

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

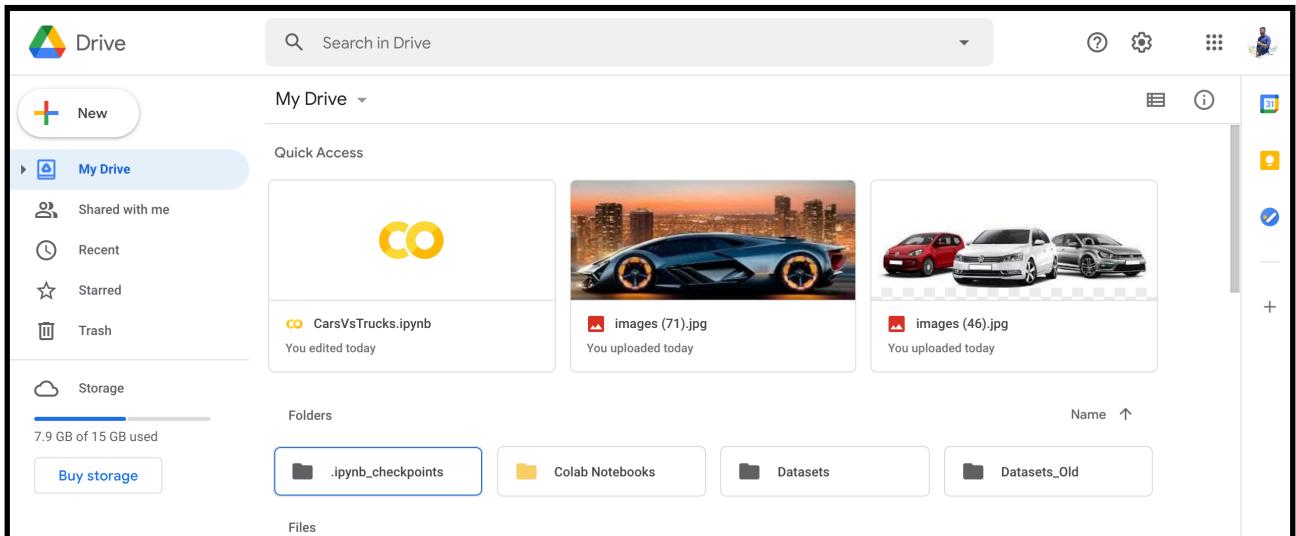
Cars dataset from below

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 , matplotlib ...etc that is needed for image classification .

