CS634-104 Data Mining Final Term Project

Name :- Rajesh Gnanasekaran (rg48@njit.edu)

Algorithm :- CNN (Convolution Neural Network)

The convolutional neural network (CNN) is a class of deep learning neural networks.
They're most commonly used to analyse visual imagery and are frequently working behind the scenes in image classification. They can be found at the core of everything from Facebook's photo tagging to self-driving cars. They're working hard behind the scenes in everything from healthcare to security.

Image classification is the process of taking an input (like a picture) and outputting a class (like "car") or a probability that the input is a particular class ("there's a 90% probability that this input is a car").

CNNs actually use very little preprocessing. This means that they can learn the filters that have to be hand-made in other algorithms. CNNs can be used in tons of applications from image and video recognition, image classification, and recommender systems to natural language processing and medical image analysis.

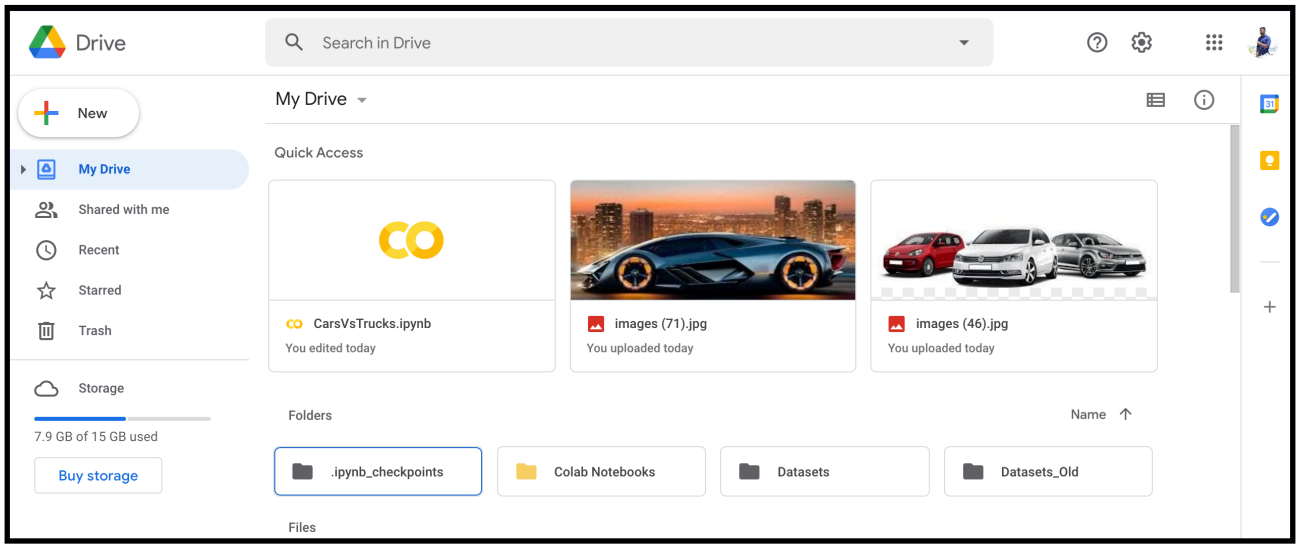
So ill be using the Dataset Cars and Trucks where we have to use image classification to distinguish cars vs trucks images.

✳ Ill be using Google's free deep learning cloud notebook
Colab notebook (https://colab.research.google.com/) for this classification.

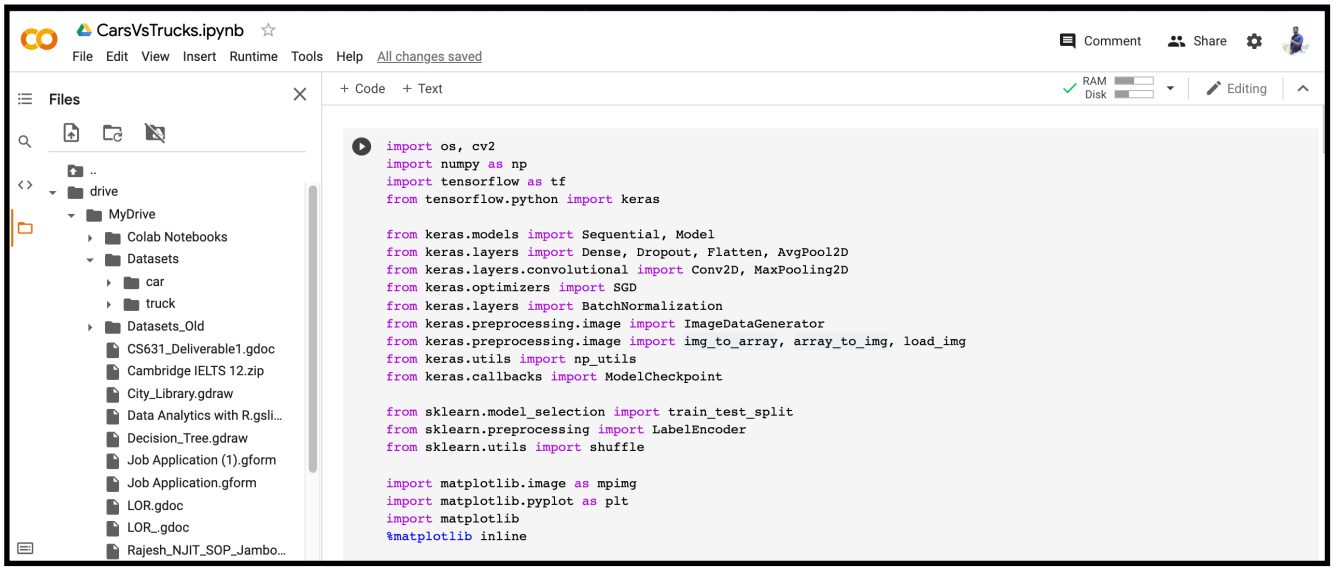
✳ The dataset ill be using cars vs trucks dataset
Cars dataset from below
https://ai.stanford.edu/~jkrause/cars/car_dataset.html
And Trucks dataset from kaggle

