Image Classification Using CIFAR-10 Dataset

Sai Ruthvik Bandaru

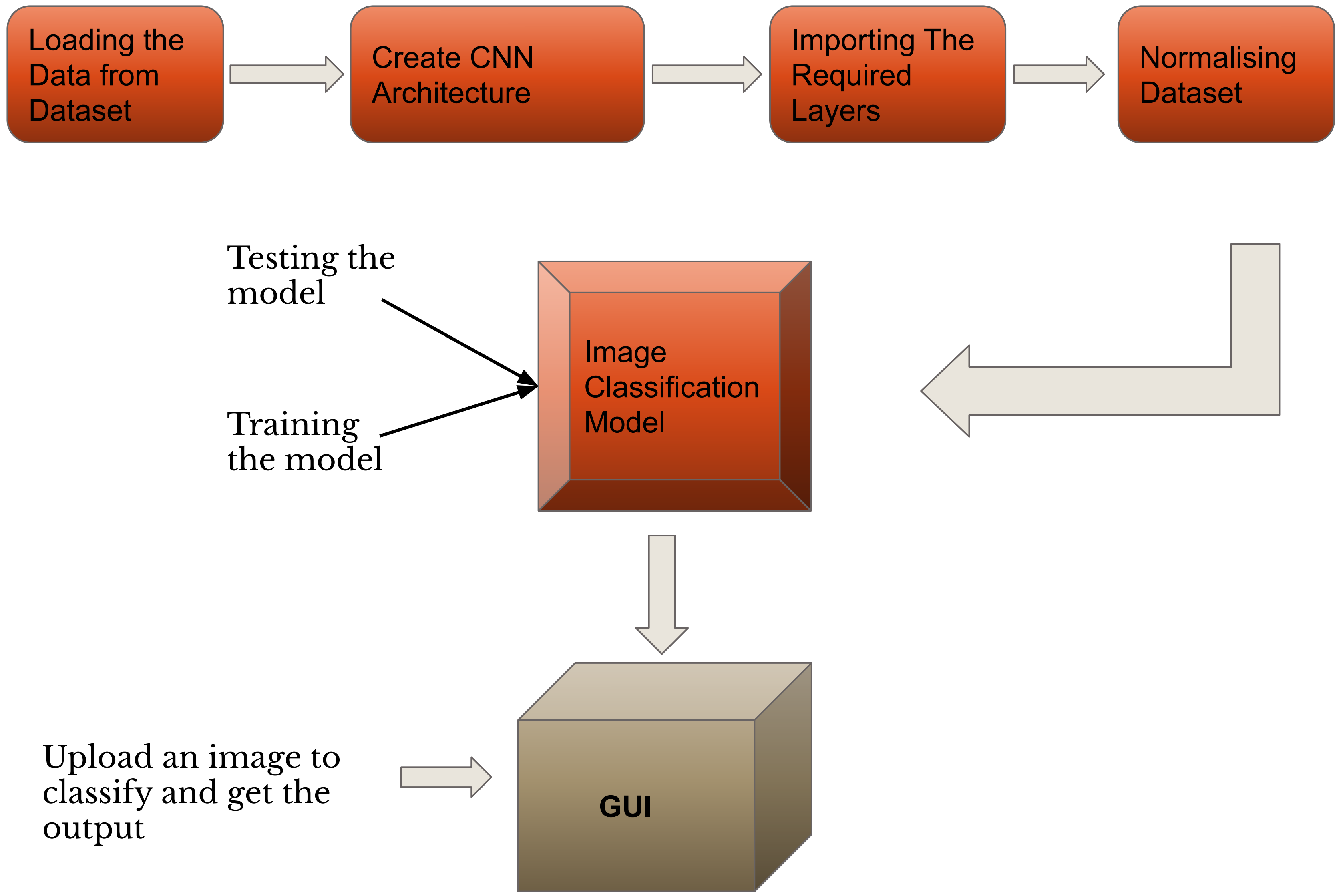
References

1. Image Classification Using CIFAR-10 Dataset
   https://towardsdatascience.com/cifar-10-image-classification-in-tensorflow-5b501f7dc77c
2. Guide to build an image classifier With CIFAR-10
   https://medium.com/@udolf15/building-a-image-classifier-using-cnn-with-cifar-10-dataset-5682afa4f51
3. Plotting the comparisons
   https://towardsdatascience.com/fixing-the-keyerror-acc-and-keyerror-val-acc-errors-in-keras-2-3-x-or-newer-b29b52609af9

Contents

- Overall framework
- Convolutional Neural Network (CNN)
- Building CNN in Keras and Tensorflow
- Plotting Results
- GUI-Tkinter
- Summary

Overall Framework



Loading the Data from Dataset
Create CNN Architecture
Importing The Required Layers
Normalising Dataset
Testing the model
Training the model
Image Classification Model
Upload an image to classify and get the output
GUI

Problem Formulation

Objective:
To Classify the image with KERAS and TENSORFLOW using CIFAR-10 Dataset.

Dataset details:

- CIFAR-10 known as Canadian Institute for Advanced Research.
- consists of 60000 30X30 colour images in 10 classes -6000 images per class.
- The Dataset has images from Aeroplane,Automobile, Bird,Cat,Deer,Dog,Frog,Horse,Ship,Truck.
- https://www.kaggle.com/c/cifar-10

Assumptions:

- Images are already available from the dataset module of keras.
- GUI is available using Tkinter python library.

Build The Convolutional Neural Network (CNN)

This shows us that there are totally 14 layers where 9 layers are convolutional layers and 5 layers are fully connected layers which are in a convolutional network and are practically a multilayer perceptron.The objective of the complete fully connected structure is to tune the weight parameters.The main task of the convolutional layer is to detect local conjunctions of features from the previous layer and mapping their appearance to a feature map.