A screenshot of a Google Colab notebook titled 'CarsVsTrucks.ipynb'. The code cell contains the following Python 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
```

The sidebar on the left shows the file structure of the Google Drive, including 'MyDrive' with 'Datasets' and 'Datasets_Old' folders containing various files like 'CS631_Deliverable1.gdoc', 'Cambridge IELTS 12.zip', 'City_Library.gdraw', etc.

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



The screenshot shows the Google Colab interface with a notebook titled "CarsVsTrucks.ipynb". The left sidebar displays a file tree with a "MyDrive" folder containing various files and subfolders like "Colab Notebooks" and "Datasets". The main area contains Python code for loading images and their types from a directory in Google Drive:

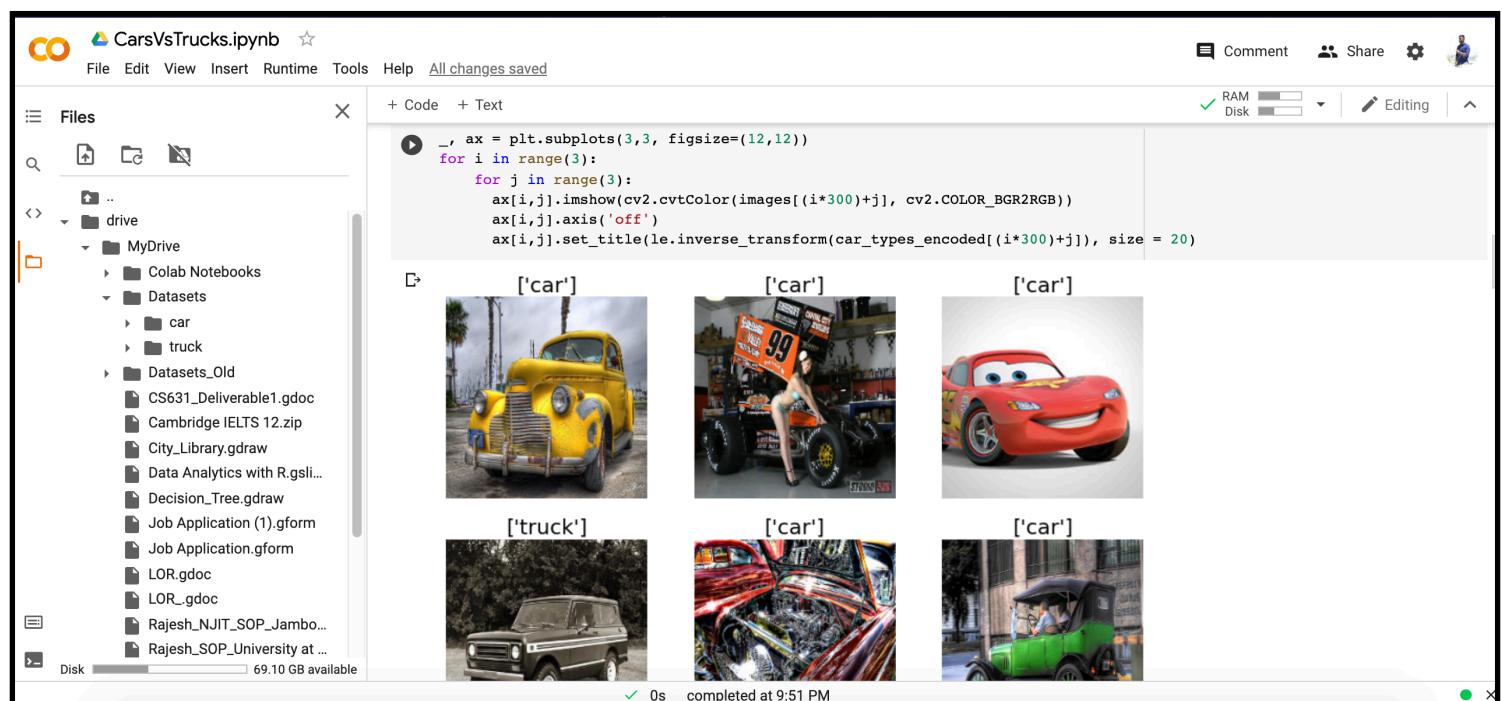
```
data_dir = '/content/drive/MyDrive/Datasets/'

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)
```

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



The screenshot shows the Google Colab interface with the same notebook. The code has been run, and the output is a 3x3 grid of images. Each image is labeled with its category: three images are labeled '[car]' and one is labeled '[truck]'. The images include a yellow classic car, a person sitting on a black truck, a red Lightning McQueen toy car, and a dark SUV.

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

The screenshot shows a Google Colab interface with a code cell containing Python code for building a neural network. The code defines a Sequential model with several layers: Conv2D, BatchNormalization, AvgPool2D, MaxPooling2D, Conv2D again, Flatten, and Dense layers. The model is compiled with Adam optimizer, binary_crossentropy loss, and accuracy metric. A summary of the model is displayed below the code, showing the layers, their types, output shapes, and parameter counts. The total parameters are listed as 831,970.

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

Model: "sequential_1"

```

| Layer (type) | Output Shape | Param # |
|-------------------|----------------------|---------|
| conv2d_4 (Conv2D) | (None, 224, 224, 32) | 896 |

✓ 0s completed at 9:51 PM

This screenshot shows the same Google Colab session with a more detailed summary of the model's layers. The summary table includes additional layers such as batch normalization, average pooling, and max pooling, along with their respective output shapes and parameter counts. The total number of parameters is explicitly stated as 831,970.