Steps :-

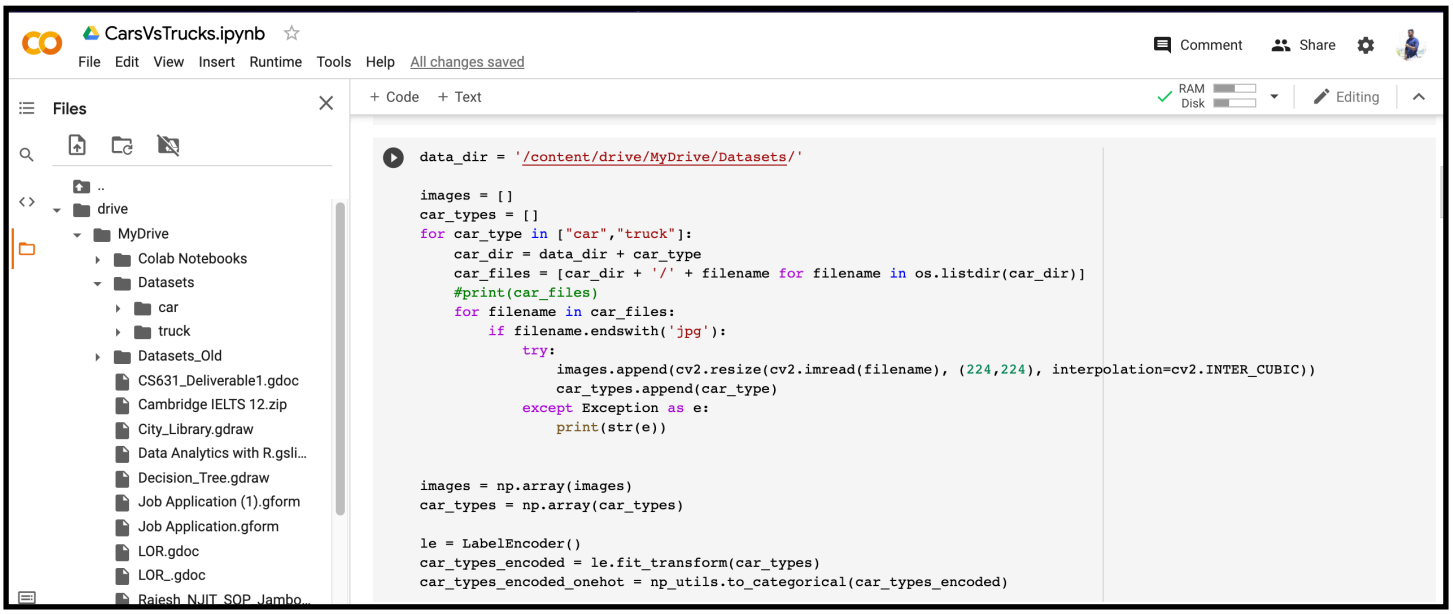
First I have added the dataset in my Google drive so that I can connect the google drive with Google Colab notebook.



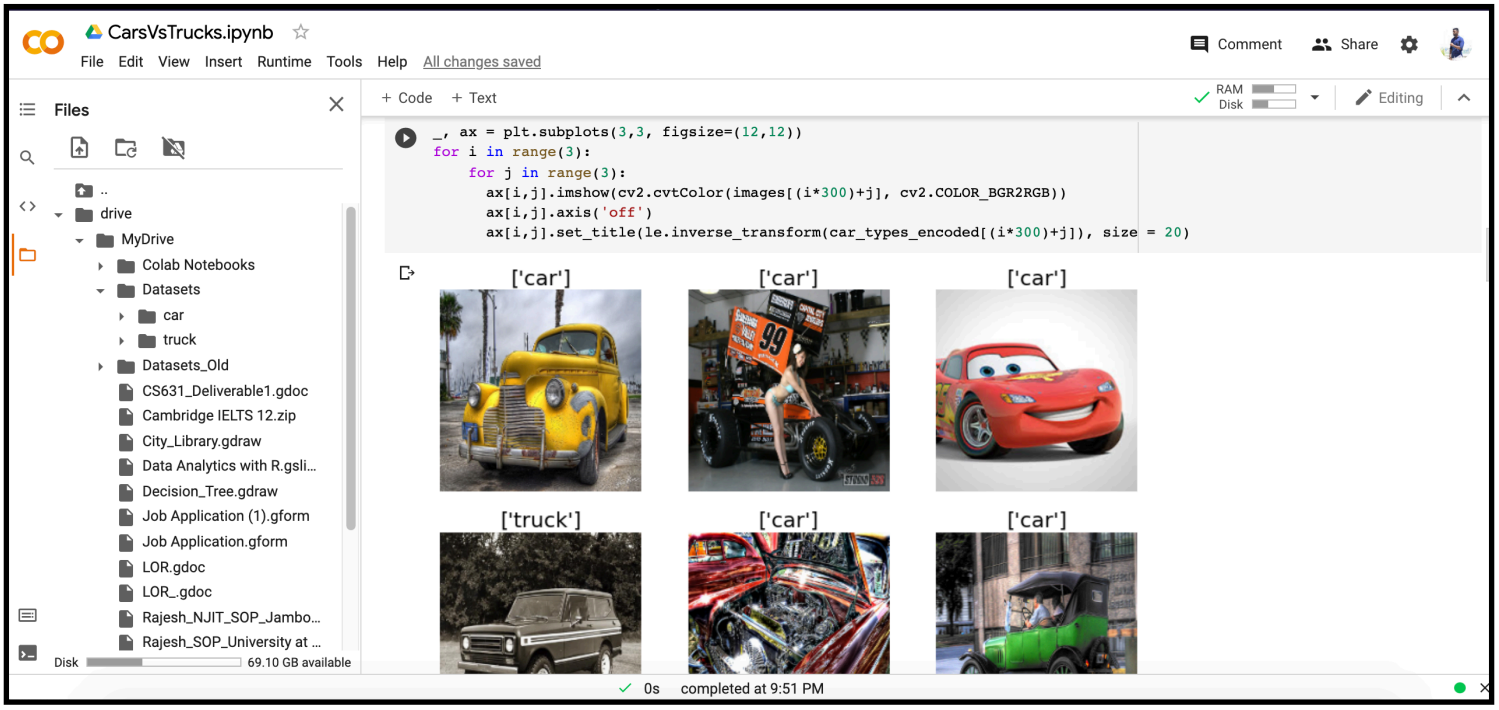
Created a new notebook in google Colab and import all the library files like TensorFlow , Keras , matplotkib …etc that is needed for image classification .