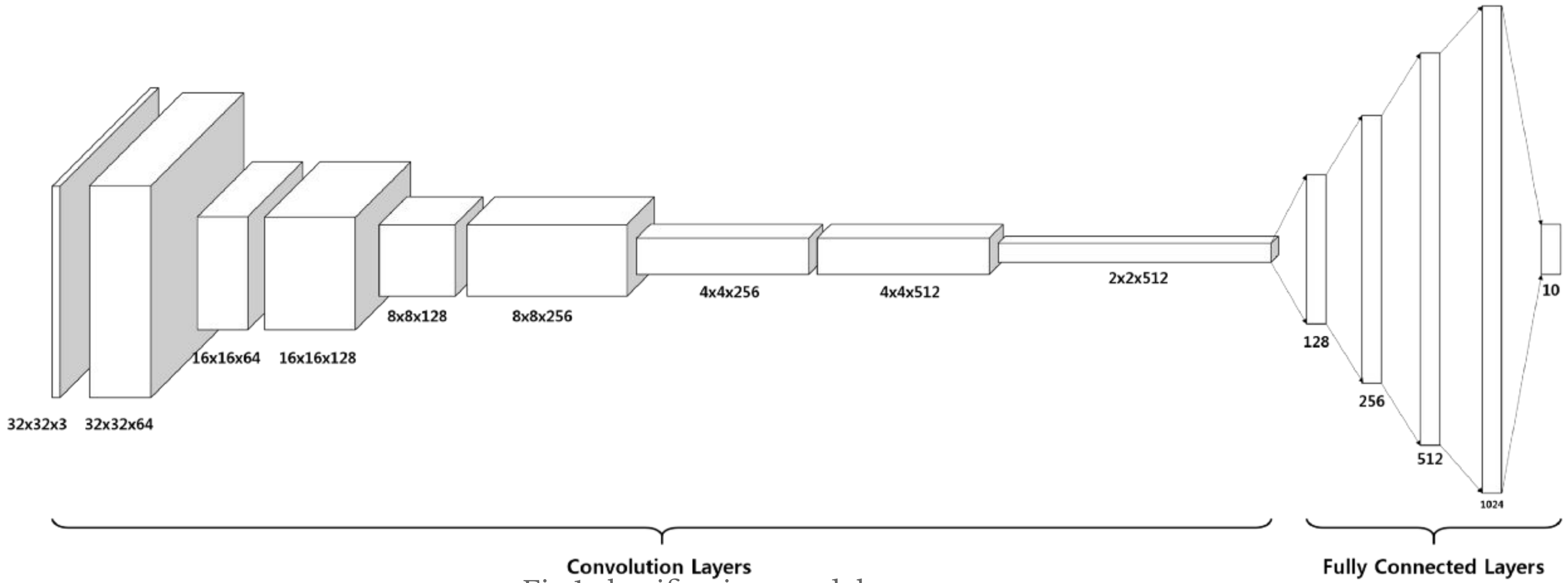


Fig.1 classification model

Source:https://miro.medium.com/max/1250/1*xlOZHo8svfDWDyxFFnMurQ.png

Rectified Linear Units (ReLU)

The rectified linear units are a special implementation that combines non-linearity and rectification layers in convolutional neural networks.

we are choosing ReLU because,The rectified linear units come with three significant advantages in convolutional neural networks compared to the traditional logistic or hyperbolic tangent activation functions:

- They reduce the problem of a vanishing gradient that is common in deep neural architectures.
- Rectified linear units threshold negative values to zero, and therefore solve the cancellation problem which occur in Convolutional Networks.
- Rectified linear units consist of only simple operations in terms of computation and therefore much more efficient to implement in convolutional neural networks.

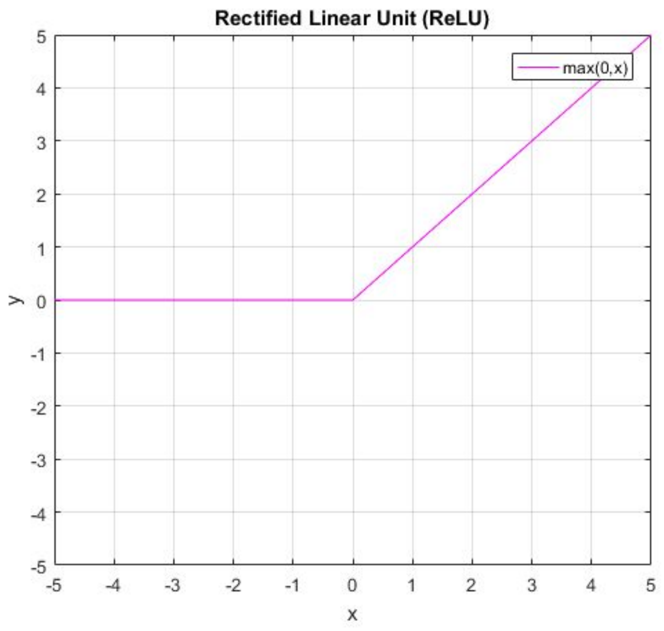


Fig.2 ReLU plot

Source:ReLU Function

Max Pooling

- Spatial Pooling reduces the dimensions of each feature map but will get the most important information.Spatial Pooling can be of different types: Max, Average, Sum etc.
- In case of Max Pooling, we define a spatial neighborhood and take the largest element from the rectified feature map within that window.In practice, Max Pooling has been shown to work better so we are performing Max Pooling in this project.

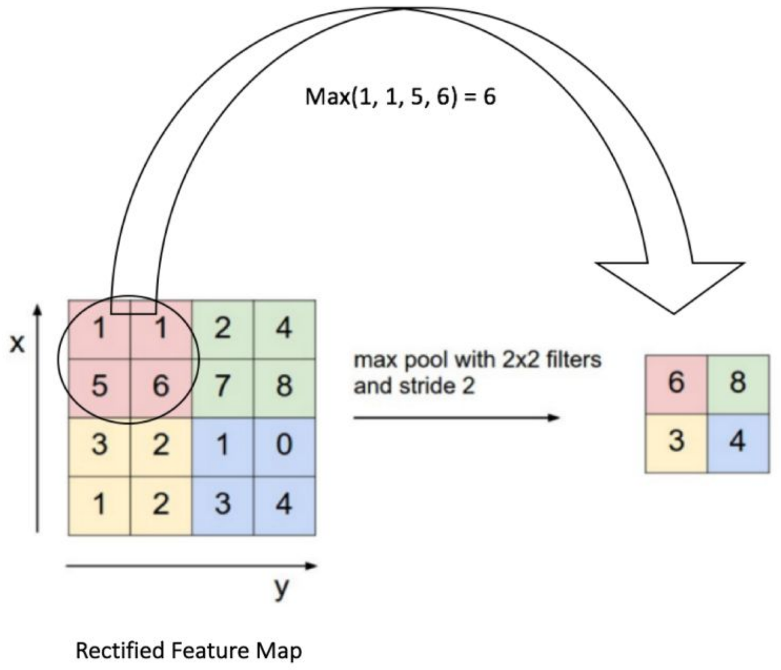


Fig.3 Max Pool with 2X2

The above figure shows us that Max Pooling operation on a Rectified Feature map is done using a 2X2 window.

Source:Max Pooling

Flatten

- Flattening is used to convert the data into a 1-dimensional array for giving input to the next layer.
- We flatten the output of the convolutional layers to create a single long feature vector which will be connected to the final classification model, which is called a fully-connected layer.
- we can see in the below figure that it is basically unstacking the volume into an array.