| Layer (type) | Output Shape | Param # |
|---|----------------------|---------|
| conv2d_4 (Conv2D) | (None, 224, 224, 32) | 896 |
| batch_normalization_1 (Batch Normalization) | (None, 224, 224, 32) | 128 |
| average_pooling2d_1 (Average Pooling2D) | (None, 112, 112, 32) | 0 |
| conv2d_5 (Conv2D) | (None, 112, 112, 32) | 9248 |
| max_pooling2d_3 (MaxPooling2D) | (None, 56, 56, 32) | 0 |
| conv2d_6 (Conv2D) | (None, 56, 56, 32) | 9248 |
| max_pooling2d_4 (MaxPooling2D) | (None, 28, 28, 32) | 0 |
| conv2d_7 (Conv2D) | (None, 28, 28, 32) | 9248 |
| max_pooling2d_5 (MaxPooling2D) | (None, 14, 14, 32) | 0 |
| flatten_1 (Flatten) | (None, 6272) | 0 |
| dense_2 (Dense) | (None, 128) | 802944 |
| dense_3 (Dense) | (None, 2) | 258 |

Total params: 831,970
Trainable params: 831,906
Non-trainable params: 64

✓ 0s completed at 9:51 PM

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

```
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
#rint(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)

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

[38] # training process
nb_epoch = 1
batch_size = 4
```

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.

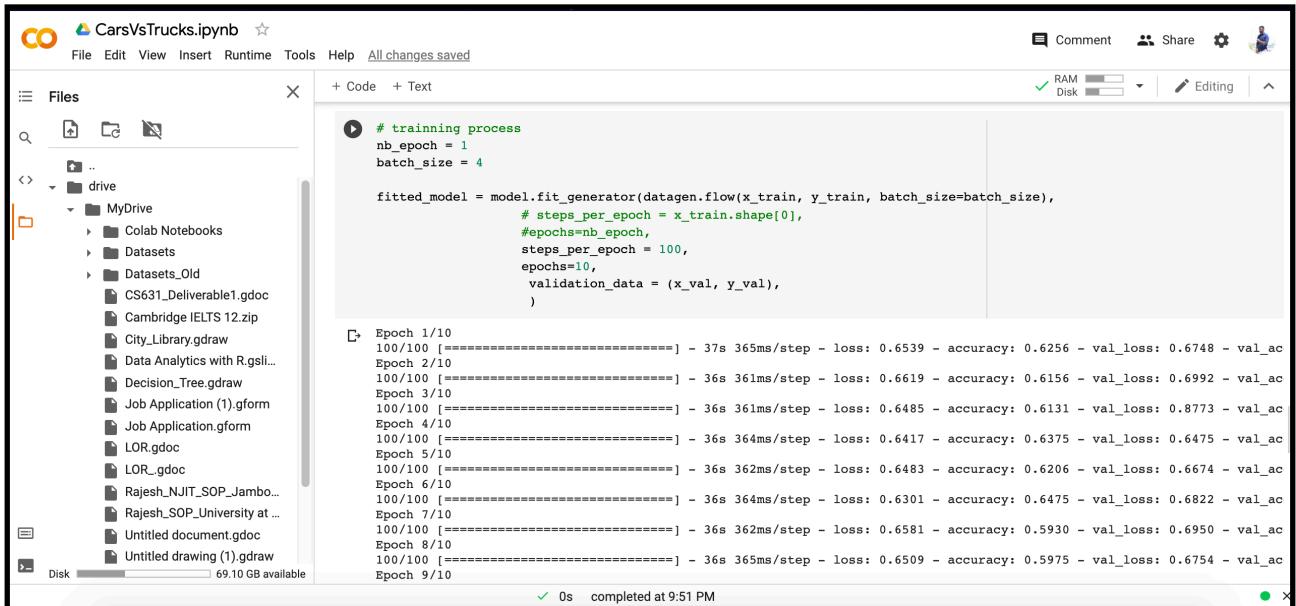
****steps_per_epoch:** Integer. Total number of steps (batches of samples) to yield from generator before declaring one epoch finished and starting the next epoch. It should typically be equal to $\text{ceil}(\text{num_samples} / \text{batch_size})$.

For example, if you have 100 training samples, then $\text{num_samples} = 100$, or the number of rows of x_{train} is 100.