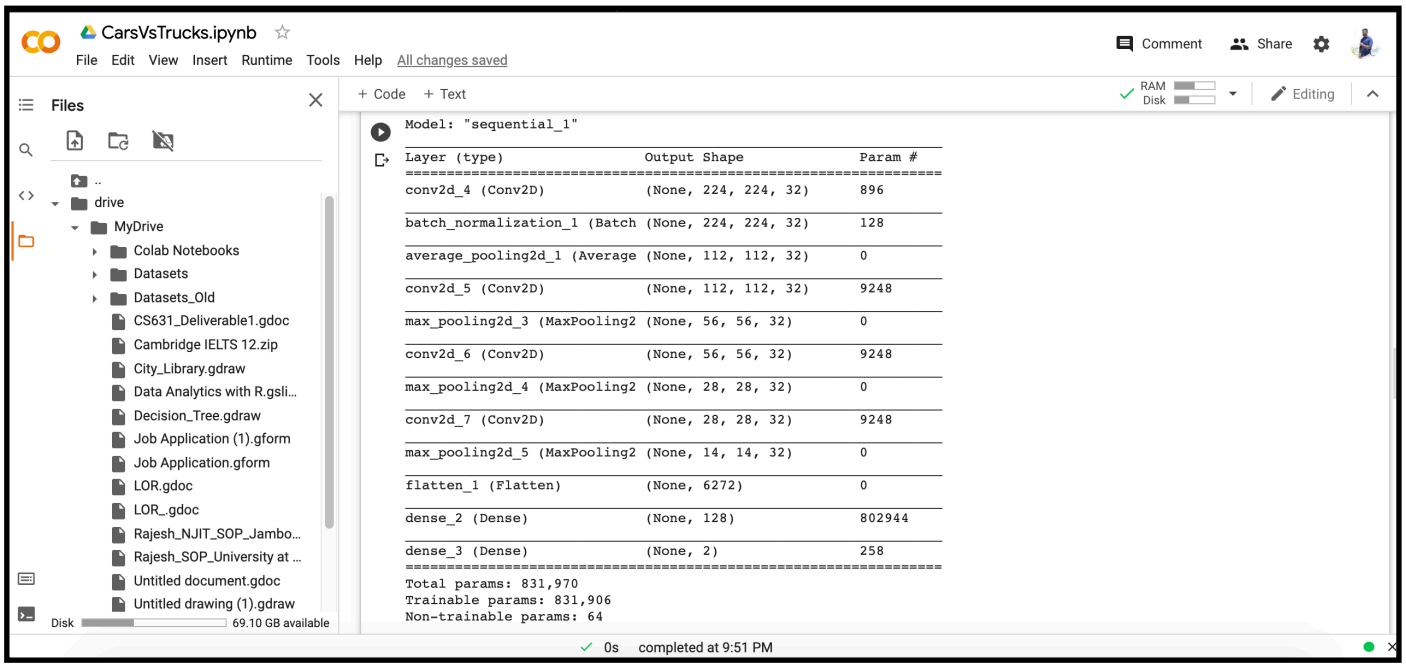
Connected the Colab notebook with Google drive so that we can access the datasets which I have uploaded in google drive.



Then displayed the cars and truck plot them in a set.



Encoded and resize the car and truck images , flatten the images , convert into 2D image shape.



Stored the car images in train data , also stored the truck images in respective train data .

Then the main process comes to calculate the accuracy of the image classified , I have Set the epoch value to 1 and batch size is 4 since it takes time to process the images.

