

A
Mini Project
On
**DEEPGRIP: CRICKET BOWLING DELIVERY DETECTION
WITH SUPERIOR CNN ARCHITECTURES**

(Submitted in partial fulfillment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

V. Madhusudhan Reddy(217R1A05R3)

E. Sai Akash Reddy (217R1A05M6)

K. Keerthana Reddy(217R1A05N6)

Under the Guidance of

K. PRAVEEN KUMAR

(Assistant Professor)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi)

Recognized Under Section 2(f) & 12(B) of the UGCAct.1956,

Kandlakoya (V), Medchal Road, Hyderabad-501401.

2021-25

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the project entitled “**DEEPGRIP:CRICKET BOWLING DELIVERY DETECTION WITH SUPERIOR CNN ARCHITECTURES**” being submitted by **V. MADHUSUDHAN REDDY (217R1A05R3), E. SAIKASH REDDY (217R1A05M6) and K. KEERTHANA REDDY (217R1A05N6)** in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out by them under our guidance and supervision during the year 2024-25.

The results embodied in this project have not been submitted to any other University or Institute for the award of any degree or diploma.

K. Praveen Kumar
(Assistant Professor)
INTERNAL GUIDE

Dr. A. Raji Reddy
DIRECTOR

Dr. N. Bhaskar
HOD

EXTERNAL EXAMINER

Submitted for viva voice Examination held on _____

ACKNOWLEDGEMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We take this opportunity to express my profound gratitude and deep regard to our guide **Mr. K. Praveen Kumar**, Associate Professor for his exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given by him shall carry us a long way in the journey of life on which we are about to embark.

We also take this opportunity to express a deep sense of gratitude to Project Review Committee (PRC) Coordinators: **Dr. J. Narasimha Rao, Mr. K. Ranjith Reddy, Dr. K. Maheshwari, Mrs. K. Shilpa** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to **Dr. N. Bhaskar**, Head of the Department of Computer Science and Engineering for providing encouragement and support for completing this project successfully.

We are obliged to **Dr. A. Raji Reddy**, Director for being cooperative throughout the course of this project. We would like to express our sincere gratitude to **Sri. Ch. Gopal Reddy**, Chairman for providing excellent infrastructure and a nice atmosphere throughout the course of this project.

The guidance and support received from all the members of **CMR Technical Campus** who contributed to the completion of the project. We are grateful for their constant support and help.

Finally, we would like to take this opportunity to thank our family for their constant encouragement, without which this assignment would not be completed. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project.

V.MADHUSUDHAN REDDY (217R1A05R3)
E.SAI AKASH REDDY (217R1A05M6)
K. KEERTHANA REDDY (217R1A05N6)

ABSTRACT

Delivery in cricket is the sole action of bowling a cricket ball towards the batsman. The outcome of the ball is immensely pivoted on the grip of the bowler. An instance when whether the ball is going to take a sharp turn or keeps straight through with the arm depends entirely upon the grip. And to the batsmen, the grip of the cricket bowl is one of the biggest enigmas. Without acknowledging the grip of the bowl and having any clue of the behavior of the ball, the mis-hit of a ball is the most likely outcome due to the variety in bowling present in modern-day cricket. This project proposed a novel strategy to identify the type of delivery from the finger grip of a bowler while the bowler makes a delivery. The main purpose of this research is to utilize the preliminary CNN architecture and the transfer learning models to perfectly classify the grips of bowlers. A new dataset of 5573 images from Real-Time videos in offline mode were prepared for this research, named GRIP DATASET, consisted of grip images of 13 different classes. Hence the preliminary CNN model and the pre-trained transfer learning models - Vgg16, Vgg19, ResNet101, ResNet152, Dense Net, Mobile Net, Alex Net, Inception V3, and Nas Net were used to train with GRIP DATASET and analyze the outcome of grips. The training and validation accuracies of the models are noteworthy with the maximum validation accuracy of the preliminary model reaching 98.75%. This study is expected to be yet another steppingstone in the use of deep learning for the game of cricket.

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
Figure 3.1	Project Architecture	6
Figure 3.2	Use case diagram	7
Figure 3.3	Class diagram	8
Figure 3.4	Sequence diagram	9
Figure 3.5	Activity diagram	10

LIST OF SCREENSHOTS

SCREENSHOT NO.	SCREENSHOT NAME	PAGE NO.
Screenshot 5.1	Dataset Screen	15
Screenshot 5.2	Training Dataset	16
Screenshot 5.3	Connecting dataset	17
Screenshot 5.4	Graph Plotting	18
Screenshot 5.5	Processing Dataset	19
Screenshot 5.6	Splitting Dataset	20
Screenshot 5.7	Calculating Metrics	21
Screenshot 5.8	Training CNN Model	22
Screenshot 5.9	Alexnet Accuracy	23
Screenshot 5.10	Alexnet Code	24
Screenshot 5.11	Alexnet Result	25
Screenshot 5.12	Resnet Model	26
Screenshot 5.13	Resnet Result	27
Screenshot 5.14	VGG16	28
Screenshot 5.15	VGG16 Result	29
Screenshot 5.16	Densenet	30
Screenshot 5.17	Densenet Result	31
Screenshot 5.18	Graph Plotting	32
Screenshot 5.19	Graph Loss	33

LIST OF SCREENSHOTS

SCREENSHOT NO.	SCREENSHOT NAME	PAGE NO.
Screenshot 5.20	Graph Plotting	34
Screenshot 5.21	Performance	35
Screenshot 5.22	Grip Prediction	36
Screenshot 5.23	Grip Predicted	37
Screenshot 5.24	Predicted Output	38

LIST OF SCREENSHOTS

TABLE OF CONTENTS

ABSTRACT	i
LIST OF FIGURES	ii
LIST OF SCREENSHOTS	iii
1. INTRODUCTION	1
1.1 PROJECT SCOPE	1
1.2 PROJECT PURPOSE	1
1.3 PROJECT FEATURES	1
2. SYSTEM ANALYSIS	2
2.1 PROBLEM DEFINITION	2
2.2 EXISTING SYSTEM	2
2.2.1 LIMITATIONS OF THE EXISTING SYSTEM	2
2.3 PROPOSED SYSTEM	3
2.3.1 ADVANTAGES OF PROPOSED SYSTEM	3
2.4 FEASIBILITY STUDY	3
2.4.1 ECONOMIC FESIBILITY	4
2.4.2 TECHNICAL FEASIBILITY	4
2.4.3 BEHAVIOURAL FEASIBILITY	4
2.5 HARDWARE & SOFTWARE REQUIREMENTS	5
2.5.1 HARDWARE REQUIREMENTS	5
2.5.2 SOFTWARE REQUIREMENTS	5
3. ARCHITECTURE	6
3.1 PROJECT ARCHITECTURE	6
3.2 DESCRIPTION	6
3.3 USECASE DIAGRAM	7
3.4 CLASS DIAGRAM	8
3.5 SEQUENCE DIAGRAM	9
3.6 ACTIVITY DIAGRAM	10
4. IMPLEMENTATION	11
4.1 SAMPLE CODE	11
5. SCREENSHOTS	15
6. TESTING	39

6.1	INTRODUCTION TO TESTING	39
6.2	TYPES OF TESTING	39
6.2.1	UNIT TESTING	39
6.2.2	INTEGRATION TESTING	39
6.2.3	FUNCTIONAL TESTING	40
7.	CONCLUSION & FUTURE SCOPE	41
7.1	PROJECT CONCLUSION	41
7.2	FUTURE SCOPE	41
8.	BIBLIOGRAPHY	42
8.1	REFERENCES	42
8.2	WEBSITES	42

1. INTRODUCTION

1. INTRODUCTION

1.1 PROJECT SCOPE

This project is titled as “Deep grip: Cricket Bowling Delivery Detection With Superior CNN Architectures”. This software Develop an AI system for automatic detection of cricket bowling deliveries using advanced CNN architectures. The project focuses on identifying different delivery types from video footage, with model training for high accuracy and real-time performance. Video pre-processing and motion feature extraction are key components. A dashboard will visualize delivery patterns for player analysis.

1.2 PROJECT PURPOSE

The purpose of this project is to create an AI-based system that can accurately detect and classify various types of cricket bowling deliveries using superior CNN architectures. This will aid in enhancing match analysis, training, and coaching by providing real-time insights into bowler performance.

1.3 PROJECT FEATURES

The project features automatic detection and classification of various cricket bowling deliveries, such as yorkers, bouncers, and spin, using advanced CNN models for high accuracy and efficiency. It includes real-time detection capabilities, making it suitable for live matches or training sessions. Video pre-processing and motion feature extraction are applied to improve the accuracy of delivery recognition. The system ensures fast processing and high performance for practical use. Additionally, an analytics dashboard provides visual insights into bowler performance and delivery patterns for enhanced analysis.

2. SYSTEM ANALYSIS

2. SYSTEM ANALYSIS

SYSTEM ANALYSIS

System analysis for this project focuses on the design and functionality of the AI-based cricket bowling delivery detection system. The system will process cricket video footage, applying video pre-processing techniques to extract key frames and motion features related to bowling deliveries. Superior CNN architectures will be employed to detect and classify various delivery types (e.g., yorkers, bouncers) with high accuracy. The analysis includes optimizing model performance for real-time use, ensuring low latency during live matches or training sessions

2.1 PROBLEM DEFINITION

The problem addressed by this project is the lack of automated and accurate systems for detecting and classifying different types of cricket bowling deliveries from video footage in real-time. Current methods of analyzing bowlers' performance are often manual, time-consuming, and subject to human error, which limits their effectiveness during live matches or training sessions. Coaches and analysts need a tool that can quickly and accurately detect bowling delivery types (e.g., yorkers, bouncers, spin) to provide immediate insights for strategy and improvement.

2.2 EXISTING SYSTEM

Before training with the prominent CNN architectures, a simple preliminary CNN architecture was developed with lower parameters and much shallower and simpler architecture than other transfer learning models. It is used to test the hypothesis about analyzing the grips of fingers using CNN architectures through its outcomes. Then the prominent pre-trained transfer learning models like Vgg16, Vgg19, Inception V3, Mobile Net, Alex Net, Nas Net, Res Net 101, Res Net 152.

2.2.1 LIMITATIONS OF EXISTING SYSTEM

- Prediction of bowling action is very difficult

- We are getting results with low accuracy.
- Time taken process.

To avoid all these limitations and make the working more accurately the system needs to be implemented efficiently.

2.3 PROPOSED SYSTEM

The aim of proposed system is to develop a system of improved facilities. In proposed ,we are training different deep learning algorithms such as CNN, Alex net, Resnet101, VGG16 and Dense Net to predict cricket ball Grip Type such as ‘Arm, Swing, Carom, Flipper, Leg Break and Googly. The proposed system helps the user to work user friendly and he can easily do his jobs without time lagging.

2.3.1 ADVANTAGES OF THE PROPOSED SYSTEM

The system is very simple in design and to implement. The system requires very low system resources and the system will work in almost all configurations. It has got following features

- Prediction of bowling action is easy
- We will get results with better accuracy
- Less time will take to predict result

2.4 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. Three key considerations involved in the feasibility analysis are

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

2.4.1 ECONOMIC FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased

2.4.2 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.4.3 BEHAVIORAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system

2.5 HARDWARE & SOFTWARE REQUIREMENTS

2.5.1 HARDWARE REQUIREMENTS:

Hardware interfaces specifies the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

- System : Pentium IV 2.4 GHz
- Hard disk : 40 GB
- RAM : 4GB and above
- Monitor : 15 VGA Color
- Floppy Drive : 1.44MB

2.5.2 SOFTWARE REQUIREMENTS:

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software requirements:

- Operating system : Windows 7 and above
- Languages : Python

3. ARCHITECTURE

3. ARCHITECTURE

3.1 PROJECT ARCHITECTURE

This project architecture shows the procedure followed for Cricket Bowling Detection with Superior CNN Architecture, starting from input to final prediction.