You can specify your own batch size. In this case, say $\text{batch_size} = 20$. As a result, you can set your $\text{steps_per_epoch} = 100/20 = 5$ 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.



The screenshot shows a Google Colab notebook titled "CarsVsTrucks.ipynb". The left sidebar displays a file tree with various documents and datasets. The main area shows Python code for training a model:

```
# trainning process
nb_epoch = 1
batch_size = 4

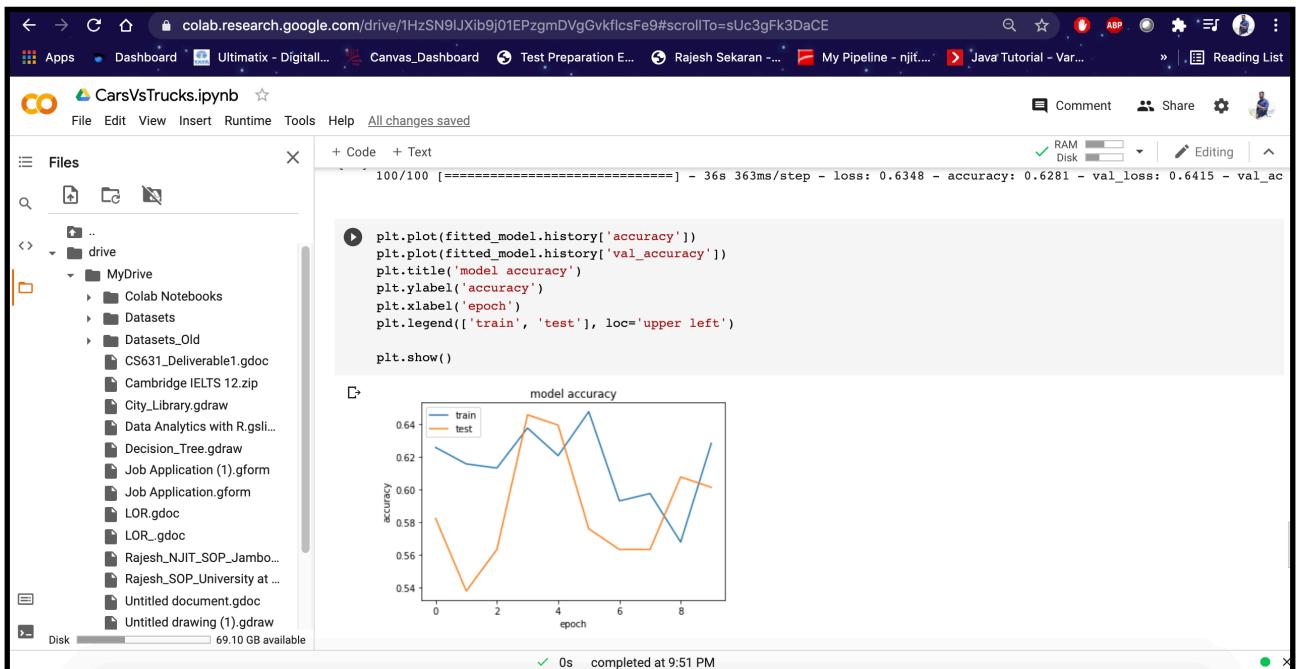
fitted_model = model.fit_generator(datagen.flow(x_train, y_train, batch_size=batch_size),
                                    # steps_per_epoch = x_train.shape[0],
                                    #epochs=nb_epoch,
                                    steps_per_epoch = 100,
                       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

Below the code, the output shows the training progress for 10 epochs:

```
Epoch 1/10
100/100 [=====] - 37s 365ms/step - loss: 0.6539 - accuracy: 0.6256 - val_loss: 0.6748 - val_ac
Epoch 2/10
100/100 [=====] - 36s 361ms/step - loss: 0.6619 - accuracy: 0.6156 - val_loss: 0.6992 - val_ac
Epoch 3/10
100/100 [=====] - 36s 361ms/step - loss: 0.6485 - accuracy: 0.6131 - val_loss: 0.8773 - val_ac
Epoch 4/10
100/100 [=====] - 36s 364ms/step - loss: 0.6417 - accuracy: 0.6375 - val_loss: 0.6475 - val_ac
Epoch 5/10
100/100 [=====] - 36s 362ms/step - loss: 0.6483 - accuracy: 0.6206 - val_loss: 0.6674 - val_ac
Epoch 6/10
100/100 [=====] - 36s 364ms/step - loss: 0.6301 - accuracy: 0.6475 - val_loss: 0.6822 - val_ac
Epoch 7/10
100/100 [=====] - 36s 362ms/step - loss: 0.6581 - accuracy: 0.5930 - val_loss: 0.6950 - val_ac
Epoch 8/10
100/100 [=====] - 36s 365ms/step - loss: 0.6509 - accuracy: 0.5975 - val_loss: 0.6754 - val_ac
Epoch 9/10
```

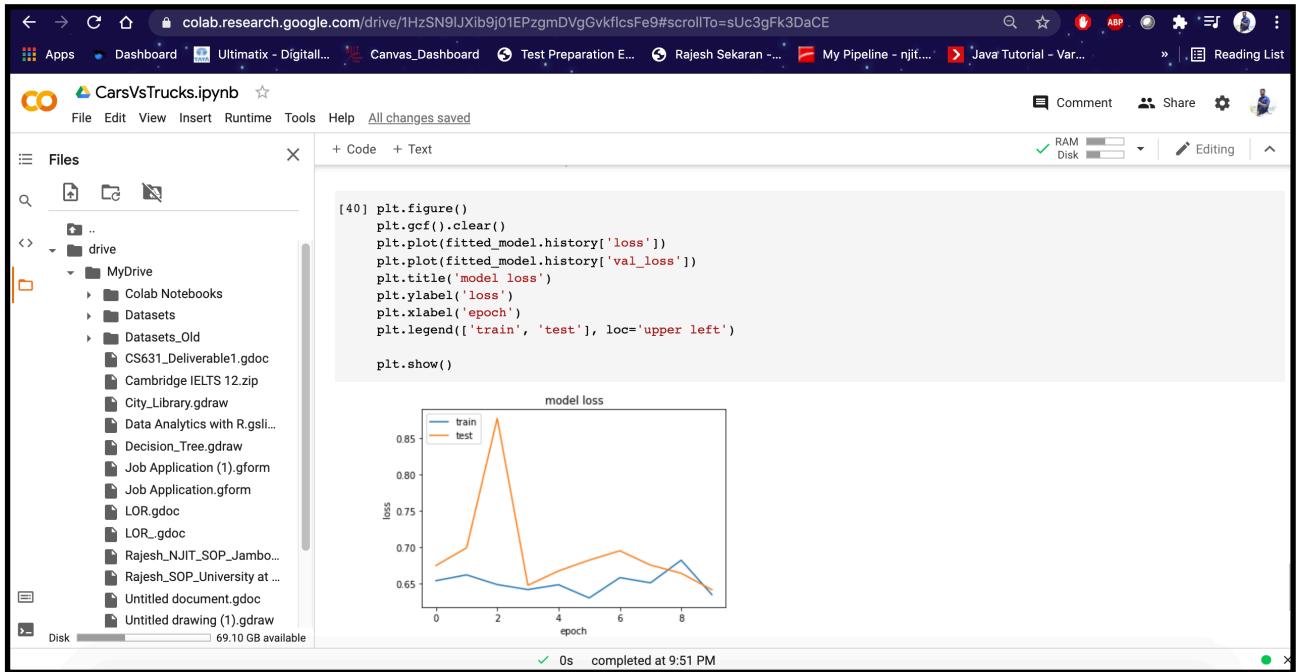
The status bar at the bottom indicates "0s completed at 9:51 PM".

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.

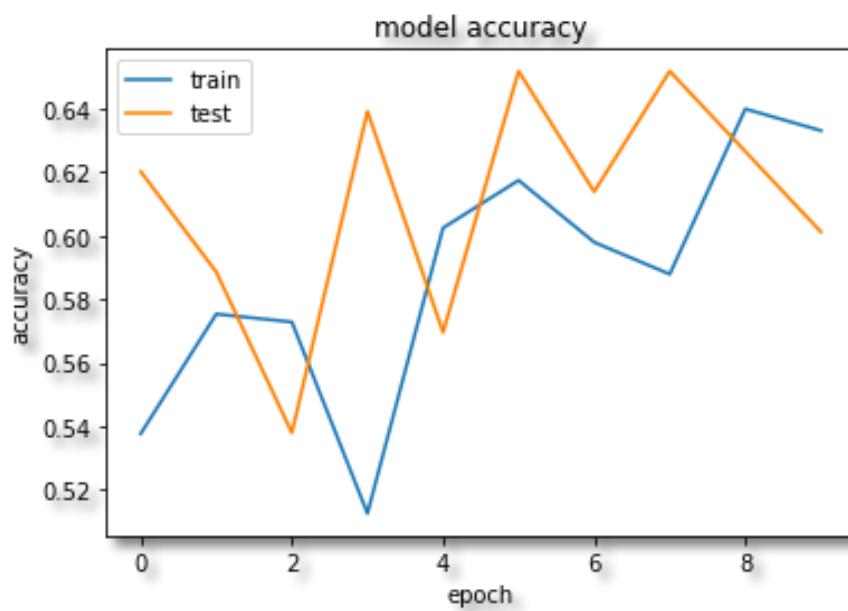
I also calculated the model loss accuracy as shown in below the train data loss has a big spike and then decreased , whereas the training data has stable loss.



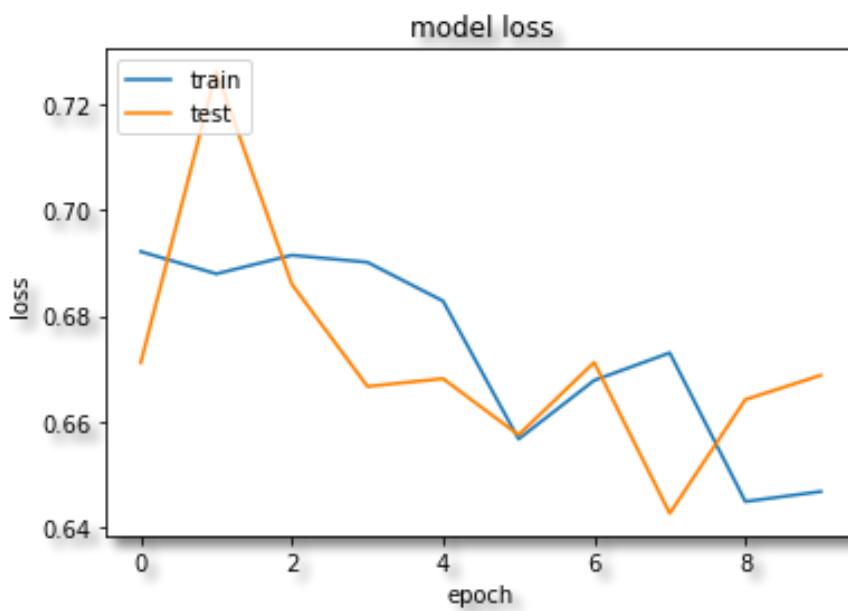
Testing another set of dataset :-