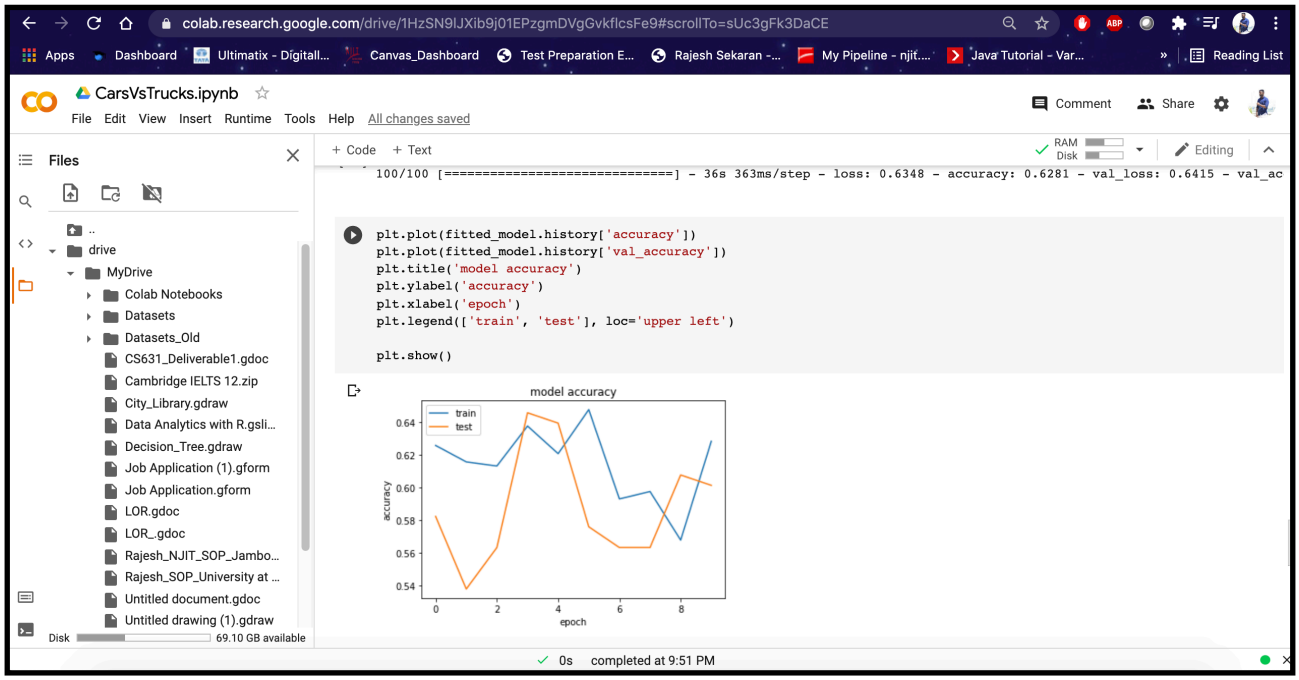
**steps_per_epoch: Integer. Total number of steps (batches of samples) to yield from generator before declaring one epoch finished and starting the next epoch. It should typically be equal to ceil(num_samples / batch_size).

For example, if you have 100 training samples, then num_samples = 100, or the number of rows of x_train is 100.
You can specify your own batch size. In this case, say batch_size = 20. As a result, you can set your steps_per_epoch = 100/20 = 5 because in this way you can make use of the complete training data for each epoch.

The above way is good way to process large data , we can split large data in small steps per epoch and run accordingly to the needed batch size.

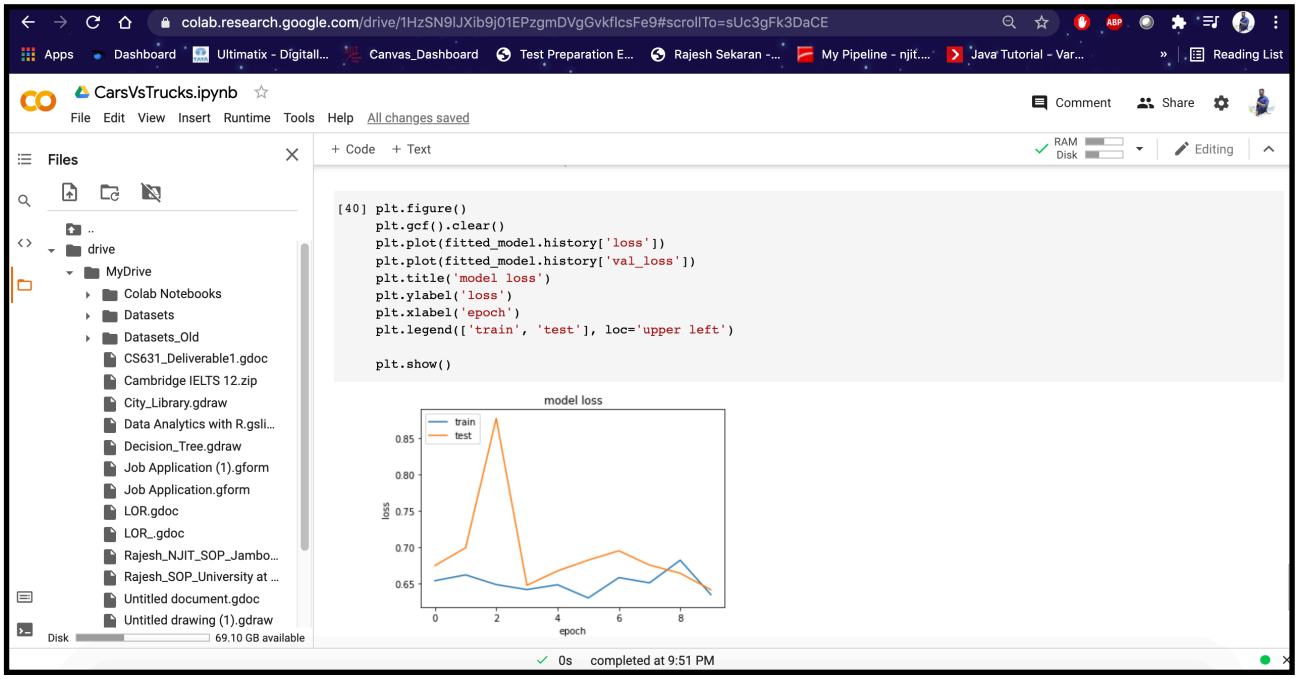
Basically epoch means the number of times the process will iterate to get the result so that its easy to analyse the data and predict how good the image are being classified.