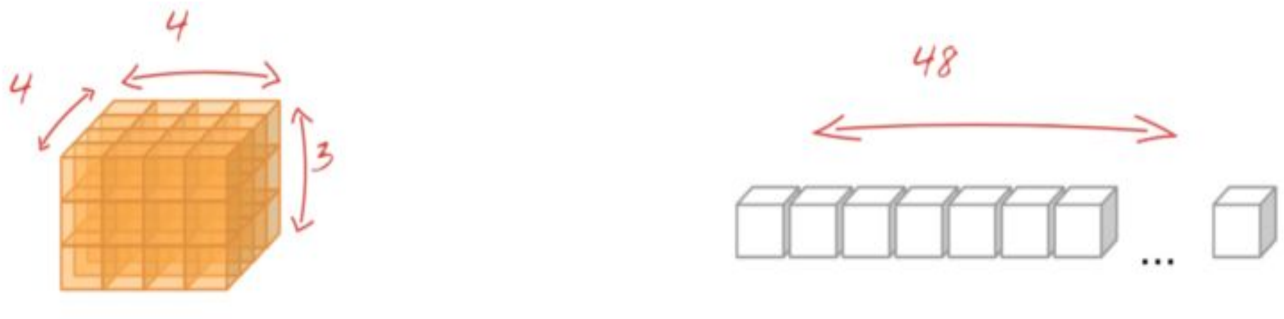


Fig.4 Flatten to 1-D array

Dense

- Any single convolution can be replaced by a Dense layer that would perform the same association of neighboring pixels for each pixel.
- It would mean one neuron per pixel with not-null coefficients only on the neighbors.
- It basically reduces the number of parameters to learn with a very good performance.

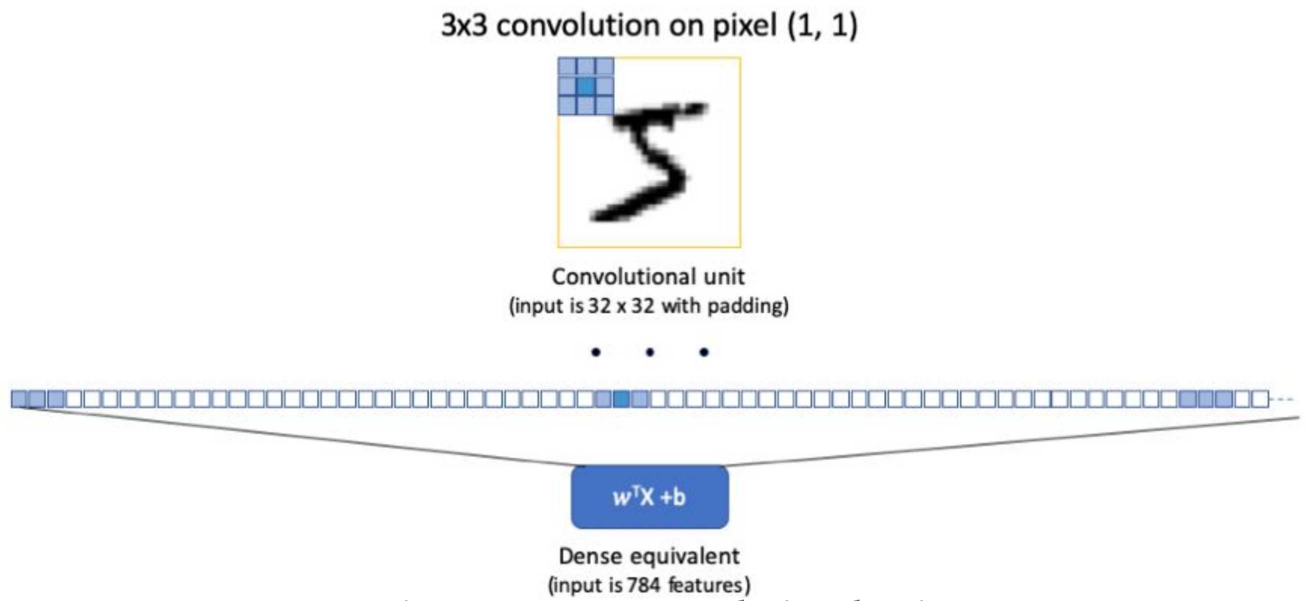


Fig.5 Dense on Convolutional unit

Source:Dense

Load the Data

In this project we need not download CIFAR-10 dataset as Keras.dataset already have CIFAR-10 dataset,so we need to import that also.Now,we will use the dataset from keras and divide them into training set and testing set.

```
from keras.datasets import cifar10
import matplotlib.pyplot as plt
(train_X,train_Y),(test_X,test_Y)=cifar10.load_data()
```

Importing the layers and modules to create our CNN architecture

This model is totally consisting of 14 layers in total.So we are going to now import the necessary blocks to make the model.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
```

Importing the layers and modules to create our CNN architecture (Cont’d)

- CONV2D is used to perform convolution
- MaxPooling2D is used to perform 2D max pooling.
- Flatten is used to flatten the input and helps it to not affect the batch size
- Dense helps in fully connecting layer.
- Activation helps in applying an activation function.

Normalize the dataset

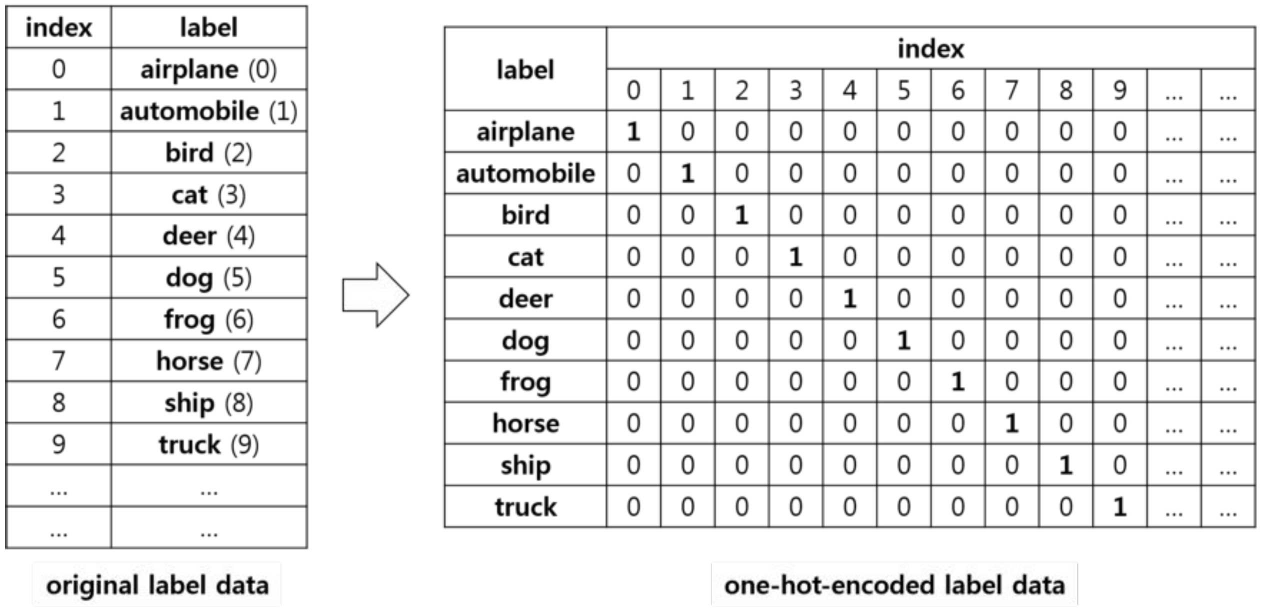
Normalization is applied as part of data preparation for Deep learning. The main objective of this normalization is to change the values of numeric columns in the dataset to a common scale without changing any differences in the ranges of values.So when you think about the image data,all values must be ranging from 0 to 255.

