_19 (Dense)             (None, 10)                650
_________________________________________________________________
dense_20 (Dense)             (None, 10)                110
=================================================================
Total params: 122,680
Trainable params: 122,680
Non-trainable params: 0
_________________________________________________________________
```

Train Model

Now define model architecture and then assign loss function, optimizer, and metrics in our model by using model.compile**( )** method.
Then we will initialize the number of epochs and use model.fit( ) function to start the training of our model by passing training data and validation data.
Initially we will train our model for 10 epochs and then gradually increase the number of epochs.



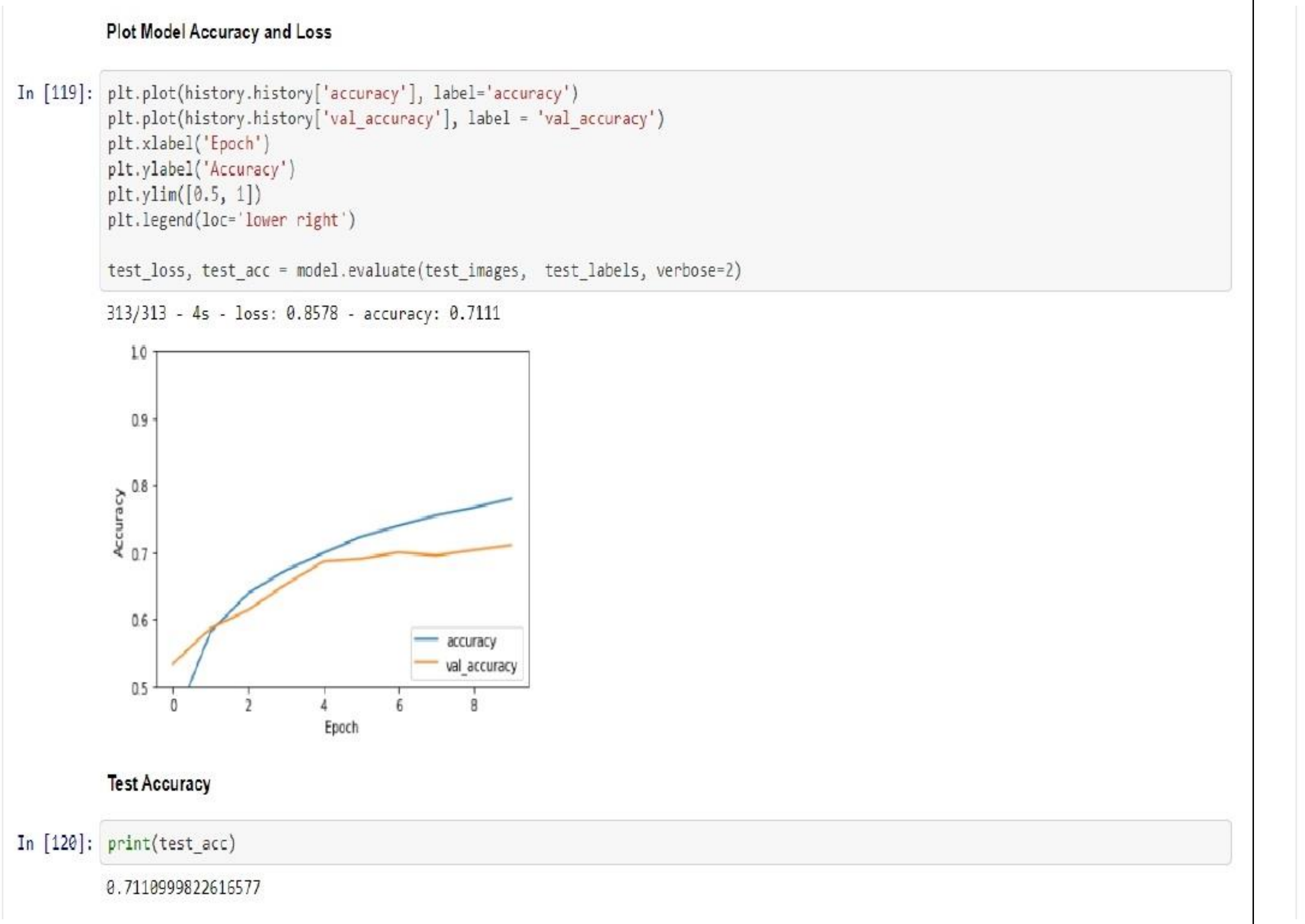
Train Model