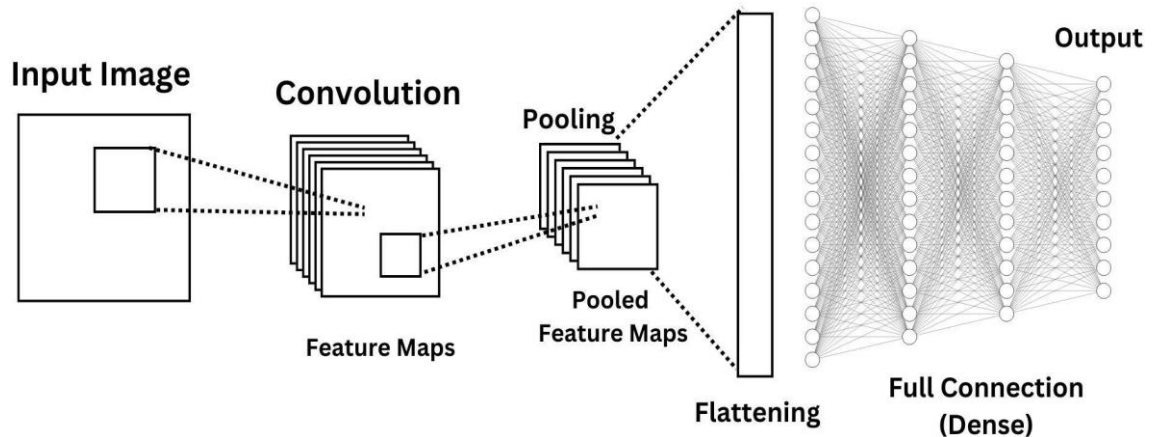


Figure 3.1: Project Architecture of for Cricket Bowling Delivery Detection with Superior CNN Architecture

3.2 DESCRIPTION

Input Data: Input data is generally in .jpg format or .png format where the data is fetched and mapped in the data framed from the source columns.

Reading Data: Torch vision library is used to read the data into the data frame.

Separating Features: In this following step we are going to separate the features which we take to train the model by giving the target value i.e. 1/0 for the particular of features.

Normalization: Normalization is a very important step while we are dealing with the large values in the features as the higher bit integers will cost high computational power and time. To achieve the efficiency in computation we are going to normalize the data values.

Training and test data: Training data is passed to the VGG classifier to train the model. Test data is used to test the trained model whether it is making correct predictions or not.

VGG Classifier: the purpose of choosing the VGG classifier for this project the efficiency and accuracy that we have observed when compared to other classifiers.

3.3 USE CASE DIAGRAM

A use case diagram is a visual representation of the interactions between users (actors) and a system, showing how the system responds to different user actions or requests.

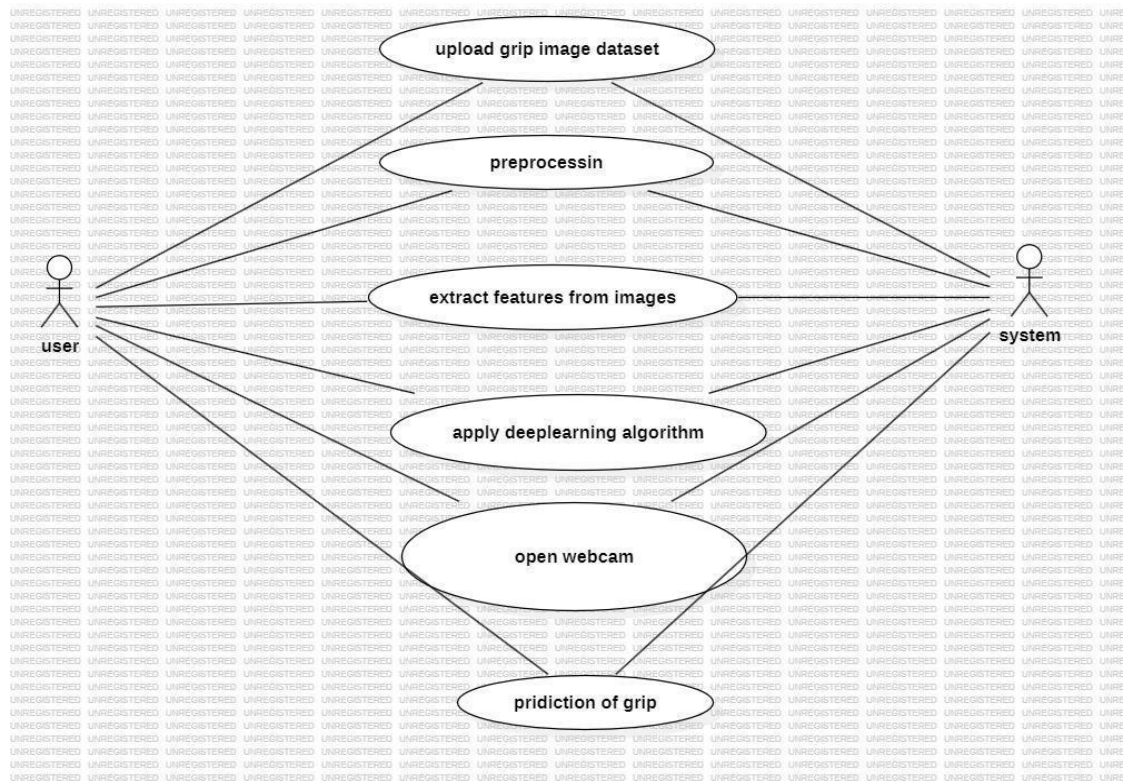


Figure 3.2: Use Case Diagram for user for Cricket Bowling Delivery Detection with Superior CNN Architecture

3.4 CLASS DIAGRAM

Class Diagram is a collection of classes and objects.

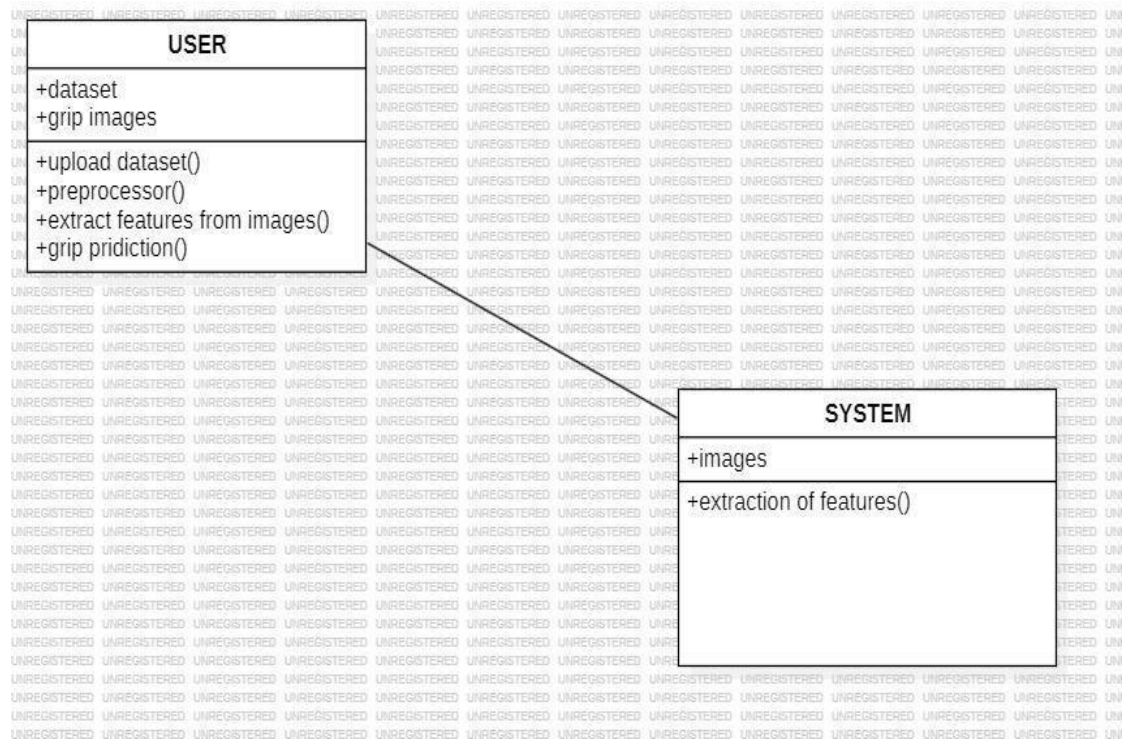


Figure 3.3: Class Diagram for Cricket Bowling Delivery Detection with Superior CNN Architecture

3.5 SEQUENCE DIAGRAM

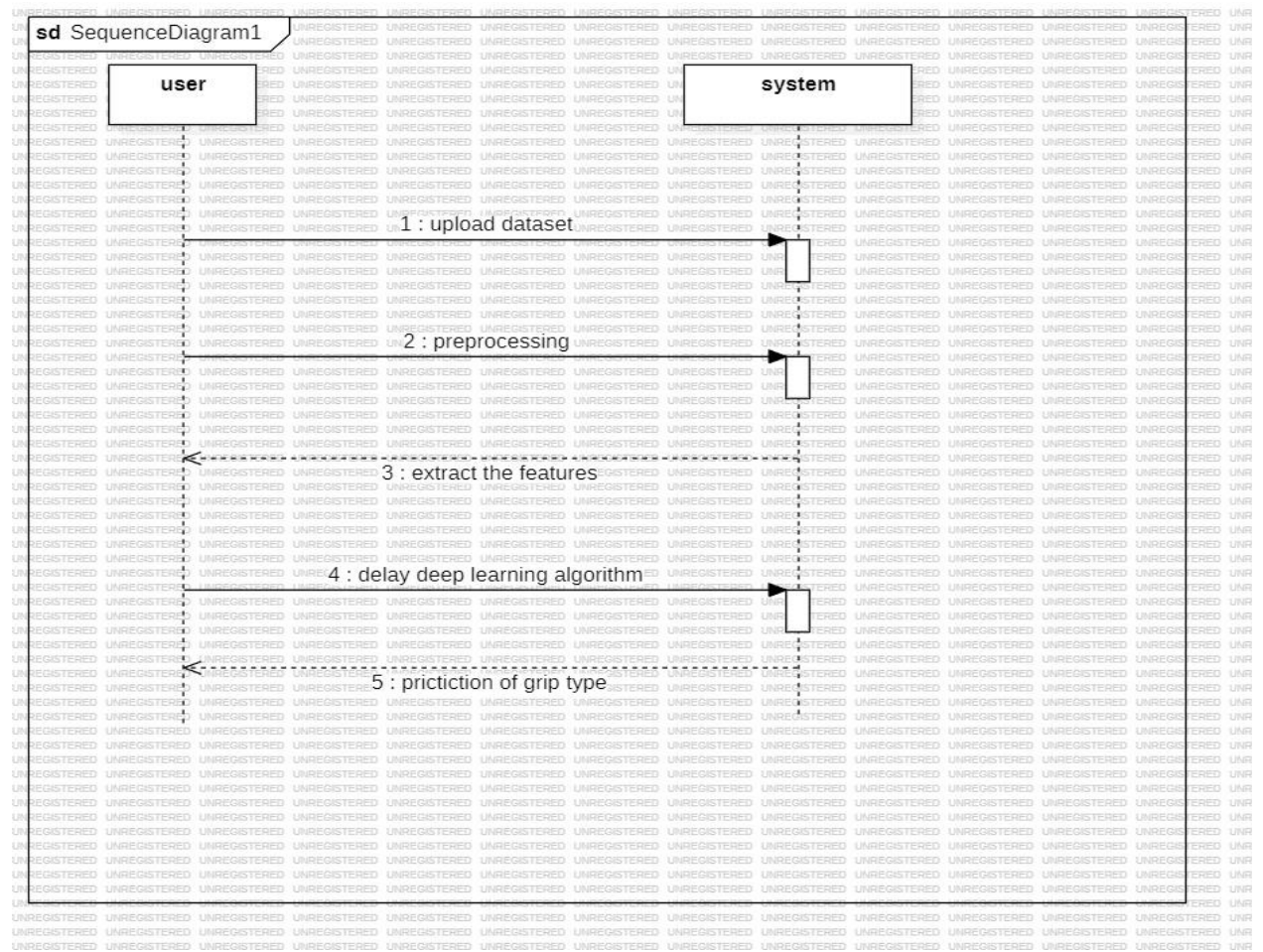


Figure 3.4: Sequence Diagram for Cricket Bowling Delivery Detection with Superior CNN Architecture

3.6 ACTIVITY DIAGRAM

It describes about flow of activity states.