Then we plot the model accuracy of the training data and test data accordingly.

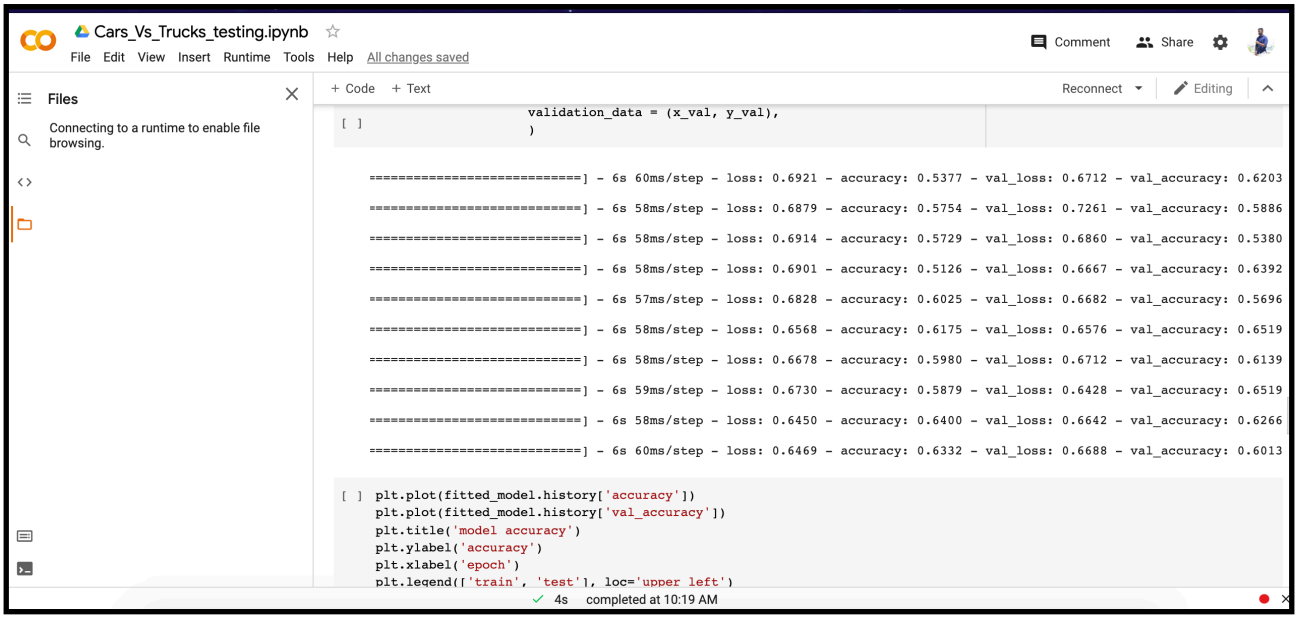


From the above graph we can see that the train data has better result than the test data , but even though its not completely accurate but the results are good.

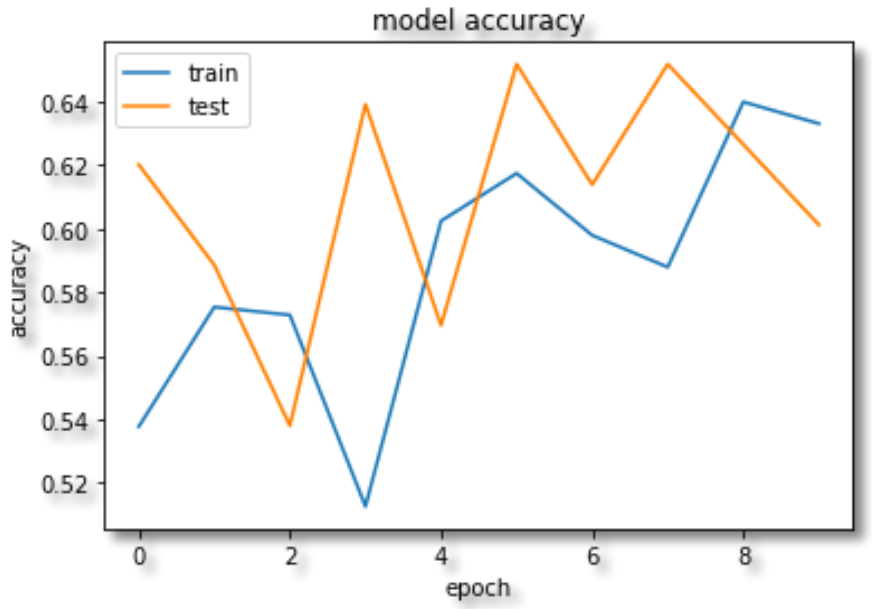
I also calculated the model loss accuracy as shown in below the train data loss has a big spike and then decreased , whereas the training data has stable loss.