```
In [118]: model.compile(optimizer='adam',
                        loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                        metrics=['accuracy'])

          history = model.fit(train_images, train_labels, epochs=10,
                              validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 67s 43ms/step - loss: 1.5313 - accuracy: 0.4401 - val_loss: 1.2859 - val_accuracy: 0.
5341
Epoch 2/10
1563/1563 [==============================] - 70s 44ms/step - loss: 1.1773 - accuracy: 0.5818 - val_loss: 1.1753 - val_accuracy: 0.
5871
Epoch 3/10
1563/1563 [==============================] - 68s 44ms/step - loss: 1.0261 - accuracy: 0.6392 - val_loss: 1.0670 - val_accuracy: 0.
6144
Epoch 4/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.9247 - accuracy: 0.6736 - val_loss: 0.9790 - val_accuracy: 0.
6523
Epoch 5/10
1563/1563 [==============================] - 68s 43ms/step - loss: 0.8576 - accuracy: 0.6993 - val_loss: 0.8995 - val_accuracy: 0.
6868
Epoch 6/10
1563/1563 [==============================] - 68s 44ms/step - loss: 0.7893 - accuracy: 0.7234 - val_loss: 0.8806 - val_accuracy: 0.
6907
Epoch 7/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.7437 - accuracy: 0.7399 - val_loss: 0.8828 - val_accuracy: 0.
7002
Epoch 8/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.6996 - accuracy: 0.7554 - val_loss: 0.9084 - val_accuracy: 0.
6955
Epoch 9/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.6652 - accuracy: 0.7670 - val_loss: 0.8865 - val_accuracy: 0.
7037
Epoch 10/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.6252 - accuracy: 0.7806 - val_loss: 0.8578 - val_accuracy: 0.
```

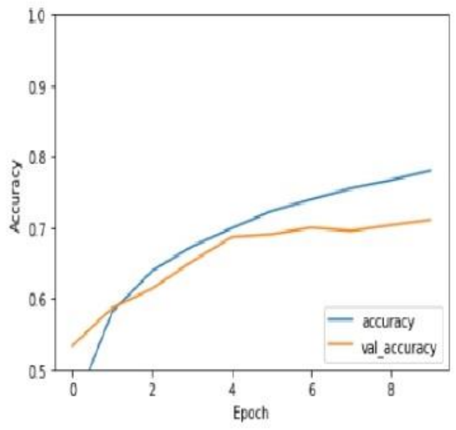
Outcome

While training the model with 10 epochs we have achieved maximum accuracy of 78% on training data and 71% accuracy on test data.
Output graph of the validation accuracy and validation loss.



Plot Model Accuracy and Loss

```
In [119]: plt.plot(history.history['accuracy'], label='accuracy')
          plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
          plt.xlabel('Epoch')
          plt.ylabel('Accuracy')
          plt.ylim([0.5, 1])
          plt.legend(loc='lower right')