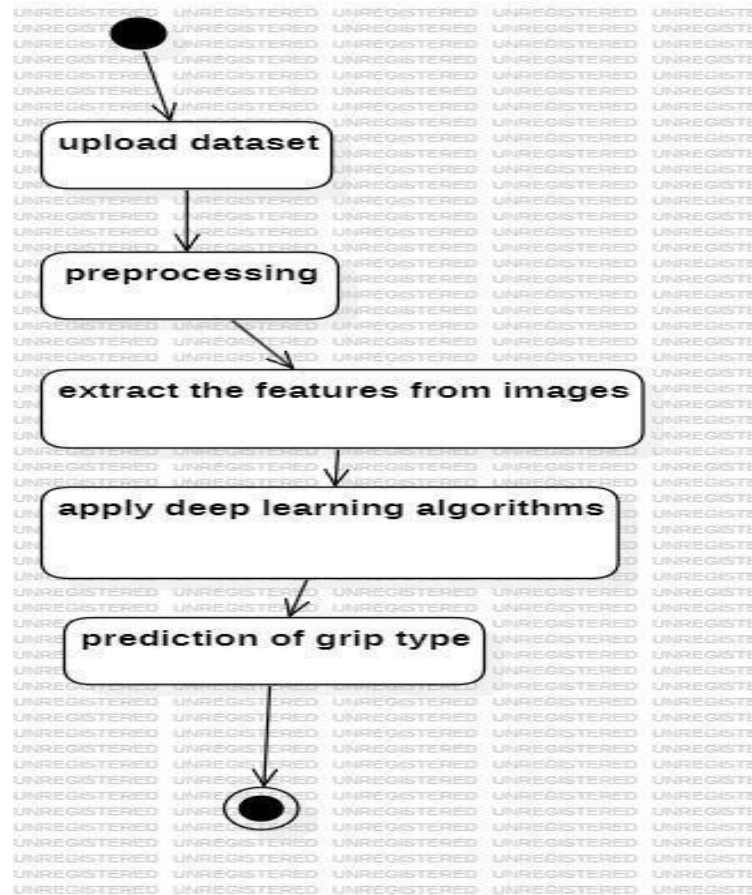


Figure 3.5: Activity Diagram for Cricket Bowling Delivery Detection with Superior CNN Architecture

4. IMPLEMENTATION

4. IMPLEMENTATION

4.1 SAMPLE CODE

Check_images.py:

```
# Imports python
modules from time
import time, sleep

# Imports print functions that check
the lab from
print_functions_for_lab_checks
import *

# Imports functions created for this
program from get_input_args import
get_input_args from get_pet_labels
import get_pet_labels from
classify_images import
classify_images
from adjust_results4_isadog import
adjust_results4_isadog from
calculates_results_stats import
calculates_results_stats from print_results import
print_results

def main():
# Measures total program runtime by collecting start
time start_time = time()

# Define get_input_args function within the file get_input_args.py
# This function retrieves 3 Command Line Arguments from user as
input from # the user running the program from a terminal window.
```

```

This function returns # the collection of these command line
arguments from the function call as
# the variable in_arg
in_arg          =
get_input_args()

# Function that checks command line arguments
using in_arg
check_command_line_arguments(in_arg)

# Define get_pet_labels function within the file
get_pet_labels.py # Once the get_pet_labels function has
been defined replace 'None'
# in the function call with in_arg.dir Once you have done the
replacements # your function call should look like this:
#     get_pet_labels(in_arg.dir)
# This function creates the results dictionary that contains the results,
# this dictionary is returned from the function call as the variable
results results = get_pet_labels(in_arg.dir)

# Function that checks Pet Images in the results Dictionary using results
check_creating_pet_image_labels(results)

# Define classify_images function within the file classiy_images.py
# Once the classify_images function has been defined replace
first 'None' # in the function call with in_arg.dir and replace the
last 'None' in the
# function call with in_arg.arch Once you have done the
replacements your # function call should look like this:
#     classify_images(in_arg.dir, results, in_arg.arch)
# Creates Classifier Labels with classifier function, Compares Labels,

# and adds these results to the results dictionary -
results classify_images(in_arg.dir, results,

```

```

in_arg.arch)

# Function that checks Results Dictionary using
results check_classifying_images(results)


# Define adjust_results4_isadog function within the file
adjust_results4_isadog.py # Once the adjust_results4_isadog function
has been defined replace 'None'
# in the function call with in_arg.dogfile Once you have
done the # replacements your function call should look
like this:
#     adjust_results4_isadog(results, in_arg.dogfile)
# Adjusts the results dictionary to determine if classifier
correctly # classified images as 'a dog' or 'not a dog'.
This demonstrates if
# model can correctly classify dog images as dogs (regardless of breed)
adjust_results4_isadog(results, in_arg.dogfile)


# Function that checks Results Dictionary for is-a-dog adjustment using results
check_classifying_labels_as_dogs(results)


# Define calculates_results_stats function within the file
calculates_results_stats.py # This function creates the results statistics
dictionary that contains a
# summary of the results statistics (this includes counts &
percentages). This # dictionary is returned from the function call
as the variable results_stats
# Calculates results of run and puts statistics in the Results
Statistics # Dictionary - called results_stats
results_stats = calculates_results_stats(results)


# Function that checks Results Statistics Dictionary using results_stats
check_calculating_results(results, results_stats)

```

Define print_results function within the file print_results.py

Once the print_results function has been defined

replace 'None' # in the function call with in_arg.arch

Once you have done the

replacements your function call should look like this:

print_results(results, results_stats, in_arg.arch, True, True)

Prints summary results, incorrect classifications of dogs (if requested) # and incorrectly classified breeds (if requested)

print_results(results, results_stats, in_arg.arch, True, True)

Measure total program runtime by collecting

end time end_time =time()

Computes overall runtime in seconds & prints it in

hh:mm:ss format tot_time = end_time-start_time

print("\n** Total Elapsed Runtime:",

str(int((tot_time/3600)))+":"+str(int((tot_time%3600)/60
))+":")

+str(int((tot_time%3600)%60)))

Call to main function to run the

program if __name__ == "

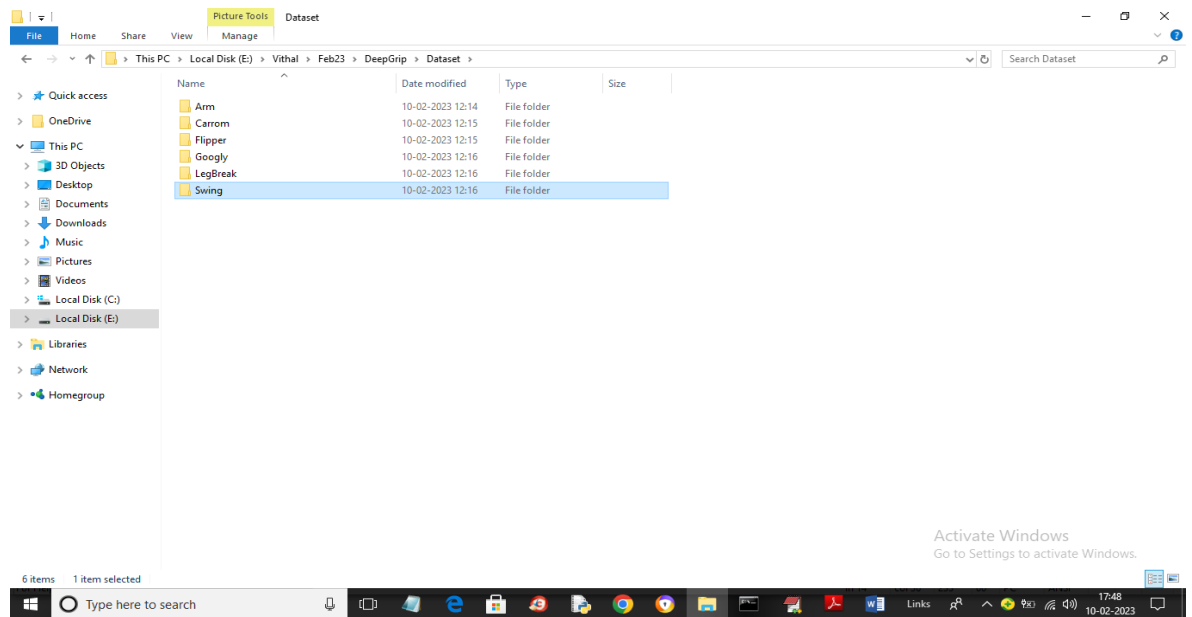
main ":

- -

main()