```
train_x=train_X.astype('float32')
test_X=test_X.astype('float32')
train_X=train_X/255.0
test_X=test_X/255.0
```

One-hot encoding for target classes

Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical.In this project for example for the classes named ‘Dog’ and ‘Cat’ undergo through one-hot encoding which allows the representation of categorical data to be more expressive in helping the problem easier for the network to model.

```
train_Y=np_utils.to_categorical(train_Y)
test_Y=np_utils.to_categorical(test_Y)
num_classes=test_Y.shape[1]
```


original label data
one-hot-encoded label data

index	label
0	**airplane** (0)
1	**automobile** (1)
2	**bird** (2)
3	**cat** (3)
4	**deer** (4)
5	**dog** (5)
6	**frog** (6)
7	**horse** (7)
8	**ship** (8)
9	**truck** (9)
...	...
...	...

label	index											
	0	1	2	3	4	5	6	7	8	9	...	...
airplane	**1**	0	0	0	0	0	0	0	0	0	...	...
automobile	0	**1**	0	0	0	0	0	0	0	0	...	...
bird	0	0	**1**	0	0	0	0	0	0	0	...	...
cat	0	0	0	**1**	0	0	0	0	0	0	...	...
deer	0	0	0	0	**1**	0	0	0	0	0	...	...
dog	0	0	0	0	0	**1**	0	0	0	0	...	...
frog	0	0	0	0	0	0	**1**	0	0	0	...	...
horse	0	0	0	0	0	0	0	**1**	0	0	...	...
ship	0	0	0	0	0	0	0	0	**1**	0	...	...
truck	0	0	0	0	0	0	0	0	0	**1**	...	...

Fig.6 One-Hot Encoding

Source: One-hot encoding

Create the sequential model

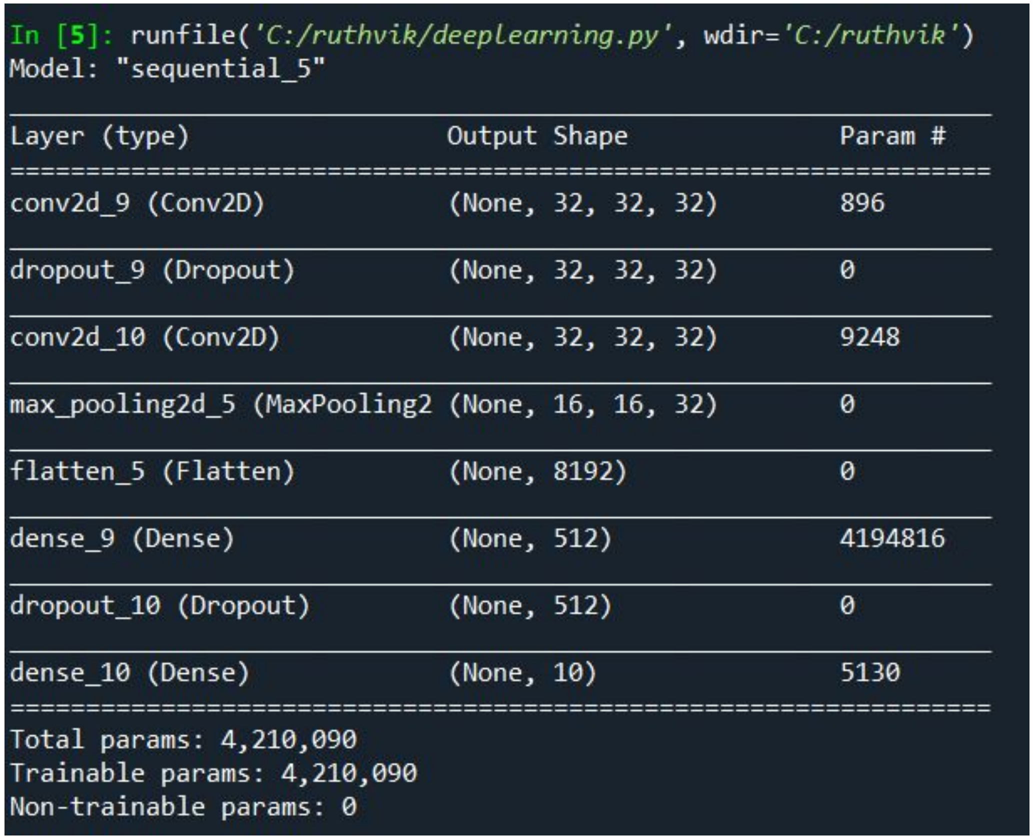
- We are performing padding such as when padding="same" adds zero padding to the input, so that the output has the same width and height.
- We will add 2X2 pooling layer after every 2 convolutional layers.
- We need to use Flatten layer before first dense layer to reshape input volume into a flat vector.Dropout layer after every pooling layer will be used to reduce overfitting in the model with setting the fraction value to .5 so that .5 fraction of units will be dropped.
- The kernel constraints are generally used to reduce the overfitting of the deep learning neural network model and 'maxnorm, helps to force weights to have a magnitude at or below a given limit.

```
model=Sequential()
model.add(Conv2D(32,(3,3),input_shape=(32,32,3),
    padding='same',activation='relu',
    kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32,(3,3),activation='relu',padding='same',kernel_constraint
=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(512,activation='relu',kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

Summary Of The Model

To view the model summary for better understanding of model architecture we use the code as shown below:

model.summary( )



```
In [5]: runfile('C:/ruthvik/deeplearning.py', wdir='C:/ruthvik')
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_9 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
dropout_9 (Dropout)          (None, 32, 32, 32)        0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 8192)              0         
_________________________________________________________________
dense_9 (Dense)              (None, 512)               4194816   
_________________________________________________________________
dropout_10 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 10)                5130      
=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
```

Fig.7 Model Summary

Configure the optimizer and compile the model

SGD is basically an optimiser and is defined as Stochastic gradient descent is an iterative method for optimizing an objective function with suitable smoothness properties.We will apply the arguments of 'SGD( )' as shown below:

- lr: float >= 0. Learning rate.
- momentum: float >= 0. Parameter updates momentum.
- decay: float >= 0. Learning rate decay over each update.
- nesterov: boolean.,Whether to apply Nesterov momentum or not

```
sgd=SGD(lr=0.01,momentum=0.9,decay=(0.01/25),nesterov=False)
model.compile(loss='categorical_crossentropy',
  optimizer=sgd,
  metrics=['accuracy'])
```