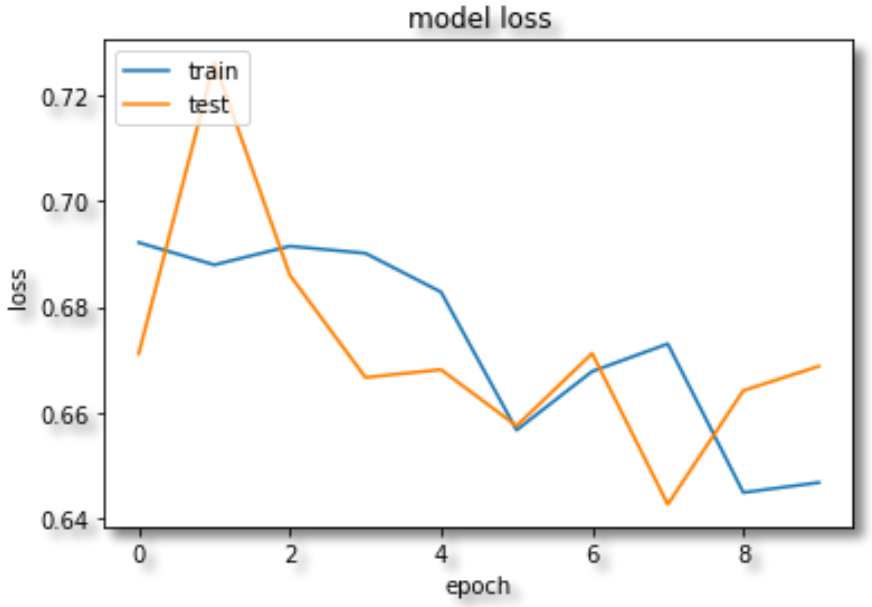
Testing another set of dataset :-



Model Accuracy :



Model Loss :



==========

Source Code :-

```
import os, cv2
import numpy as np
import tensorflow as tf
from tensorflow.python import keras

from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Flatten, AvgPool2D
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.optimizers import SGD
from keras.layers import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing.image import img_to_array, array_to_img,
load_img
from keras.utils import np_utils
from keras.callbacks import ModelCheckpoint

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import matplotlib
%matplotlib inline

data_dir = '/content/drive/MyDrive/Datasets/'

images = []
car_types = []
for car_type in ["car","truck"]:
    car_dir = data_dir + car_type
    car_files = [car_dir + '/' + filename for filename in
os.listdir(car_dir)]
    #print(car_files)
    for filename in car_files:
```

```
        if filename.endswith('jpg'):
            try:
                images.append(cv2.resize(cv2.imread(filename),
(224,224), interpolation=cv2.INTER_CUBIC))
                car_types.append(car_type)
            except Exception as e:
                print(str(e))


images = np.array(images)
car_types = np.array(car_types)

le = LabelEncoder()
car_types_encoded = le.fit_transform(car_types)
car_types_encoded_onehot =
np_utils.to_categorical(car_types_encoded)

_, ax = plt.subplots(3,3, figsize=(12,12))
for i in range(3):
    for j in range(3):
      ax[i,j].imshow(cv2.cvtColor(images[(i*300)+j],
cv2.COLOR_BGR2RGB))
      ax[i,j].axis('off')

ax[i,j].set_title(le.inverse_transform(car_types_encoded[(i*300)+j
]), size = 20)

images, car_types, car_types_encoded = shuffle(images, car_types,
car_types_encoded)
car_types_encoded.resize((images.shape[0],1))
print(car_types_encoded.shape)

model = Sequential()

model.add(Conv2D(32, (3, 3),input_shape=(224, 224, 3),strides =
(1,1),  padding = 'same',kernel_initializer='he_normal',
activation='relu'))
model.add(BatchNormalization())
```

```
model.add(AvgPool2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3),strides = (1,1),  padding = 'same',
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3),strides = (1,1),  padding = 'same',
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3),strides = (1,1),  padding = 'same',
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(units=128, activation='relu',
kernel_initializer='normal'))
model.add(Dense(units=2, activation='sigmoid',
kernel_initializer='normal'))

model.compile(optimizer='adam',loss='binary_crossentropy',metrics=
['accuracy'])
model.summary()

car_types_encoded = car_types_encoded.reshape((images.shape[0],1))
car_types_2class = np.zeros((images.shape[0],2))
for i in range(images.shape[0]):
    if car_types_encoded[i][0] == 0:
        car_types_2class[i][0] = 1
    else:
        car_types_2class[i][1] = 1
#print(car_types_2class[1:100,:])
x_train, x_val, y_train, y_val = train_test_split(images,
car_types_2class, test_size=0.2, random_state=0)
x_train = x_train / 255
x_val = x_val / 255

# set train Generator
```

```
datagen =
ImageDataGenerator(rotation_range=30,width_shift_range=0.2,height_
shift_range=0.2,horizontal_flip=True)
datagen.fit(x_train)


 def parse_args(args):
  group = parser.add_mutually_exclusive_group()
  parser.add_argument('--steps',            help='Number of steps
per epoch.', type=int, default=10000)

# trainning process
nb_epoch = 1
batch_size = 4

fitted_model = model.fit_generator(datagen.flow(x_train, y_train,
batch_size=batch_size),
                   # steps_per_epoch = x_train.shape[0],
                   #epochs=nb_epoch,
                   steps_per_epoch = 100,
                   epochs=10,
                    validation_data = (x_val, y_val),
                    )


plt.plot(fitted_model.history['accuracy'])
plt.plot(fitted_model.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

plt.show()

plt.figure()
plt.gcf().clear()
plt.plot(fitted_model.history['loss'])
plt.plot(fitted_model.history['val_loss'])
plt.title('model loss')
```

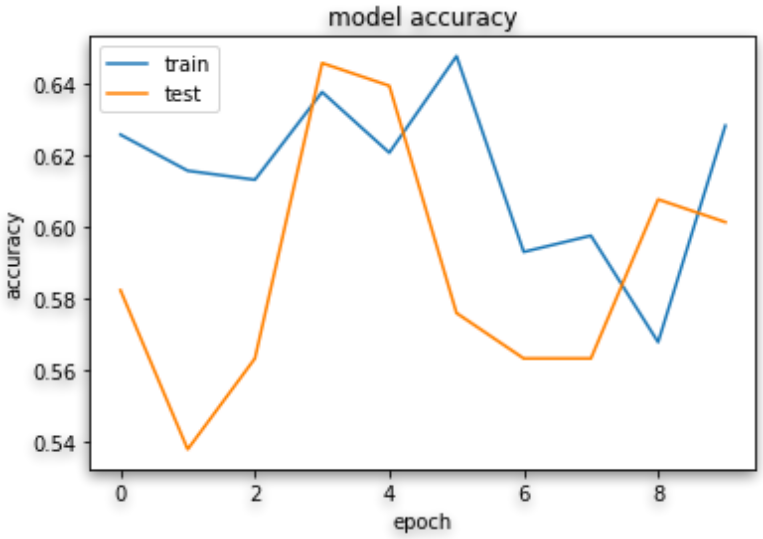
```
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')

plt.show()
```