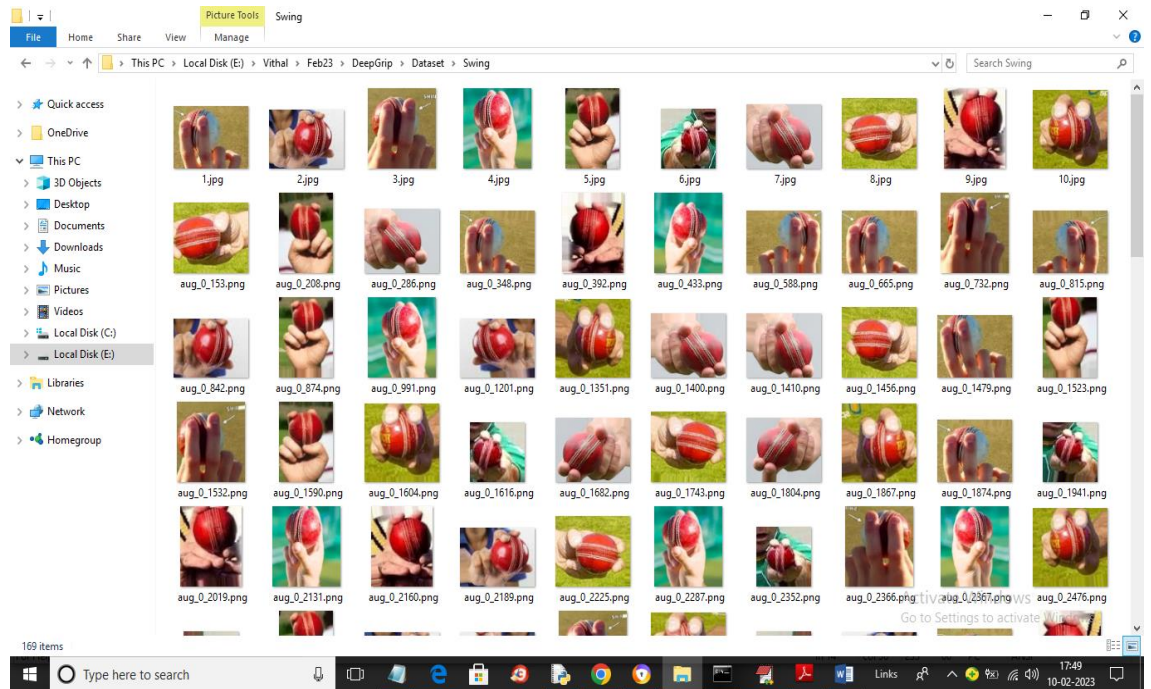
5. SCREENSHOTS

5.1 DATASET SCREEN



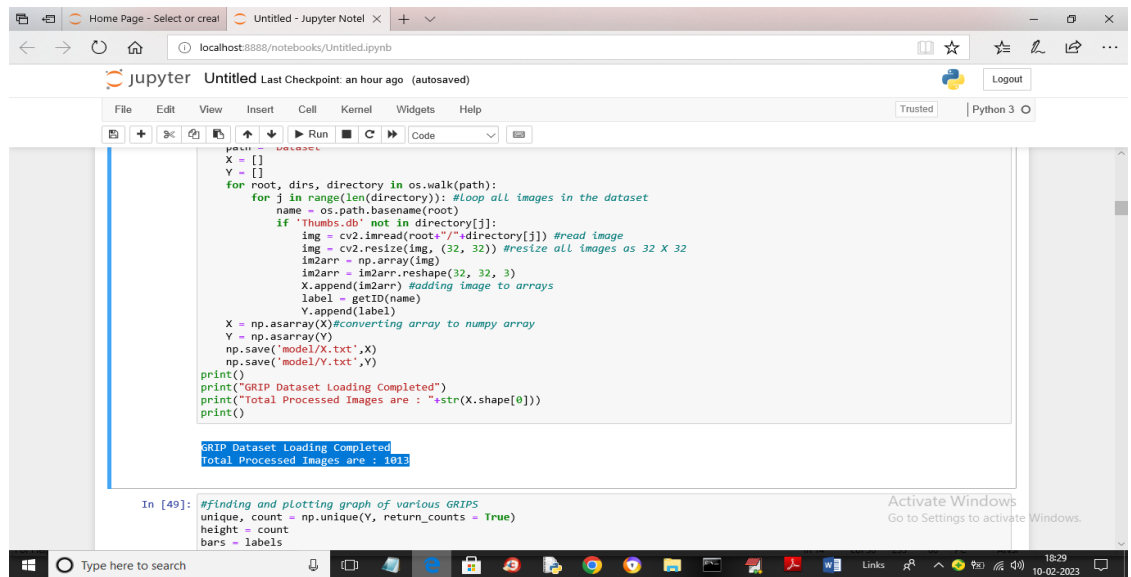
Screenshot 5.1: In above Dataset screen we have 6 different folders and just go inside any folder to view images like below screen

5.2 TRAINING DATASETS



Screenshot 5.2 : So by using above images we are training all deep learning algorithms and their performance in terms of accuracy, precision, recall, FSCORE and confusion matrix.

5.3 CONNECTING TO DATASET



```

python -m unittest
X = []
Y = []
for root, dirs, directory in os.walk(path):
    for j in range(len(directory)): #loop all images in the dataset
        name = os.path.basename(root)
        if 'Thumbs.db' not in directory[j]:
            img = cv2.imread(root+"/"+directory[j]) #read image
            img = cv2.resize(img, (32, 32)) #resize all images as 32 X 32
            im2arr = np.array(img)
            im2arr = im2arr.reshape(32, 32, 3)
            X.append(im2arr) #adding image to arrays
            label = getID(name)
            Y.append(label)

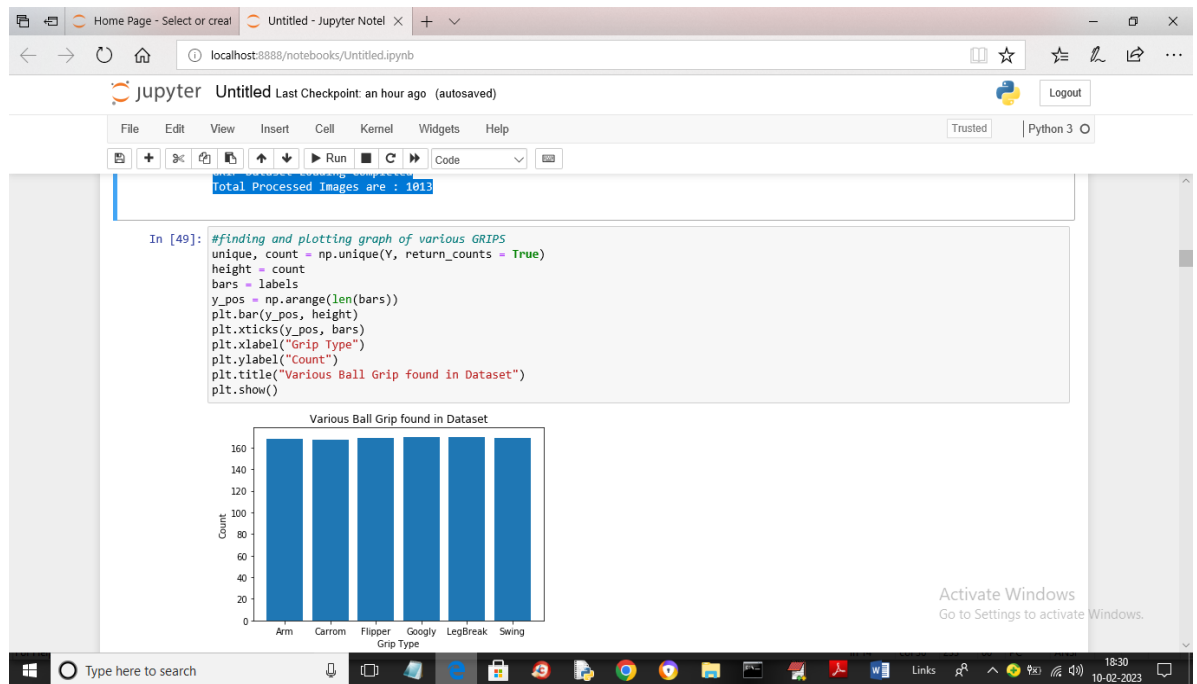
X = np.asarray(X)#converting array to numpy array
Y = np.asarray(Y)
np.save('model/X.txt',X)
np.save('model/Y.txt',Y)
print()
print("GRIP Dataset Loading Completed")
print("Total Processed Images are : "+str(X.shape[0]))
print()

GRIP Dataset Loading Completed
Total Processed Images are : 1013

In [49]: #finding and plotting graph of various GRIPS
unique, count = np.unique(Y, return_counts = True)
height = count
bars = labels
  
```

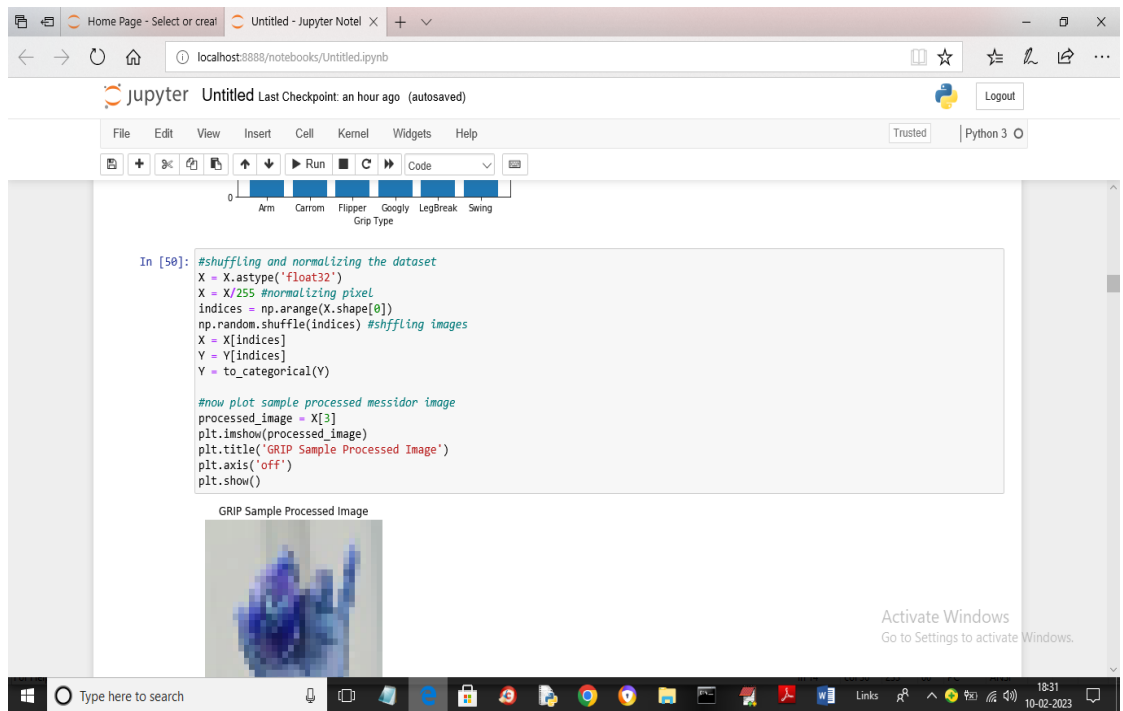
Screenshot 5.3 :In above screen connecting to dataset and then looping all images and resizing to equal as 32 X 32 with 3 channels such as RGB and then adding X and Y array and then in blue color text we can see total images loaded

5.4 GRAPH PLOTTING



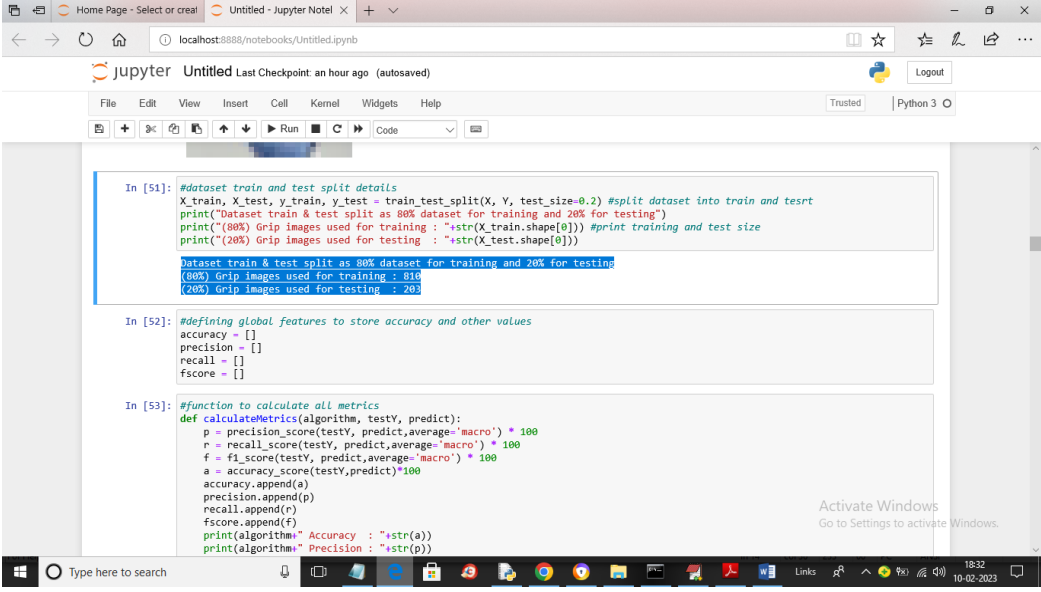
Screenshot 5.4 : In above screen we are plotting graph of different grips found in dataset where x-axis represents grip name and y-axis represents count

5.5 PROCESSING DATASET



Screenshot 5.5 : In above screen we are processing dataset such as shuffling images and then normalizing pixel values and then displaying one processed image

5.6 SPLITTING DATASET



The screenshot shows a Jupyter Notebook window titled 'Untitled - Jupyter Noteb...' with the URL 'localhost:8888/notebooks/Untitled.ipynb'. The notebook contains three code cells:

```
In [51]: #dataset train and test split details
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2) #split dataset into train and test
print("Dataset train & test split as 80% dataset for training and 20% for testing")
print("(80%) Grip images used for training : "+str(X_train.shape[0])) #print training and test size
print("(20%) Grip images used for testing : "+str(X_test.shape[0]))

Dataset train & test split as 80% dataset for training and 20% for testing
(80%) Grip images used for training : 810
(20%) Grip images used for testing : 203
```