Train the model

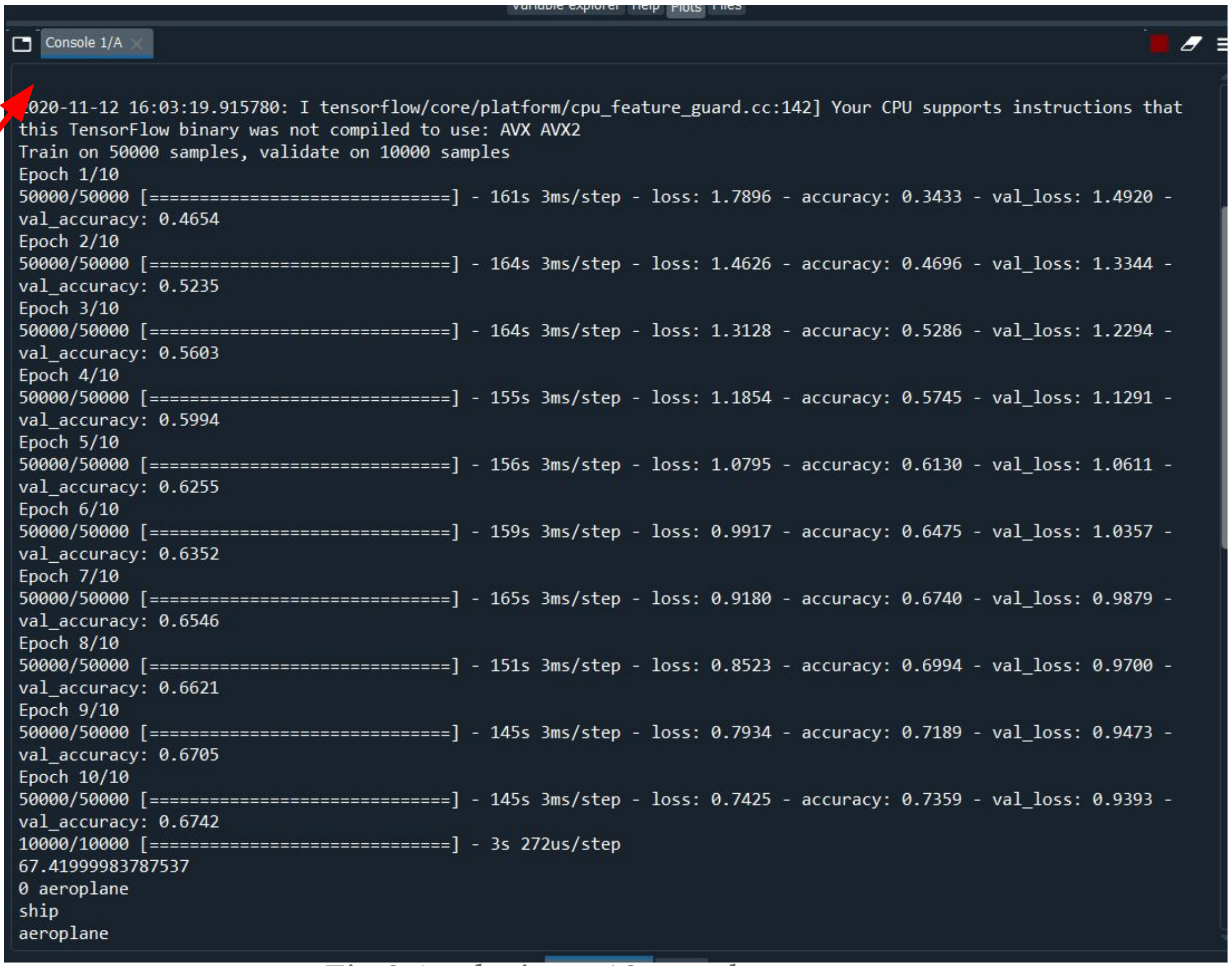
We will train the model with batch size= 32,epochs=10, and the total number of images we used to train are 50000.

```
model.fit(train_X,train_Y,
    validation_data=(test_X,test_Y),
    epochs=10,batch_size=32)
```

Trained Analysis for 10 Epochs

When the complete analysis is done for 10 epochs for this model.The parameters such as loss,accuracy,val_loss and val_accuracy has been analysed

This is the console in Python which gives us output.



```
020-11-12 16:03:19.915780: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that
this TensorFlow binary was not compiled to use: AVX AVX2
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 161s 3ms/step - loss: 1.7896 - accuracy: 0.3433 - val_loss: 1.4920 -
val_accuracy: 0.4654
Epoch 2/10
50000/50000 [==============================] - 164s 3ms/step - loss: 1.4626 - accuracy: 0.4696 - val_loss: 1.3344 -
val_accuracy: 0.5235
Epoch 3/10
50000/50000 [==============================] - 164s 3ms/step - loss: 1.3128 - accuracy: 0.5286 - val_loss: 1.2294 -
val_accuracy: 0.5603
Epoch 4/10
50000/50000 [==============================] - 155s 3ms/step - loss: 1.1854 - accuracy: 0.5745 - val_loss: 1.1291 -
val_accuracy: 0.5994
Epoch 5/10
50000/50000 [==============================] - 156s 3ms/step - loss: 1.0795 - accuracy: 0.6130 - val_loss: 1.0611 -
val_accuracy: 0.6255
Epoch 6/10
50000/50000 [==============================] - 159s 3ms/step - loss: 0.9917 - accuracy: 0.6475 - val_loss: 1.0357 -
val_accuracy: 0.6352
Epoch 7/10
50000/50000 [==============================] - 165s 3ms/step - loss: 0.9180 - accuracy: 0.6740 - val_loss: 0.9879 -
val_accuracy: 0.6546
Epoch 8/10
50000/50000 [==============================] - 151s 3ms/step - loss: 0.8523 - accuracy: 0.6994 - val_loss: 0.9700 -
val_accuracy: 0.6621
Epoch 9/10
50000/50000 [==============================] - 145s 3ms/step - loss: 0.7934 - accuracy: 0.7189 - val_loss: 0.9473 -
val_accuracy: 0.6705
Epoch 10/10
50000/50000 [==============================] - 145s 3ms/step - loss: 0.7425 - accuracy: 0.7359 - val_loss: 0.9393 -
val_accuracy: 0.6742
10000/10000 [==============================] - 3s 272us/step
67.41999983787537
0 aeroplane
ship
aeroplane
```

Fig.8 Analysis on 10 epochs

Calculate Accuracy and Save the model

Finally,we have trained the model and its time to test it and here we are also calculating the accuracy of this model and saving it using 'model.save( )'.

```
_,acc=model.evaluate(test_X,test_Y)
print(acc*100)
model.save("model1_cifar_10epoch.h5")
```

Make a dictionary to map the output classes

Dictionary in Python is an unordered collection of data values which is used to store data values like a map,Dog,Cat etc. which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key value is provided in the dictionary to make it more optimized.In this project the keys ranging from 0 to 9 and values are Aeroplane,Automobile, Bird,Cat,Deer,Dog,Frog,Horse,Ship,Truck.