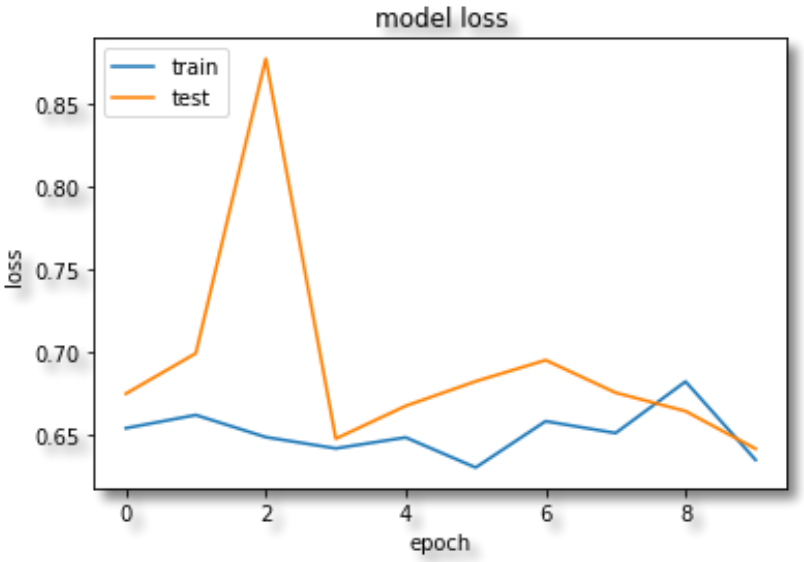
==========

Output :-

```
Epoch 1/10
100/100 [==============================] - 37s 365ms/step -
loss: 0.6539 - accuracy: 0.6256 - val_loss: 0.6748 -
val_accuracy: 0.5823
Epoch 2/10
100/100 [==============================] - 36s 361ms/step -
loss: 0.6619 - accuracy: 0.6156 - val_loss: 0.6992 -
val_accuracy: 0.5380
Epoch 3/10
100/100 [==============================] - 36s 361ms/step -
loss: 0.6485 - accuracy: 0.6131 - val_loss: 0.8773 -
val_accuracy: 0.5633
Epoch 4/10
100/100 [==============================] - 36s 364ms/step -
loss: 0.6417 - accuracy: 0.6375 - val_loss: 0.6475 -
val_accuracy: 0.6456
Epoch 5/10
100/100 [==============================] - 36s 362ms/step -
loss: 0.6483 - accuracy: 0.6206 - val_loss: 0.6674 -
val_accuracy: 0.6392
Epoch 6/10
100/100 [==============================] - 36s 364ms/step -
loss: 0.6301 - accuracy: 0.6475 - val_loss: 0.6822 -
val_accuracy: 0.5759
Epoch 7/10
100/100 [==============================] - 36s 362ms/step -
loss: 0.6581 - accuracy: 0.5930 - val_loss: 0.6950 -
val_accuracy: 0.5633
Epoch 8/10
100/100 [==============================] - 36s 365ms/step -
loss: 0.6509 - accuracy: 0.5975 - val_loss: 0.6754 -
val_accuracy: 0.5633
```

```
Epoch 9/10
100/100 [==============================] - 36s 363ms/step -
loss: 0.6821 - accuracy: 0.5678 - val_loss: 0.6642 -
val_accuracy: 0.6076
Epoch 10/10
100/100 [==============================] - 36s 363ms/step -
loss: 0.6348 - accuracy: 0.6281 - val_loss: 0.6415 -
val_accuracy: 0.6013
```





Challenges Faced :-

The main challenge I faced , when in the training process to calculate the accuracy

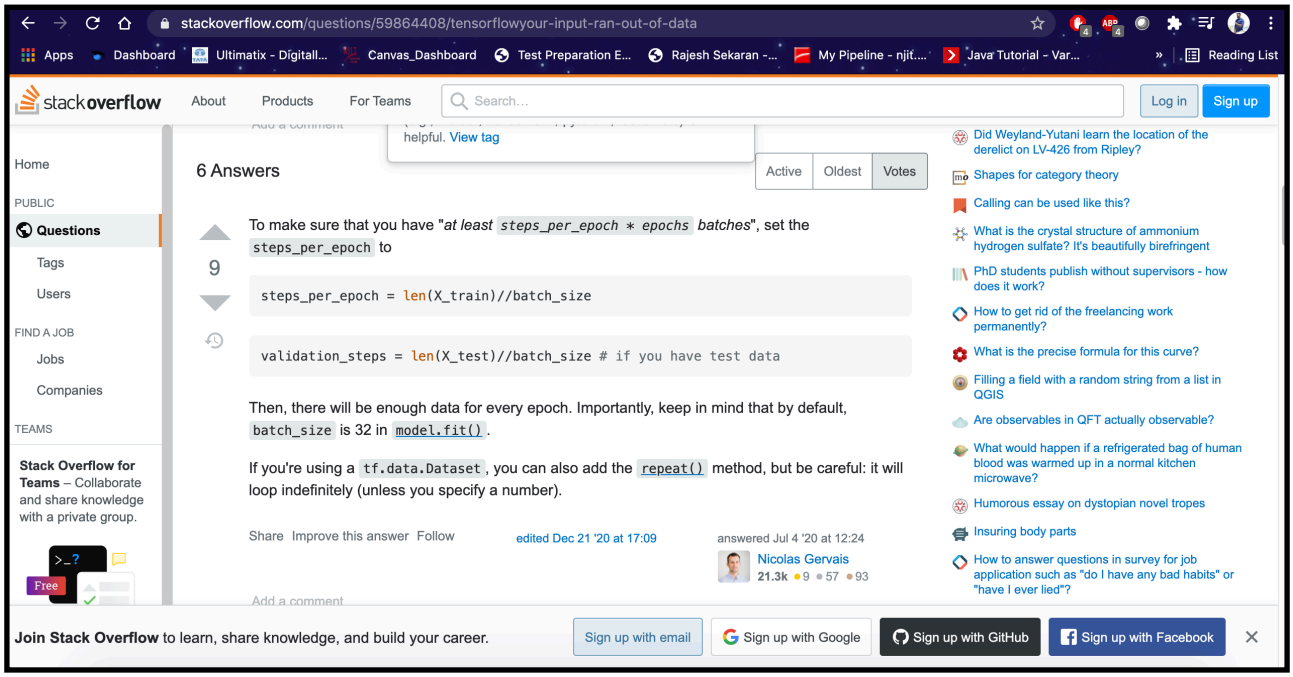
What Is the Difference Between Batch and Epoch?