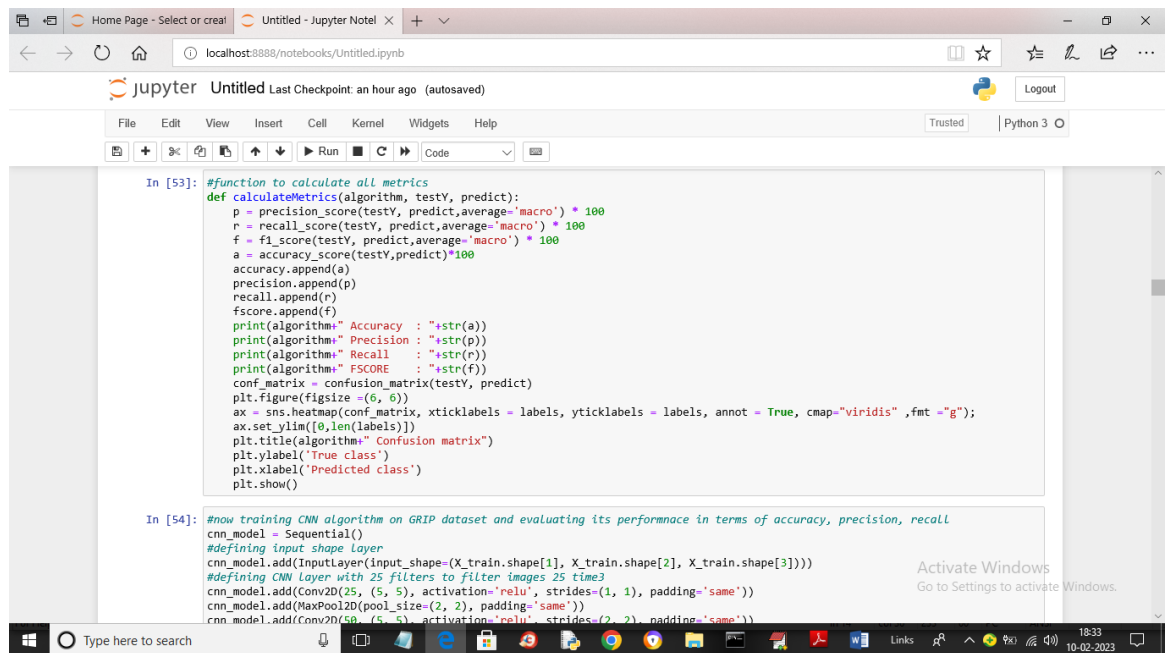
```
In [52]: #defining global features to store accuracy and other values
accuracy = []
precision = []
recall = []
fscore = []
```

```
In [53]: #function to calculate all metrics
def calculateMetrics(algorithm, testY, predict):
    p = precision_score(testY, predict, average='macro') * 100
    r = recall_score(testY, predict, average='macro') * 100
    f = f1_score(testY, predict, average='macro') * 100
    a = accuracy_score(testY, predict) * 100
    accuracy.append(a)
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    print(algorithm+" Accuracy : "+str(a))
    print(algorithm+" Precision : "+str(p))
```

The output of the first cell shows the dataset split results: 810 images for training and 203 images for testing. The second cell defines global variables for accuracy, precision, recall, and fscore. The third cell defines a function to calculate all metrics.

Screenshot 5.6 : In above screen we are splitting dataset into train and test where application using 80% images for training and 20% for testing and in blue colour you can see size of images used for training and testing and then defining global variables to store accuracy and other metrics

5.7 CALCULATING METRICS



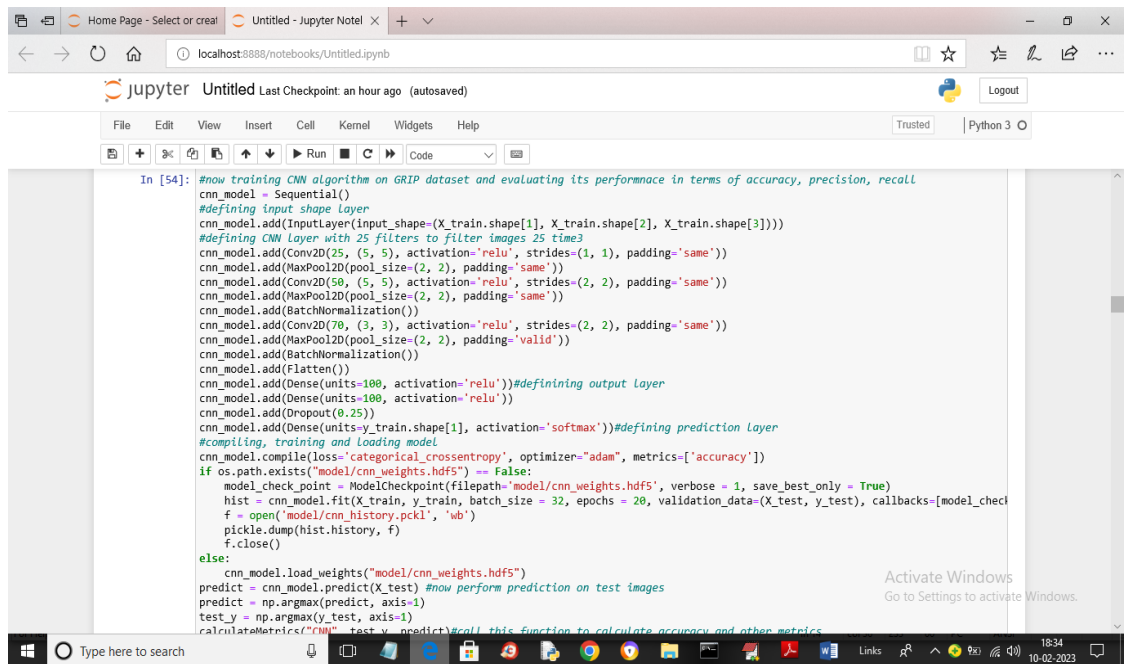
```

In [53]: #function to calculate all metrics
def calculateMetrics(algorithm, testY, predict):
    p = precision_score(testY, predict, average='macro') * 100
    r = recall_score(testY, predict, average='macro') * 100
    f = f1_score(testY, predict, average='macro') * 100
    a = accuracy_score(testY, predict) * 100
    accuracy.append(a)
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    print(algorithm+" Accuracy : "+str(a))
    print(algorithm+" Precision : "+str(p))
    print(algorithm+" Recall : "+str(r))
    print(algorithm+" FSCORE : "+str(f))
    conf_matrix = confusion_matrix(testY, predict)
    plt.figure(figsize=(6, 6))
    ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="viridis", fmt="g");
    ax.set_ylim([0, len(labels)])
    plt.title(algorithm+" Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()

In [54]: #now training CNN algorithm on GRIP dataset and evaluating its performance in terms of accuracy, precision, recall
cnn_model = Sequential()
#defining input shape layer
cnn_model.add(InputLayer(input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])))
#defining CNN layer with 25 filters to filter images 25 times
cnn_model.add(Conv2D(25, (5, 5), activation='relu', strides=(1, 1), padding='same'))
cnn_model.add(MaxPool2D(pool_size=(2, 2), padding='same'))
cnn_model.add(Conv2D(50, (5, 5), activation='relu', strides=(2, 2), padding='same'))
  
```

Screenshot 5.7 : In above screen we are defining function to calculate accuracy, precision, recall and other metrics

5.8 TRAINING CNN MODEL



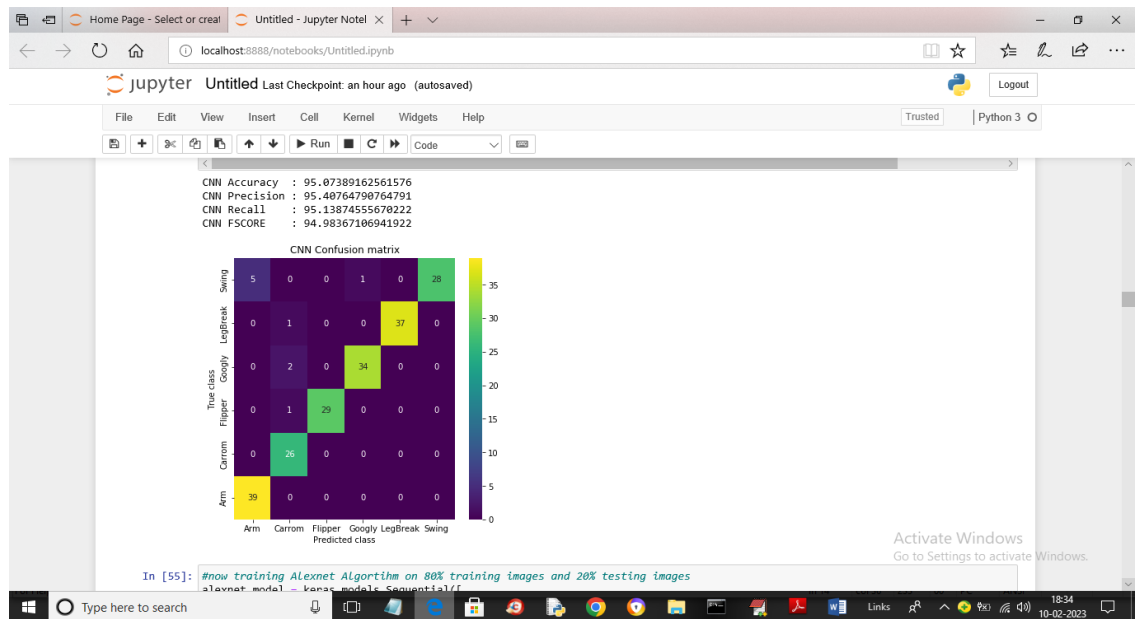
```

In [54]: #now training CNN algorithm on GRIP dataset and evaluating its performace in terms of accuracy, precision, recall
cnn_model = Sequential()
#defining input shape Layer
cnn_model.add(InputLayer(input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])))
#defining CNN Layer with 25 filters to filter images 25 time3
cnn_model.add(Conv2D(25, (5, 5), activation='relu', strides=(1, 1), padding='same'))
cnn_model.add(MaxPool2D(pool_size=(2, 2), padding='same'))
cnn_model.add(Conv2D(50, (5, 5), activation='relu', strides=(2, 2), padding='same'))
cnn_model.add(MaxPool2D(pool_size=(2, 2), padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(70, (3, 3), activation='relu', strides=(2, 2), padding='same'))
cnn_model.add(MaxPool2D(pool_size=(2, 2), padding='valid'))
cnn_model.add(BatchNormalization())
cnn_model.add(Flatten())
cnn_model.add(Dense(units=100, activation='relu'))#defining output Layer
cnn_model.add(Dense(units=100, activation='relu'))
cnn_model.add(Dropout(0.25))
cnn_model.add(Dense(units=y_train.shape[1], activation='softmax'))#defining prediction Layer
#compiling, training and Loading model
cnn_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
if os.path.exists('model/cnn_weights.hdf5') == False:
    model_checkpoint = ModelCheckpoint(filepath='model/cnn_weights.hdf5', verbose = 1, save_best_only = True)
    hist = cnn_model.fit(X_train, y_train, batch_size = 32, epochs = 20, validation_data=(X_test, y_test), callbacks=[model_checkpoint])
    f = open('model/cnn_history.pkl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
else:
    cnn_model.load_weights('model/cnn_weights.hdf5')
predict = cnn_model.predict(X_test) #now perform prediction on test images
predict = np.argmax(predict, axis=1)
test_y = np.argmax(y_test, axis=1)
calculateMetrics('CNN', test_y, predict)