Make a dictionary to map the output classes (Cont'd)

```
results={
   0:'aeroplane',
   1:'automobile',
   2:'bird',
   3:'cat',
   4:'deer',
   5:'dog',
   6:'frog',
   7:'horse',
   8:'ship',
   9:'truck'
}
from PIL import Image
import numpy as np
im=Image.open("aeroplane10.png")
# the input image is required to be in the shape of dataset, i.e (32,32,3)
 im=im.resize((32,32))
im=np.expand_dims(im,axis=0)
im=np.array(im)
pred=model.predict_classes([im])[0]
print(pred,results[pred])
```

Plot for Accuracy vs Epoch

Accuracy comparison between train data set and test data set

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(model.history.history['accuracy'],color="#E74C3C",marker='o')
plt.plot(model.history.history['val_accuracy'],color='#641E16',marker='h')
plt.title('Accuracy comparison between Validation and Train Data
set',fontsize=15)
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()
```

This plot is w.r.t
10 Epochs

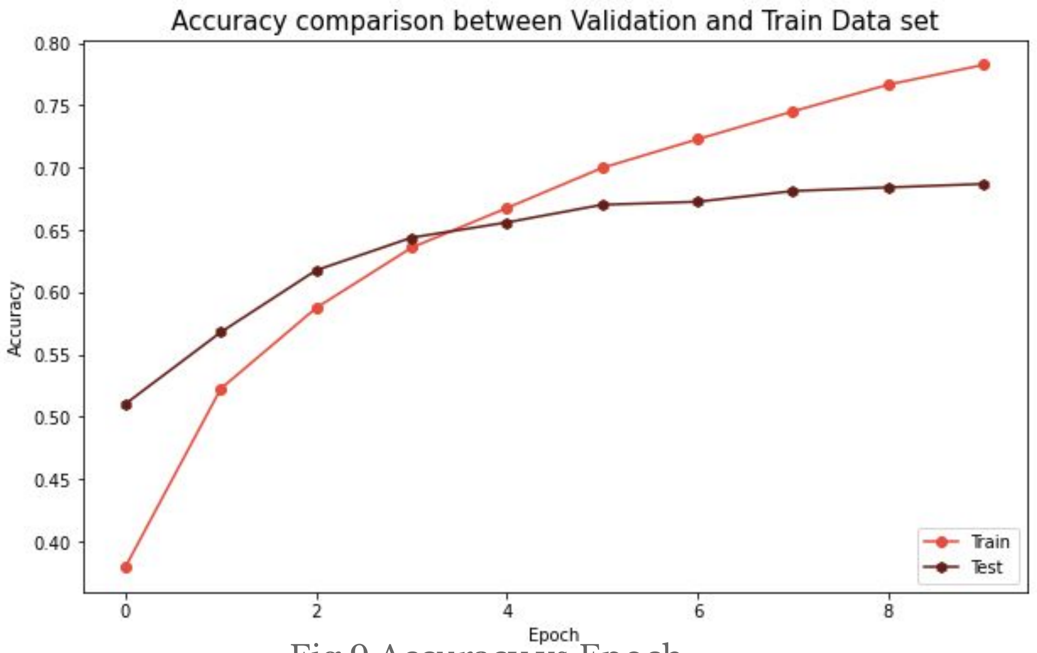


Fig.9 Accuracy vs Epoch

Plot for Loss vs Epoch

Loss comparison between train data set and test data set