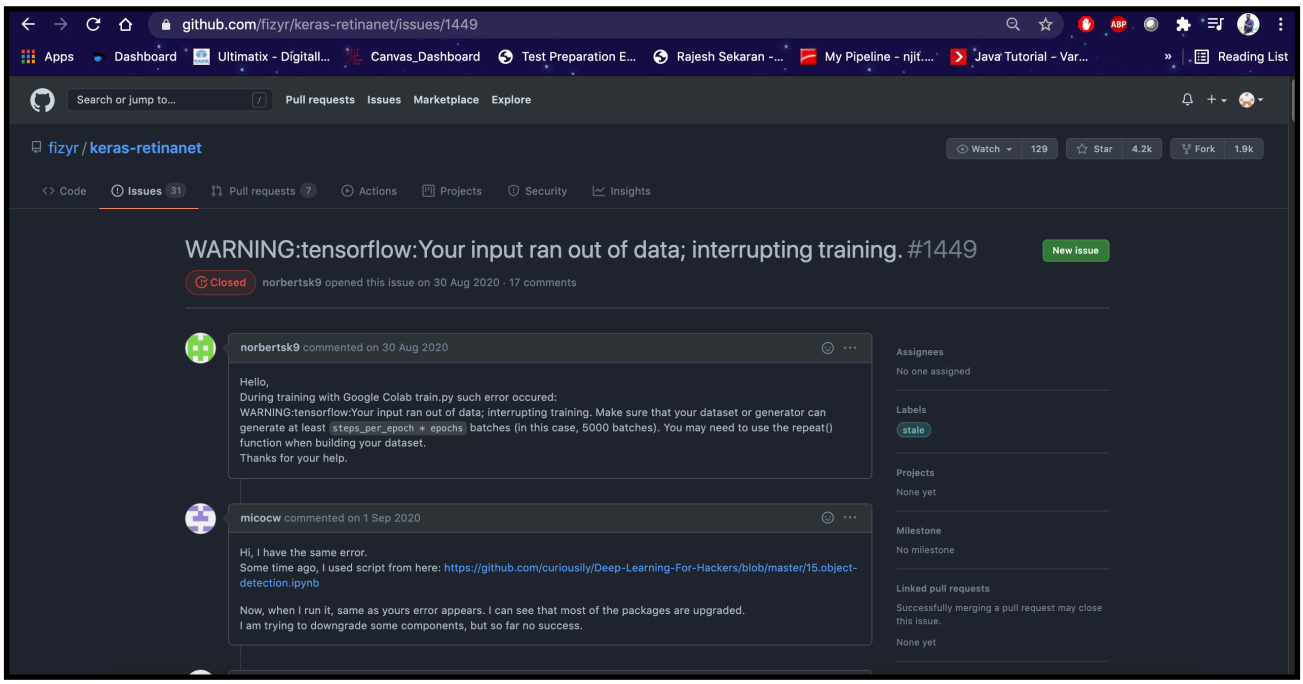
The batch size is a number of samples processed before the model is updated.
The number of epochs is the number of complete passes through the training dataset.
The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset.

So when in my dataset I was keeping the epoch time as 20 and batch size 4 , also the steps per epoch = x_train.shape[0]

So I was facing the error of " WARNING :tensorflow:your input ran out of data; interrupting training. make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches (in this case, 6300 batches). you may need to use the repeat() function when building your dataset."

When I googled it , many other coders where also facing this issue.





So I checked with one of the examples and tried to understood

Assuming you have a dataset with 200 samples (rows of data) and you choose a batch size of 5 and 1,000 epochs.

= > This means that the dataset will be divided into 40 batches, each with five samples. The model weights will be updated after each batch of five samples. This also means that one epoch will involve 40 batches or 40 updates to the model.

= > With 1,000 epochs, the model will be exposed to or pass through the whole dataset 1,000 times. That is a total of 40,000 batches during the entire training process.

So I finally changed accordingly the number of epochs and the batch size , after which I was able to execute the program properly and was able to classify the images.

GitHub Link :- https://github.com/Rajesh007x/CNN_DataMining

Referral Links :- https://www.tensorflow.org/tutorials/keras/regression
https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/classification.ipynb
https://www.tensorflow.org/tutorials/keras/classification