```

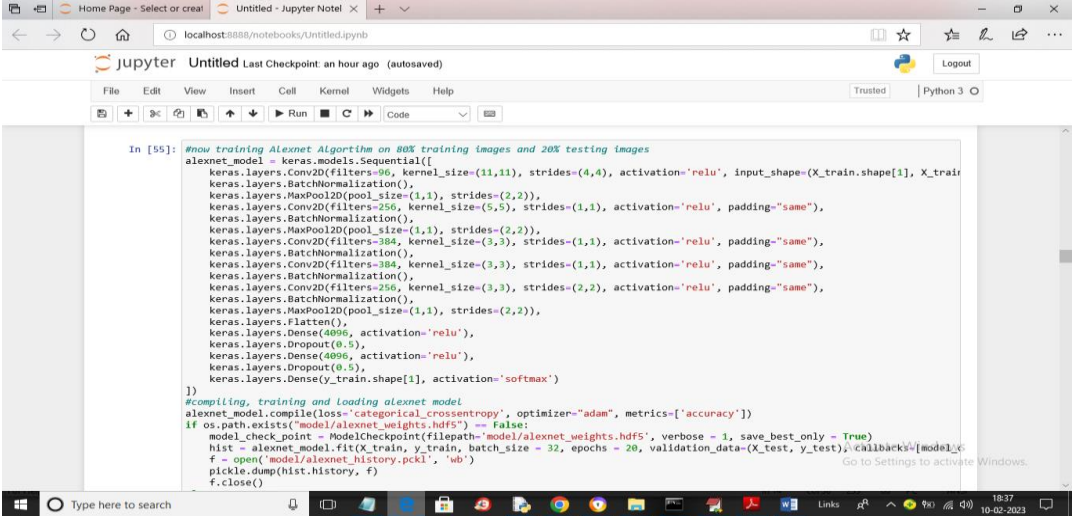
Screenshot 5.8 : In above screen we are defining and training CNN model and after Executing above block will get below output

5.9 ALEXNET ACCURACY



Screenshot 5.9 : In above screen with CNN we got 95% accuracy and we can see other Metrics like recall and FSCORE which we are getting more than 95%. In confusion matrix graph x-axis represents Predicted Labels and y-axis represents True Labels and all blue colour boxes contains incorrect prediction count which are very few. Different colour boxes in diagonal contains correct prediction count which are huge. So CNN is accurate in prediction up to 95%

5.10 ALEXNET CODE

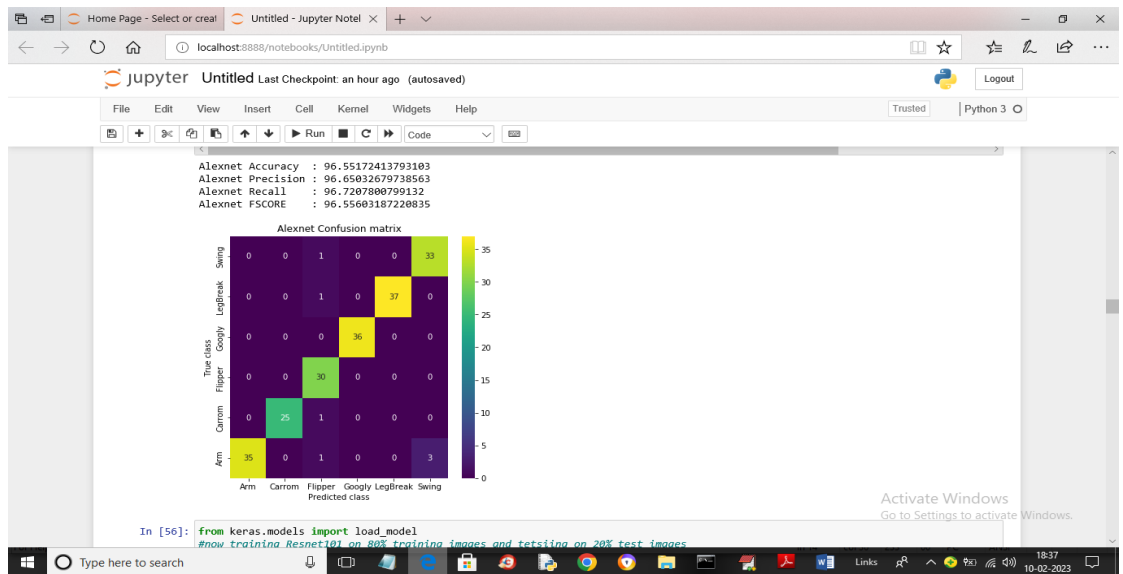


```

In [55]: #now training Alexnet Algorithm on 80% training images and 20% testing images
alexnet_model = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(2,2), activation='relu', padding='same'),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2,2), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(y_train.shape[1], activation='softmax')
])
#compiling, training and loading alexnet model
alexnet_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
if os.path.exists('model/alexnet_weights.hdf5') == False:
    model_checkpoint = ModelCheckpoint(filepath='model/alexnet_weights.hdf5', verbose = 1, save_best_only = True)
    hist = alexnet_model.fit(X_train, y_train, batch_size = 32, epochs = 20, validation_data=(X_test, y_test), callbacks=[model_checkpoint])
    f = open('model/alexnet_history.pkl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
  
```

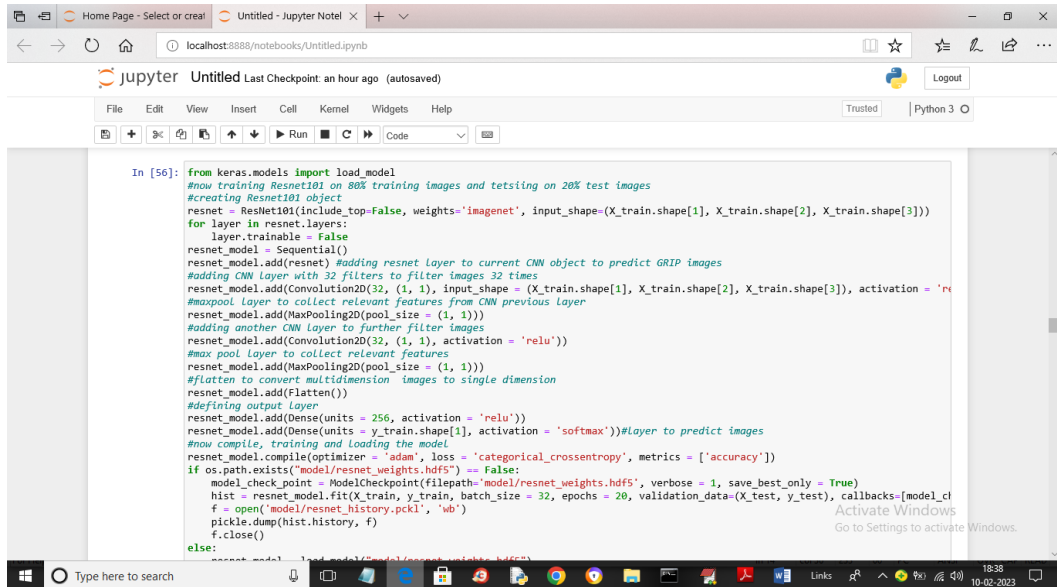
Screenshot 5.10 : In above screen showing code for ALEXNET model and after executing this algorithm will get below output

5.11 ALEXNET RESULT



Screenshot 5.11 : In above screen with ALEXNET we got 96% accuracy and we can see other metrics also

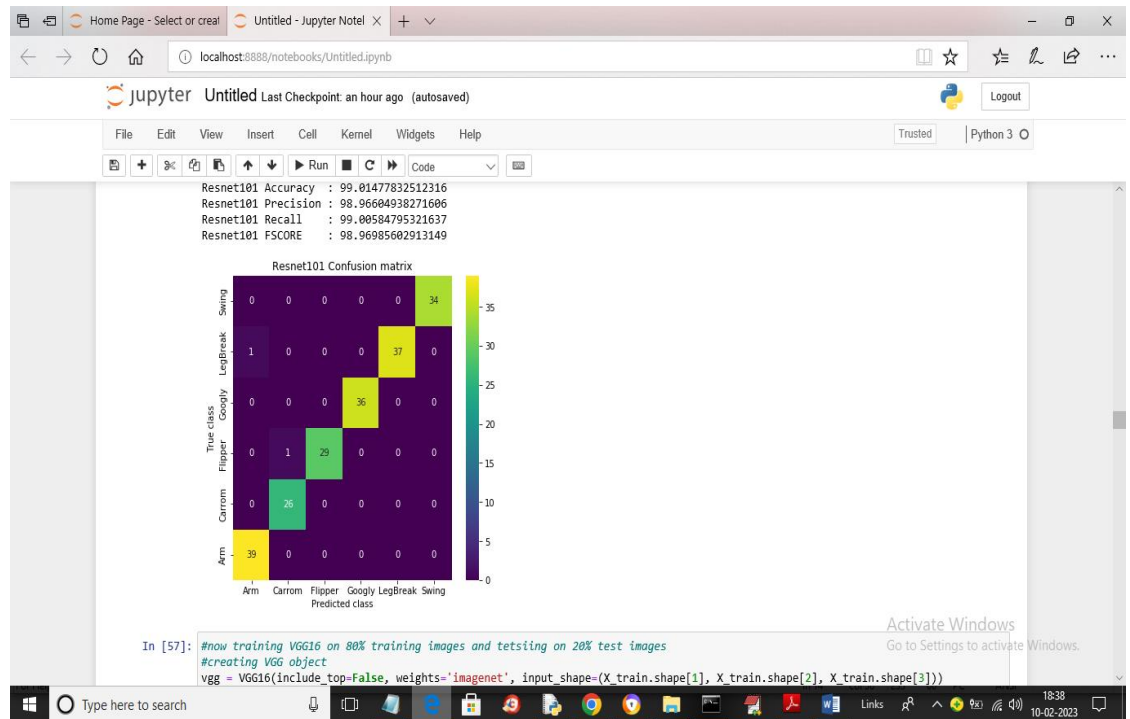
5.12 RESNET MODEL



```
In [56]: from keras.models import load_model
#now training Resnet101 on 80% training images and testing on 20% test images
#creating Resnet101 object
resnet = ResNet101(include_top=False, weights='imagenet', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3]))
for layer in resnet.layers:
    layer.trainable = False
resnet_model = Sequential()
resnet_model.add(resnet) #adding resnet Layer to current CNN object to predict GRIP images
#adding CNN layer with 32 filters to filter images 32 times
resnet_model.add(Convolution2D(32, (1, 1), input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3]), activation='relu'))
#maxpool Layer to collect relevant features from CNN previous Layer
resnet_model.add(MaxPooling2D(pool_size=(1, 1)))
#adding another CNN Layer to further filter images
resnet_model.add(Convolution2D(32, (1, 1), activation='relu'))
#max pool Layer to collect relevant features
resnet_model.add(MaxPooling2D(pool_size=(1, 1)))
#flatten to convert multidimension images to single dimension
resnet_model.add(Flatten())
#defining output Layer
resnet_model.add(Dense(units=256, activation='relu'))
resnet_model.add(Dense(units=y_train.shape[1], activation='softmax')) #Layer to predict images
#now compile, training and loading the model
resnet_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
if os.path.exists('model/resnet_weights.hdf5') == False:
    model_checkpoint = ModelCheckpoint(filepath='model/resnet_weights.hdf5', verbose=1, save_best_only=True)
    hist = resnet_model.fit(X_train, y_train, batch_size=32, epochs=20, validation_data=(X_test, y_test), callbacks=[model_checkpoint])
    f = open('model/resnet_history.pkl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
else:
    resnet_model = load_model('model/resnet_weights.hdf5')
```