```
plt.figure(figsize=(10,6))
plt.plot(model.history.history['loss'],color="Purple",marker='o')
plt.plot(model.history.history['val_loss'],color='Orange',marker='h')
plt.title('Loss comparison between Validation and Train Data
set',fontsize=15)
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='best')
plt.show()
```

This plot is w.r.t 10 Epochs

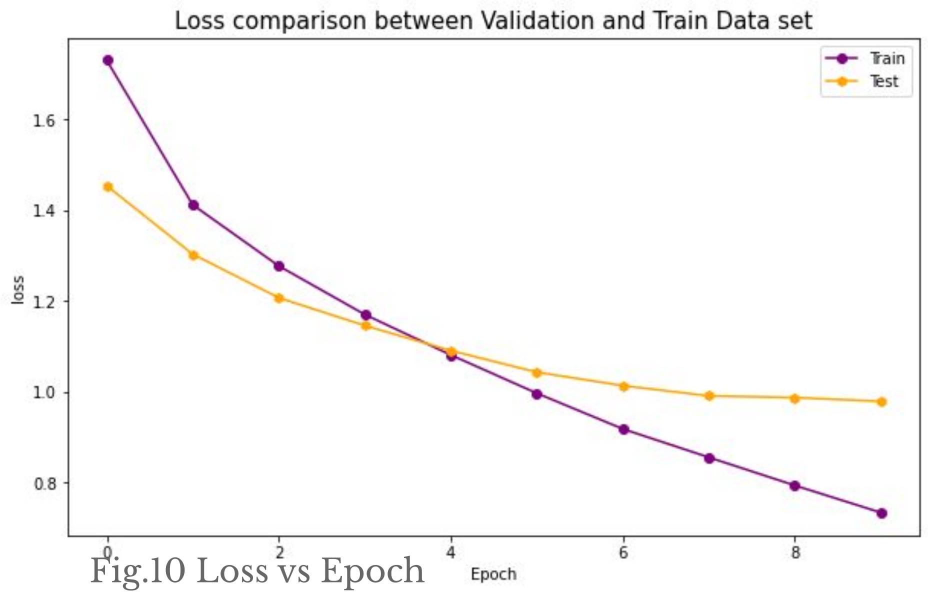


Fig.10 Loss vs Epoch

Hyperparameter Analysis

- The hyper parameters in the project are Learning Rate,Epochs and Batch Size.
- Analysis using Epochs as hyper parameter is done for 10,20 and 30 epochs respectively and observed the model loss,Validation loss,accuracy and its validation accuracy is observed and its waveforms are also been Plotted respectively.

S.No.	Loss	Accuracy	Val_loss	Val_accuracy	Epochs
1.	0.978	0.576	0.841	0.635	10
2.	0.719	0.703	0.885	0.647	20
3.	0.642	0.763	0.922	0.734	30

Hyperparameter Analysis (Cont'd)

Plots when **Epochs=20**,we can observe that the loss of the model has been decreased and the accuracy of the model has been increased which produces a more significant and better output.

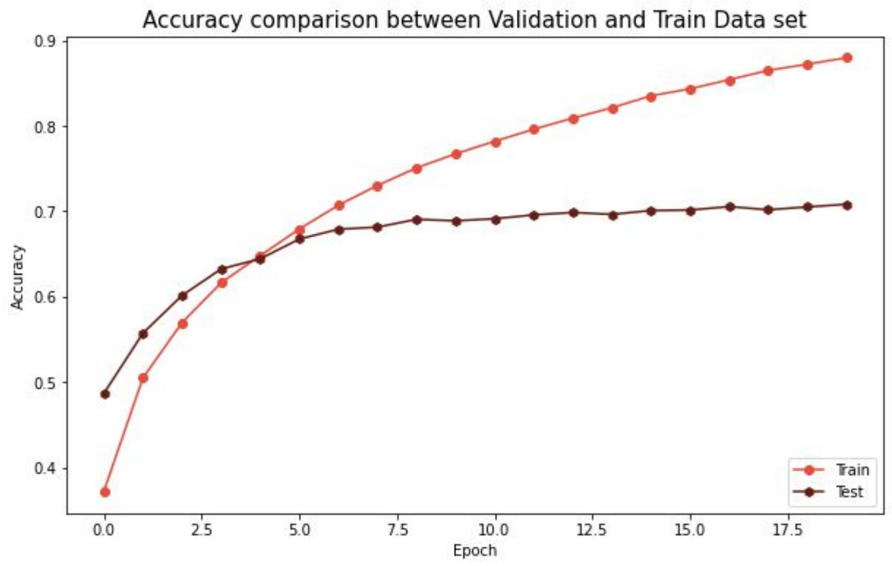


Fig.11 Accuracy vs Epoch (20)

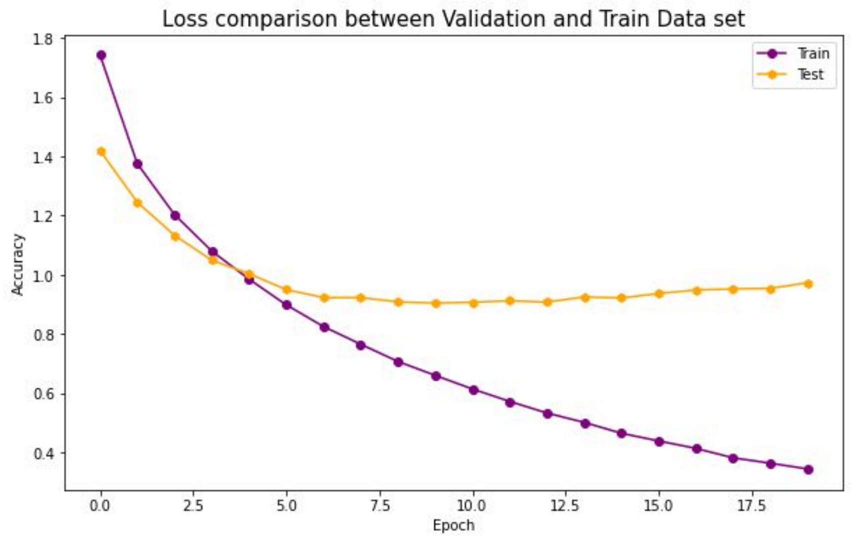


Fig.12 Loss vs Epoch (20)

Hyperparameter Analysis (Cont'd)

Plots when **Epochs=30**,we can observe that the loss of the model has been further decreased and the accuracy of the model has been increased more when compared with 20 epochs which produces a more significant and efficient output.

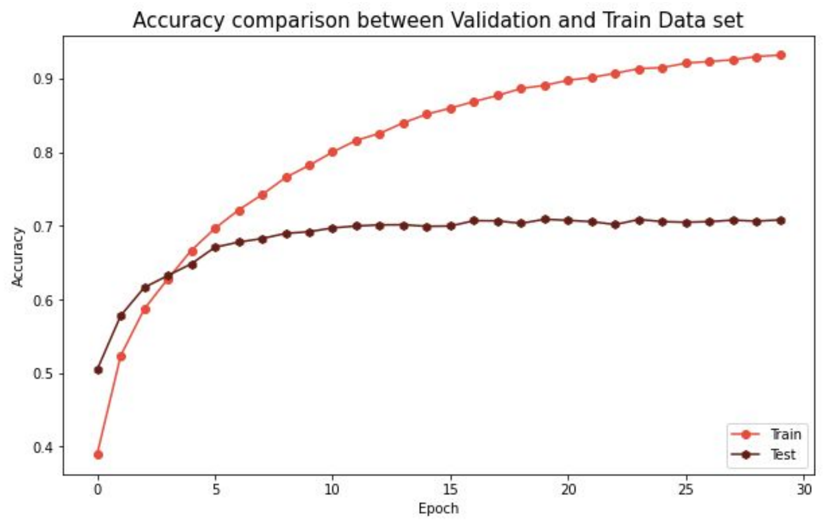


Fig.13 Accuracy vs Epoch (30)

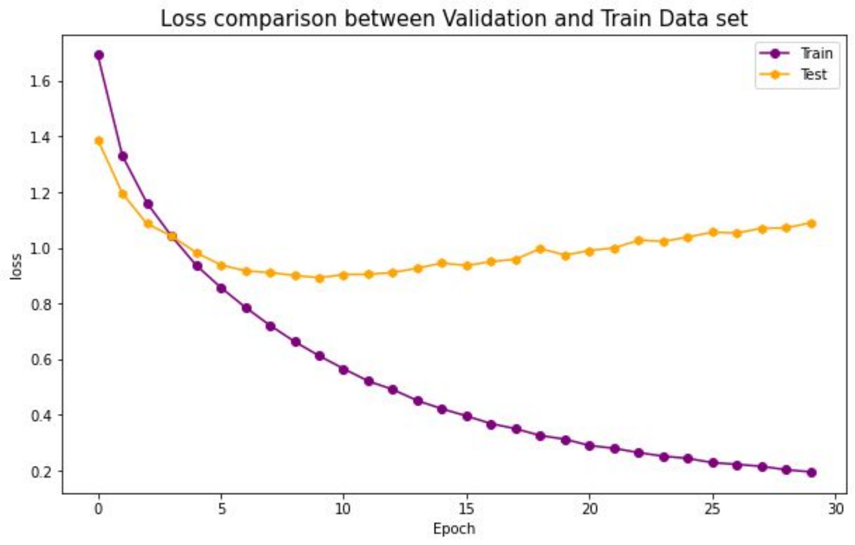


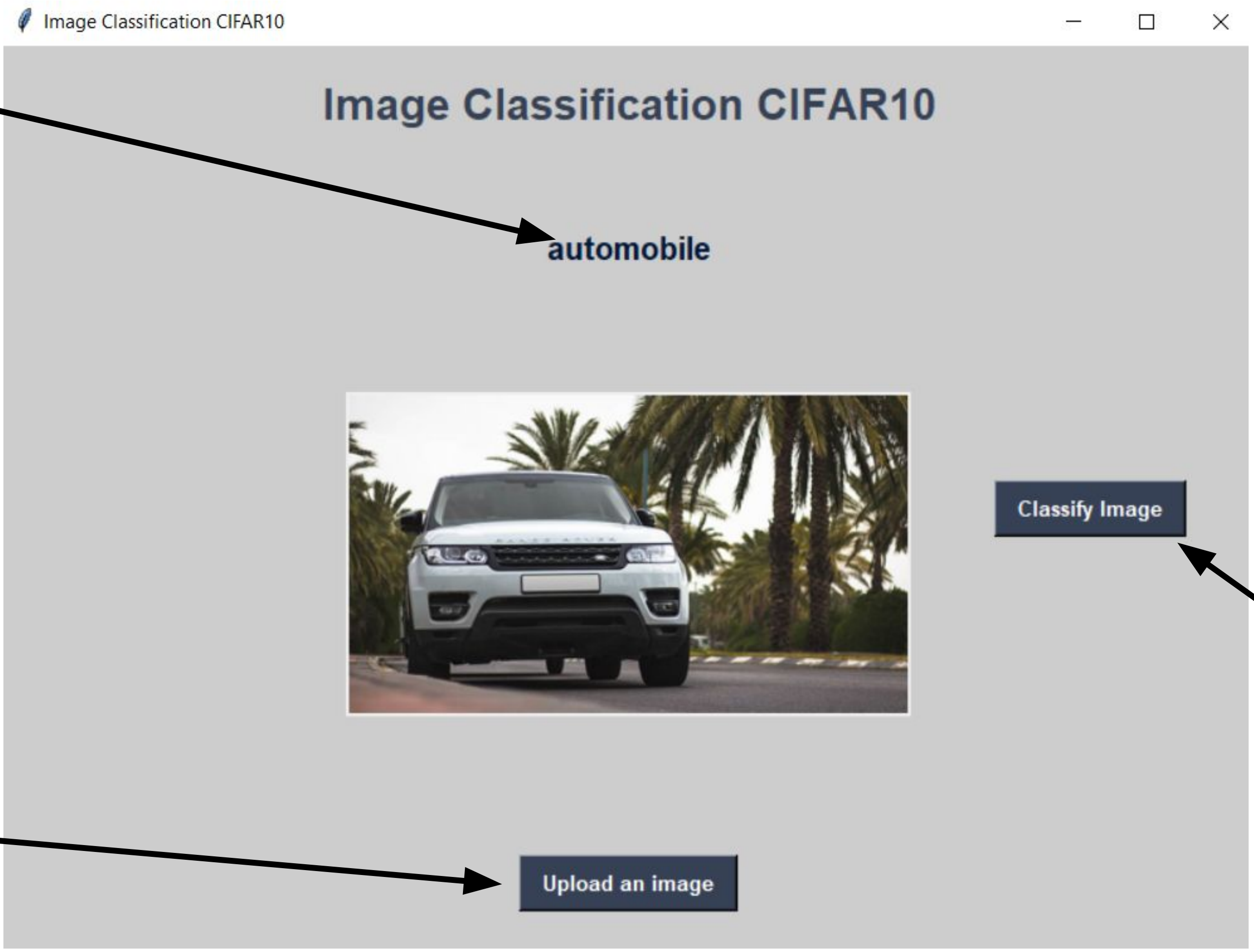
Fig.14 Accuracy vs Epoch (30)

GUI-tkinter

Tkinter is easiest and fastest way to develop a graphical user interface which provides user an easy-to-use interface with immediate feedback

This output displays us to the class the image belongs

Upload any image from your local system to know to what class the image belongs to



This option is used to classify the image given by user

Fig.15 GUI

GUI-Tkinter coding