```
File Edit View Insert Runtime Tools Help All changes saved
Comment Share Settings User
File Edit View Insert Runtime Tools Help All changes saved
Reconnect Editing
[ ] validation_data = (x_val, y_val),
[ ] =====
[ ] - 6s 60ms/step - loss: 0.6921 - accuracy: 0.5377 - val_loss: 0.6712 - val_accuracy: 0.6203
[ ] =====
[ ] - 6s 58ms/step - loss: 0.6879 - accuracy: 0.5754 - val_loss: 0.7261 - val_accuracy: 0.5886
[ ] =====
[ ] - 6s 58ms/step - loss: 0.6914 - accuracy: 0.5729 - val_loss: 0.6860 - val_accuracy: 0.5380
[ ] =====
[ ] - 6s 58ms/step - loss: 0.6901 - accuracy: 0.5126 - val_loss: 0.6667 - val_accuracy: 0.6392
[ ] =====
[ ] - 6s 57ms/step - loss: 0.6828 - accuracy: 0.6025 - val_loss: 0.6682 - val_accuracy: 0.5696
[ ] =====
[ ] - 6s 58ms/step - loss: 0.6568 - accuracy: 0.6175 - val_loss: 0.6576 - val_accuracy: 0.6519
[ ] =====
[ ] - 6s 58ms/step - loss: 0.6678 - accuracy: 0.5980 - val_loss: 0.6712 - val_accuracy: 0.6139
[ ] =====
[ ] - 6s 59ms/step - loss: 0.6730 - accuracy: 0.5879 - val_loss: 0.6428 - val_accuracy: 0.6519
[ ] =====
[ ] - 6s 58ms/step - loss: 0.6450 - accuracy: 0.6400 - val_loss: 0.6642 - val_accuracy: 0.6266
[ ] =====
[ ] - 6s 60ms/step - loss: 0.6469 - accuracy: 0.6332 - val_loss: 0.6688 - val_accuracy: 0.6013
[ ] =====
[ ] 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')
```

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 :-

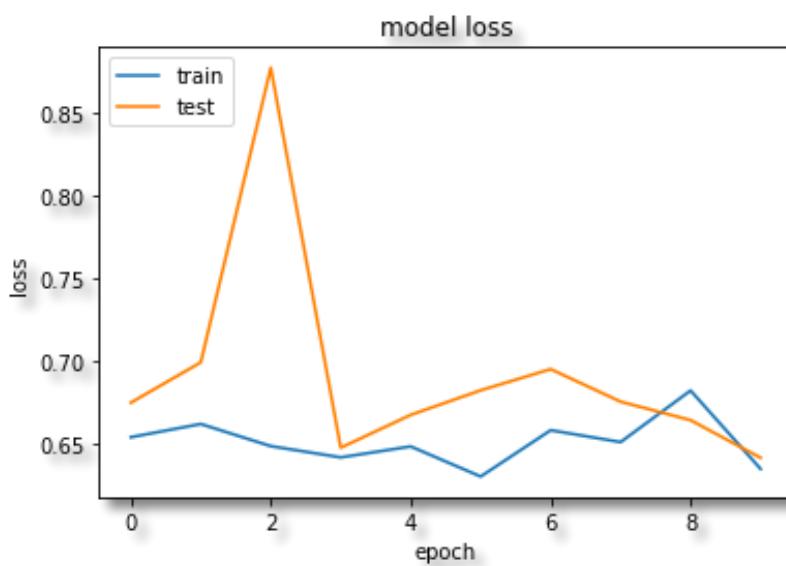
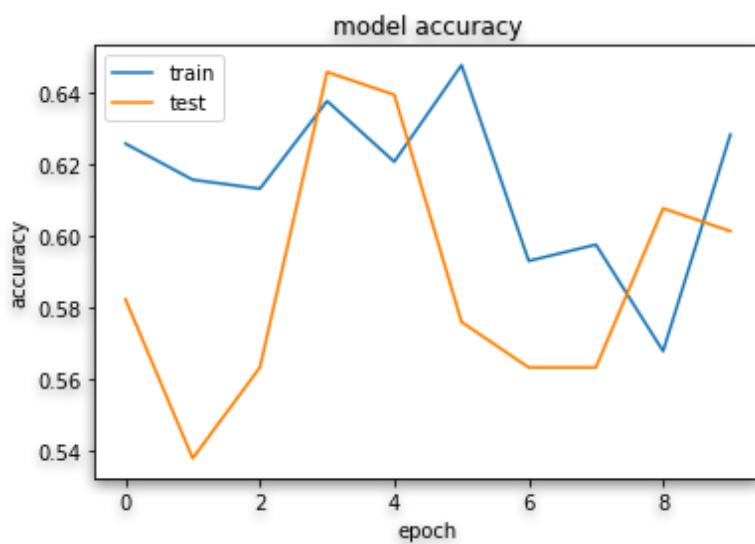
```
Epoch 1/10
100/100 [=====] - 37s 365ms/step -
loss: 0.6539 - accuracy: 0.6256 - val_loss: 0.6748 -
val_accuracy: 0.5823
Epoch 2/10
100/100 [=====] - 36s 361ms/step -
loss: 0.6619 - accuracy: 0.6156 - val_loss: 0.6992 -
val_accuracy: 0.5380
Epoch 3/10
100/100 [=====] - 36s 361ms/step -
loss: 0.6485 - 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.