          test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

313/313 - 4s - loss: 0.8578 - accuracy: 0.7111
```

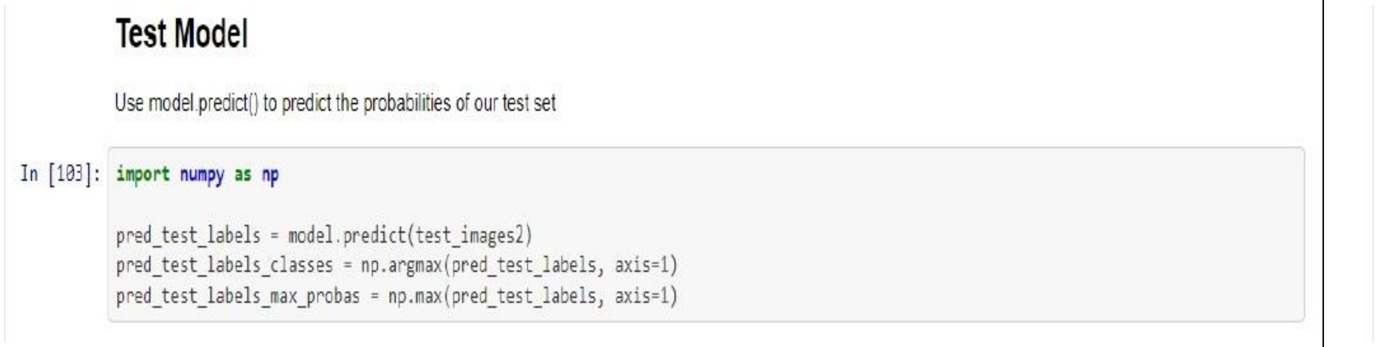


Test Accuracy

```
In [120]: print(test_acc)

0.7110999822616577
```

Test Model

Now predict the probabilities of images in our test set using the model.predict( ) function, then we will get the name of the classes with maximum probability using argmax function and get the corresponding probabilities using the np.max( ).

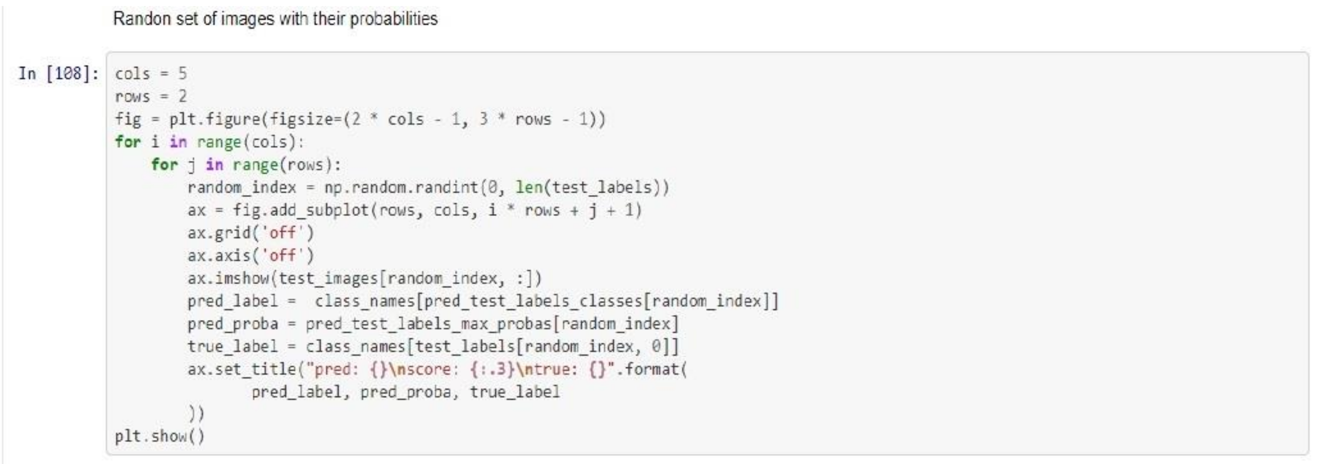


Test Model

Use model.predict() to predict the probabilities of our test set

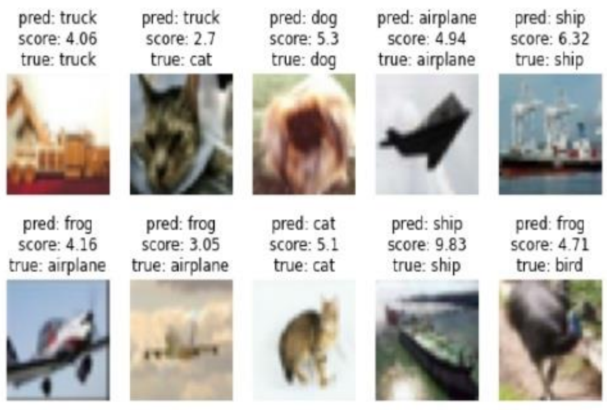
```
In [103]: import numpy as np