```
import Tkinter

top = Tkinter.Tk()
top.mainloop()
```

This basically creates a window as shown below and next stop would be load the model which we have done above using Keras for the image classification.

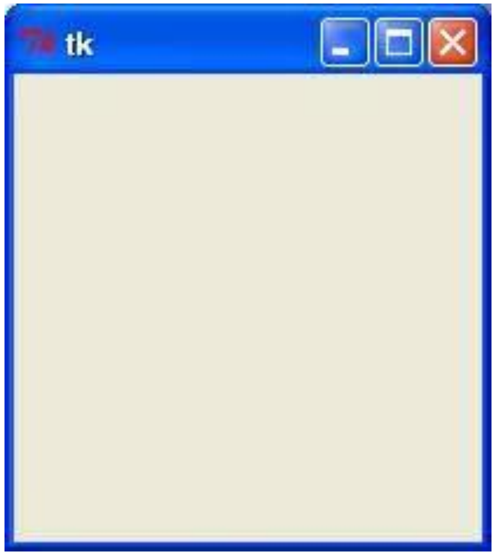


Fig.16 Open Block Of GUI

Source:https://www.tutorialspoint.com/python/python_gui_programming.htm

Image Classification Using GUI-Code

```python
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy

from keras.models import load_model
model = load_model('model1_cifar_10epoch.h5')

classes = {
    0:'aeroplane',
    1:'automobile',
    2:'bird',
    3:'cat',
    4:'deer',
    5:'dog',
    6:'frog',
    7:'horse',
    8:'ship',
    9:'truck'
}
```

Image Classification Using GUI-Code (Cont'd)

Next step is to initialize the size of the window and define its background colour and title of the window with font size and style will be also defined using the code below:

```
top=tk.Tk()
top.geometry('800x600')
top.title('Image Classification CIFAR10')
top.configure(background='#CDCDCD')
label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)
```

Now when we are opening the image from our local system that needs to be classified to which class it belongs to the file path will automatically be appended to the code and opens the image successfully everytime we perform task.

```
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((32,32))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    pred = model.predict_classes([image])[0]
    sign = classes[pred]
    print(sign)
    label.configure(foreground='#011638', text=sign)
```

Image Classification Using GUI-Code (Cont'd)

The below code here is used to define the font style,font size and its color for the classify button and we are also defining the background and foreground colors also.

```
def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",
   command=lambda: classify(file_path),padx=10,pady=5)
classify_b.configure(background='#364156', foreground='white',
font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46
```

Next,We used the Try and Except in python to perform uploading an image in this project so that when the file path is corrupted or if the image is not in the specified argument the current process will be passed.The try and except basically works on the algorithm as shown below:

- First try clause is executed i.e. the code between try and except clause.
- If there is no exception, then only try clause will run, except clause is finished.
- If any exception occured, try clause will be skipped and except clause will run.
- If any exception occurs, but the except clause within the code doesn't handle it, it is passed on to the outer try statements. If the exception left unhandled, then the execution stops.
- A try statement can have more than one except clause.

Image Classification Using GUI-Code (Cont'd)

```
def upload_image():
    try:
      file_path=filedialog.askopenfilename()
     uploaded=Image.open(file_path)
     uploaded.thumbnail(((top.winfo_width()/2.25),
    (top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass
upload=Button(top,text="Upload an image",command=upload_image,
  padx=10,pady=5)

upload.configure(background='#364156', foreground='white',
    font=('arial',10,'bold'))
upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Image Classification CIFAR10",pady=20,
font=('arial',20,'bold'))

heading.configure(background='#CDCDCD',foreground='#364156')
heading.pack()
top.mainloop()
```

Summary

CNNs are one of the most commonly used Neural Networks for Image Classification and Recognition.This is the technique that we are going to use on the CIFAR-10 dataset to classify images in one of the 10 categories and we also used GUI to classify the images.

CNN has 4 main steps:

1.**Convolution**
2.**Activation Function**
3.**Pooling**
4.**Fully Connected Layer - Dense Layer**

We finally imported all the required layers for the network then trained and tested the model using CIFAR-10 and calculated its accuracy,Loss and classified the images using the GUI.