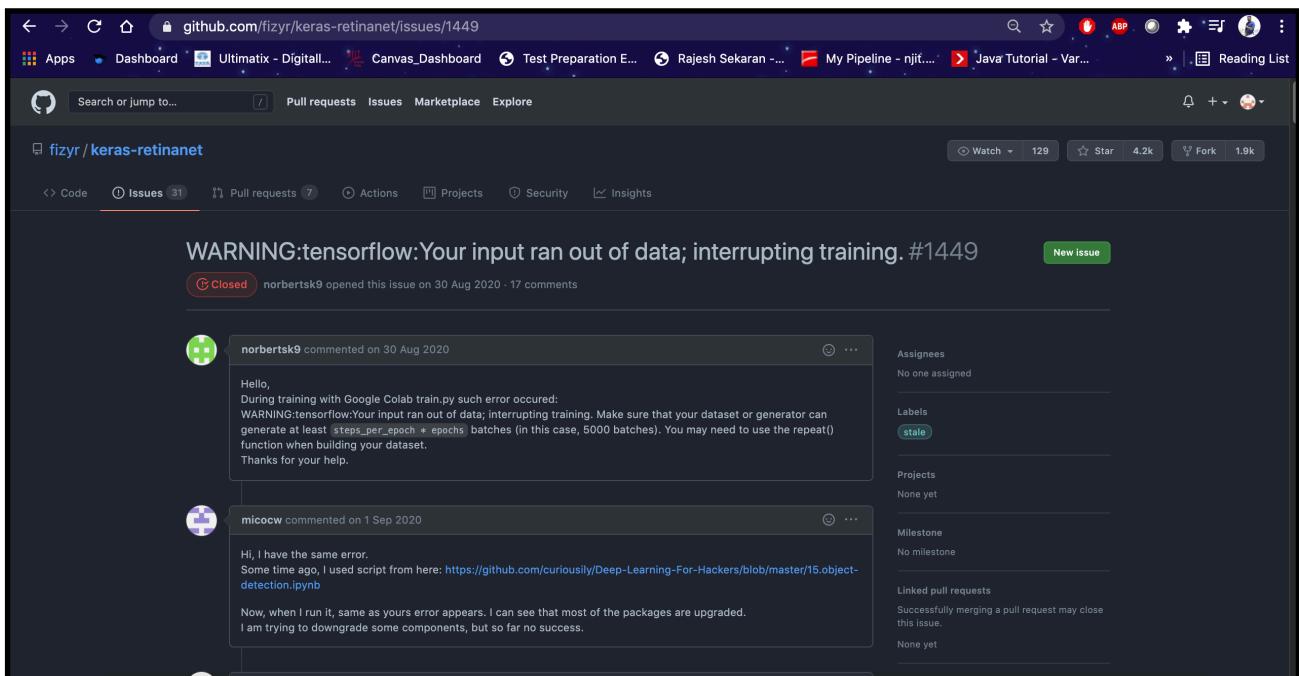
The size of a batch must be more than or equal to one and less than or equal to the number of samples in the training dataset.

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.

The screenshot shows a Stack Overflow question page with the URL stackoverflow.com/questions/59864408/tensorflow-your-input-ran-out-of-data. The question asks how to handle the error "WARNING :tensorflow:your input ran out of data; interrupting training." The accepted answer suggests setting `steps_per_epoch` to `len(X_train)//batch_size` and `validation_steps = len(X_test)//batch_size`. The post has 9 upvotes and 6 answers. The sidebar on the left includes links for Home, PUBLIC Questions, Tags, Users, FIND A JOB, and TEAMS. The footer features social media sharing options and a sign-up section.



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

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>