          pred_test_labels = model.predict(test_images2)
          pred_test_labels_classes = np.argmax(pred_test_labels, axis=1)
          pred_test_labels_max_probas = np.max(pred_test_labels, axis=1)
```

Then we will output a random set of images in the form of 2 rows and 5 column with their corresponding predicted class name, probability and the true value of that image.



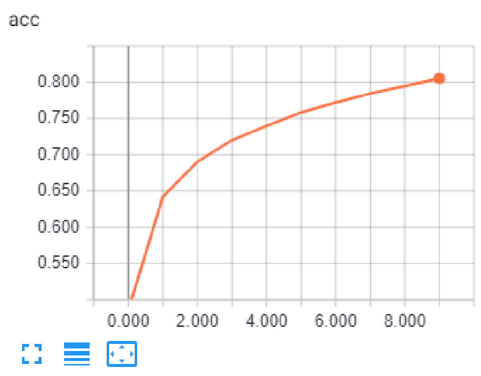
Randon set of images with their probabilities

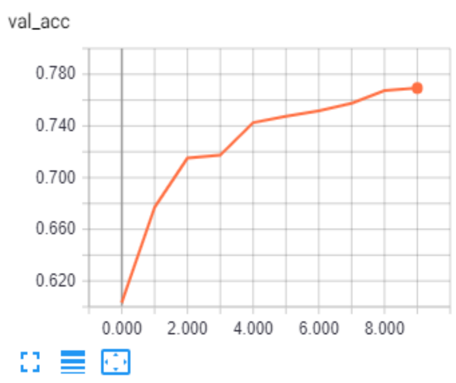
```
In [108]: cols = 5
          rows = 2
          fig = plt.figure(figsize=(2 * cols - 1, 3 * rows - 1))
          for i in range(cols):
              for j in range(rows):
                  random_index = np.random.randint(0, len(test_labels))
                  ax = fig.add_subplot(rows, cols, i * rows + j + 1)
                  ax.grid('off')
                  ax.axis('off')
                  ax.imshow(test_images[random_index, :])
                  pred_label =  class_names[pred_test_labels_classes[random_index]]
                  pred_proba = pred_test_labels_max_probas[random_index]
                  true_label = class_names[test_labels[random_index, 0]]
                  ax.set_title("pred: {}\nscore: {:.3}\ntrue: {}".format(
                         pred_label, pred_proba, true_label
                  ))
          plt.show()
```



Conclusion

Our CNN model has achieved an accuracy of 71% on validation set with only 10 epochs.
But as we gradually increase the number of epochs the model achieves max accuracy of 82% on training data and 76% on validation data.





Links

GitHub :- https://github.com/mayankprsingh/cifar10Classification/blob/main/cifar10.ipynb

Collab :- https://colab.research.google.com/drive/19x6d4e1vIUf05N4y6eYc-02e3AVlepFj?usp=sharing

References

https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://www.kaggle.com/mayank1212/kernelc1ac177b28

https://colab.research.google.com/drive/1W5QhPPMG1CBJ-tWKrR5jp2s1hxf8OiNc#scrollTo=Qu41da7vpbHi

http://parneetk.github.io/blog/cnn-cifar10/

https://medium.com/@udolf15/building-a-image-classifier-using-cnn-with-cifar-10-dataset-5682afa4f51

https://www.kaggle.com/c/cifar-10/discussion/40237

https://www.tensorflow.org/tutorials/images/cnn

https://www.tensorflow.org/tutorials/images/classification