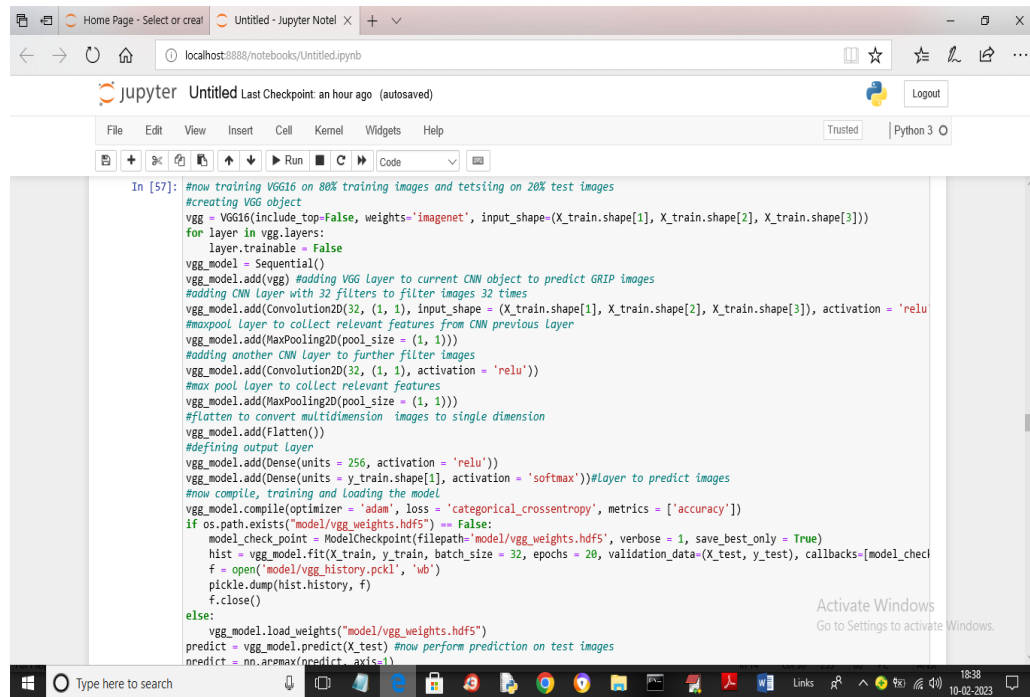
Screenshot 5.12 : In above screen we are training Resnet101 model and after executing above model will get below output

5.13 RESNET RESULT



Screenshot 5.13 : In above screen with Resnet101 we got 99% accuracy

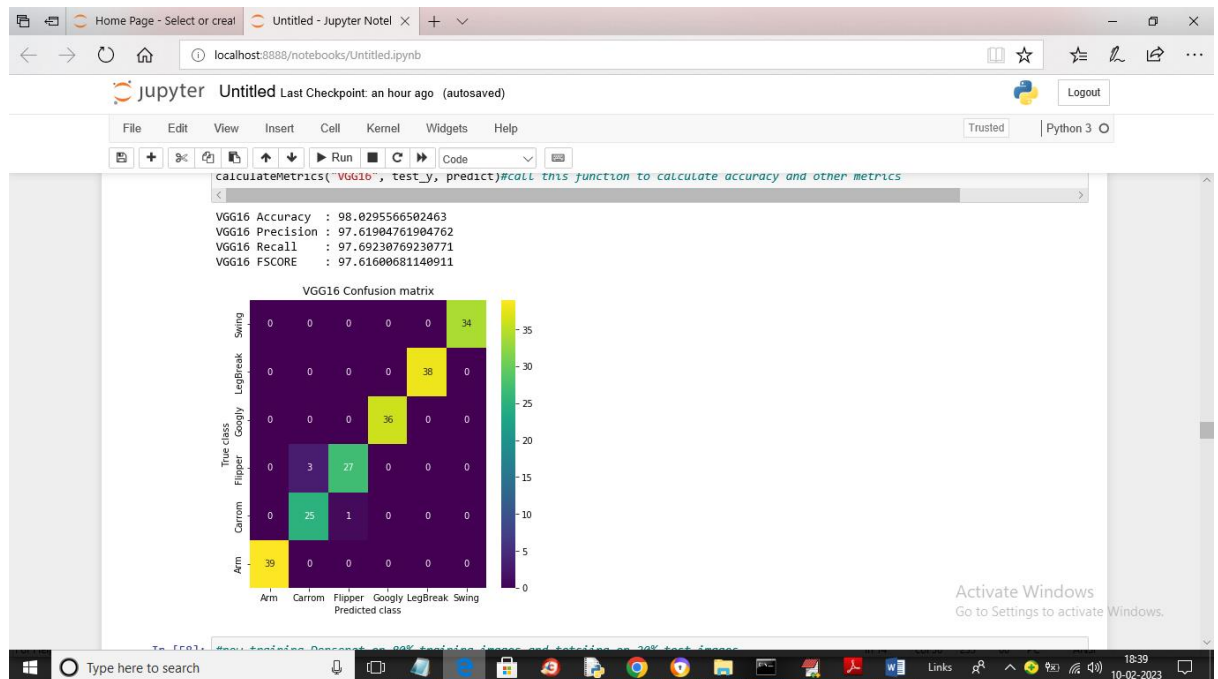
5.14 VGG16



```
In [57]: #now training VGG16 on 80% training images and testing on 20% test images
#creating VGG object
vgg = VGG16(include_top=False, weights='imagenet', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3]))
for layer in vgg.layers:
    layer.trainable = False
vgg_model = Sequential()
vgg_model.add(vgg) #adding VGG Layer to current CNN object to predict GRIP images
#adding CNN Layer with 32 filters to filter images 32 times
vgg_model.add(Convolution2D(32, (1, 1), input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3]), activation = 'relu'))
#maxpool Layer to collect relevant features from CNN previous Layer
vgg_model.add(MaxPooling2D(pool_size = (1, 1)))
#adding another CNN Layer to further filter images
vgg_model.add(Convolution2D(32, (1, 1), activation = 'relu'))
#max pool Layer to collect relevant features
vgg_model.add(MaxPooling2D(pool_size = (1, 1)))
#flatten to convert multidimension images to single dimension
vgg_model.add(Flatten())
#defining output Layer
vgg_model.add(Dense(units = 256, activation = 'relu'))
vgg_model.add(Dense(units = y_train.shape[1], activation = 'softmax')) #Layer to predict images
#now compile, training and loading the model
vgg_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
if os.path.exists("model/vgg_weights.hdf5") == False:
    model_checkpoint = ModelCheckpoint(filepath="model/vgg_weights.hdf5", verbose = 1, save_best_only = True)
    hist = vgg_model.fit(X_train, y_train, batch_size = 32, epochs = 20, validation_data=(X_test, y_test), callbacks=[model_checkpoint])
    f = open("model/vgg_history.pkl", 'wb')
    pickle.dump(hist.history, f)
    f.close()
else:
    vgg_model.load_weights("model/vgg_weights.hdf5")
predict = vgg_model.predict(X_test) #now perform prediction on test images
predicted = nn.argmax(predict, axis=-1)
```

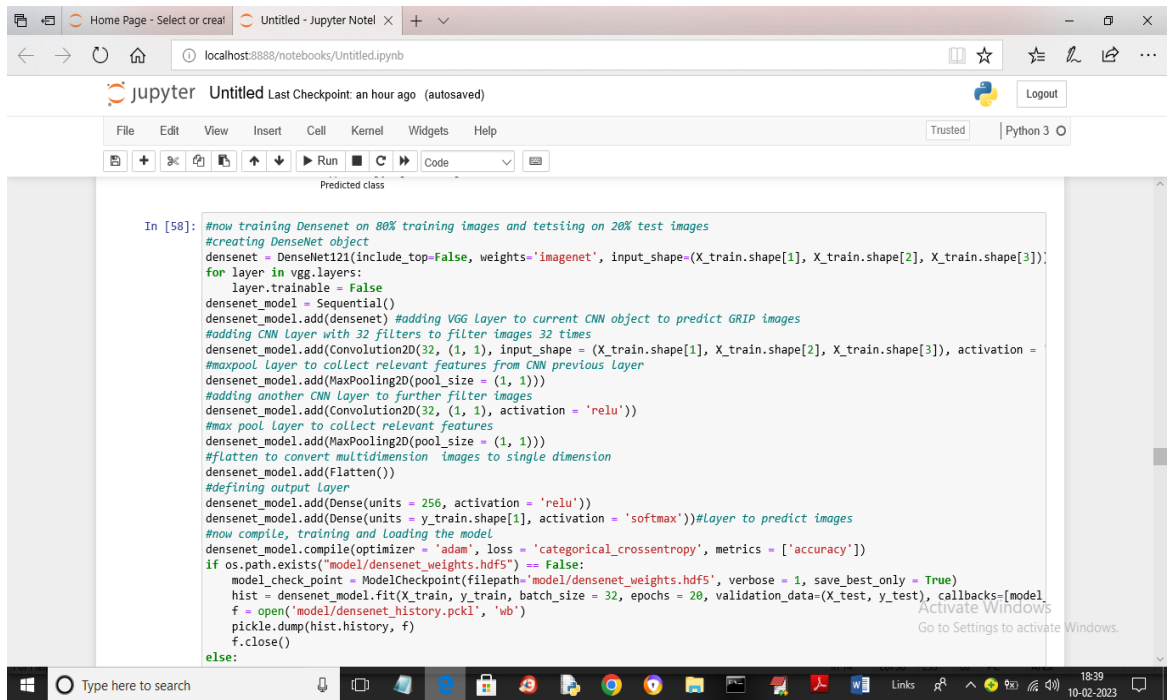
Screenshot 5.14 : In above screen we are training VGG16 algorithms and after executing above block will get below output

5.15 VGG16 RESULT



Screenshot 5.15 : In above screen with VGG16 we got 98% accuracy

5.16 DENSENET



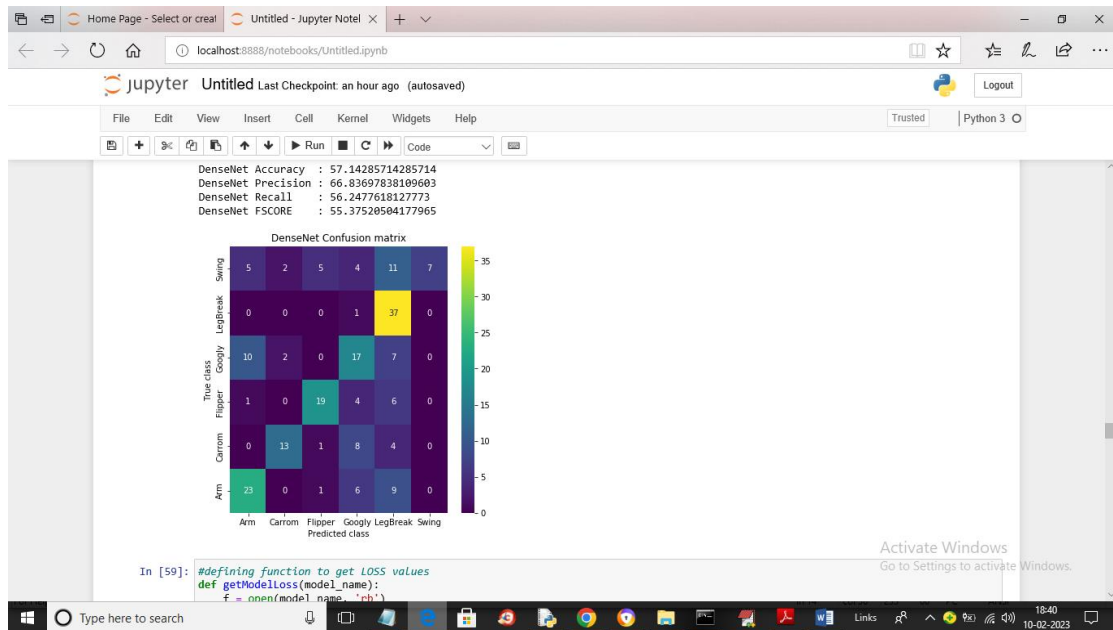
```

In [58]: #now training Densenet on 80% training images and testing on 20% test images
#creating DenseNet object
densenet = DenseNet121(include_top=False, weights='imagenet', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3]))
for layer in vgg.layers:
    layer.trainable = False
densenet_model = Sequential()
densenet_model.add(densenet) #adding VGG Layer to current CNN object to predict GRIP images
#adding CNN Layer with 32 filters to filter images 32 times
densenet_model.add(Convolution2D(32, (1, 1), input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3]), activation =
#maxpool Layer to collect relevant features from CNN previous Layer
densenet_model.add(MaxPooling2D(pool_size = (1, 1)))
#adding another CNN Layer to further filter images
densenet_model.add(Convolution2D(32, (1, 1), activation = 'relu'))
#max pool Layer to collect relevant features
densenet_model.add(MaxPooling2D(pool_size = (1, 1)))
#flatten to convert multidimension images to single dimension
densenet_model.add(Flatten())
#defining output Layer
densenet_model.add(Dense(units = 256, activation = 'relu'))
densenet_model.add(Dense(units = y_train.shape[1], activation = 'softmax'))#Layer to predict images
#now compile, training and loading the model
densenet_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
if os.path.exists("model/densenet_weights.hdf5") == False:
    model_check_point = ModelCheckpoint(filepath="model/densenet_weights.hdf5", verbose = 1, save_best_only = True)
    hist = densenet_model.fit(X_train, y_train, batch_size = 32, epochs = 20, validation_data=(X_test, y_test), callbacks=[model_check_point])
    f = open('model/densenet_history.pkl', 'wb')
    pickle.dump(hist.history, f)
    f.close()
else:

```

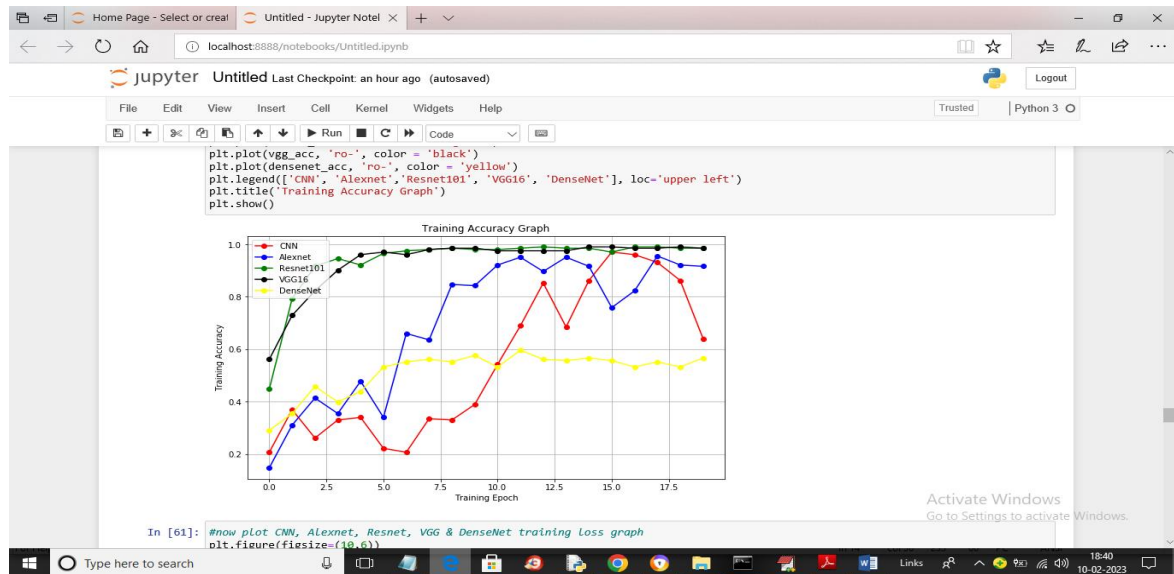
Screenshot 5.16 : In above screen we are training DenseNet121 algorithm and after this algorithm will get below output

5.17 DENSENET RESULT



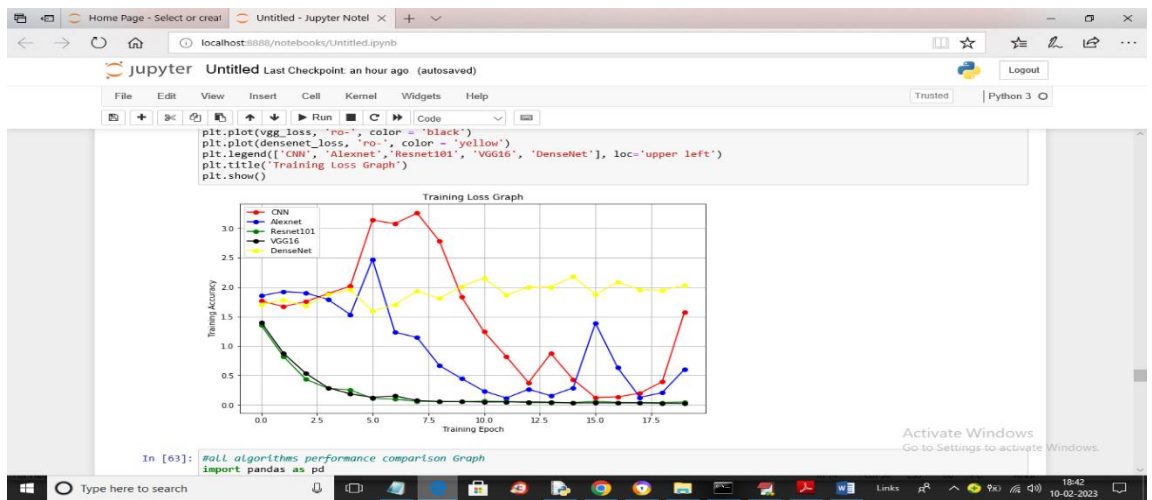
Screenshot 5.17 : In above screen with DenseNet we got 57% accuracy

5.18 GRAPH PLOTTING



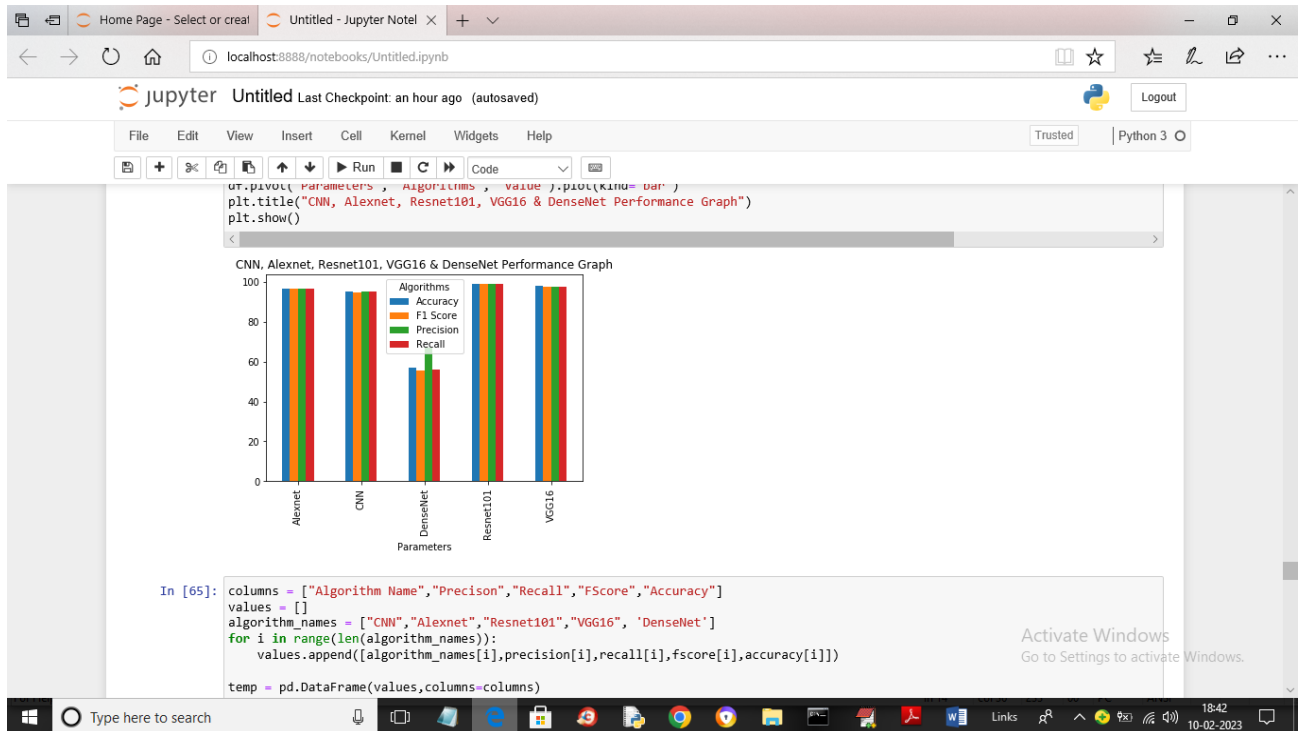
Screenshot 5.18 : In above graph we are showing training accuracy for each algorithm and each different colour line represents accuracy of different algorithm where x-axis represents training epoch and y-axis represents accuracy and with each increasing epoch accuracy get increased

5.19 GRAPH LOSS



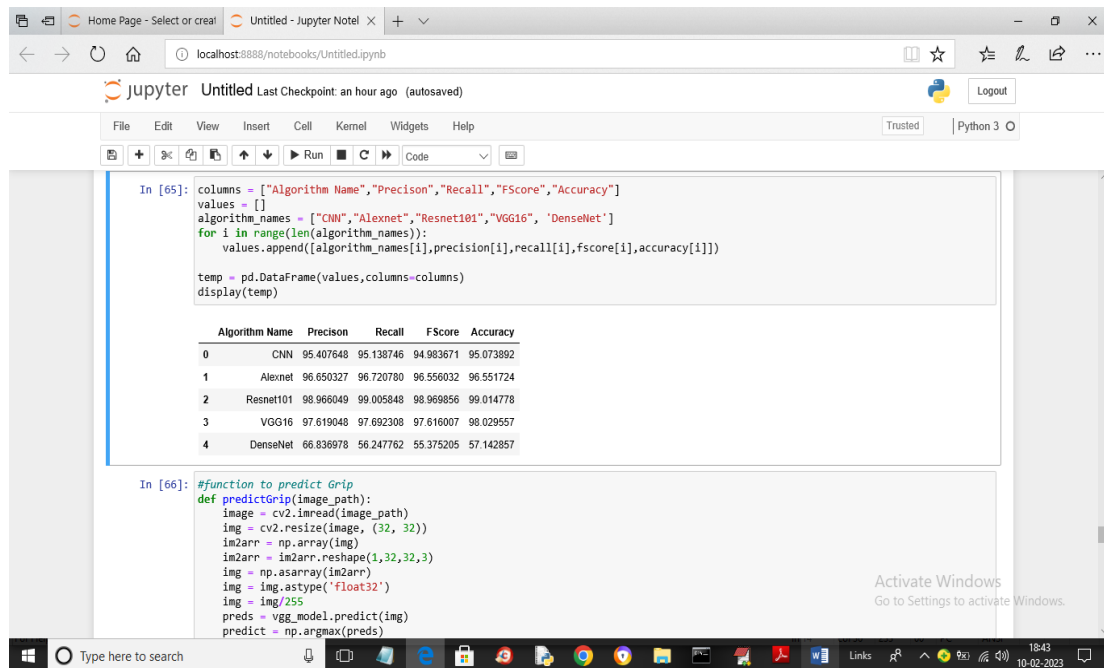
Screenshot 5.19 : In above loss graph with each increasing epoch loss got decreased

5.20 GRAPH PLOTTING



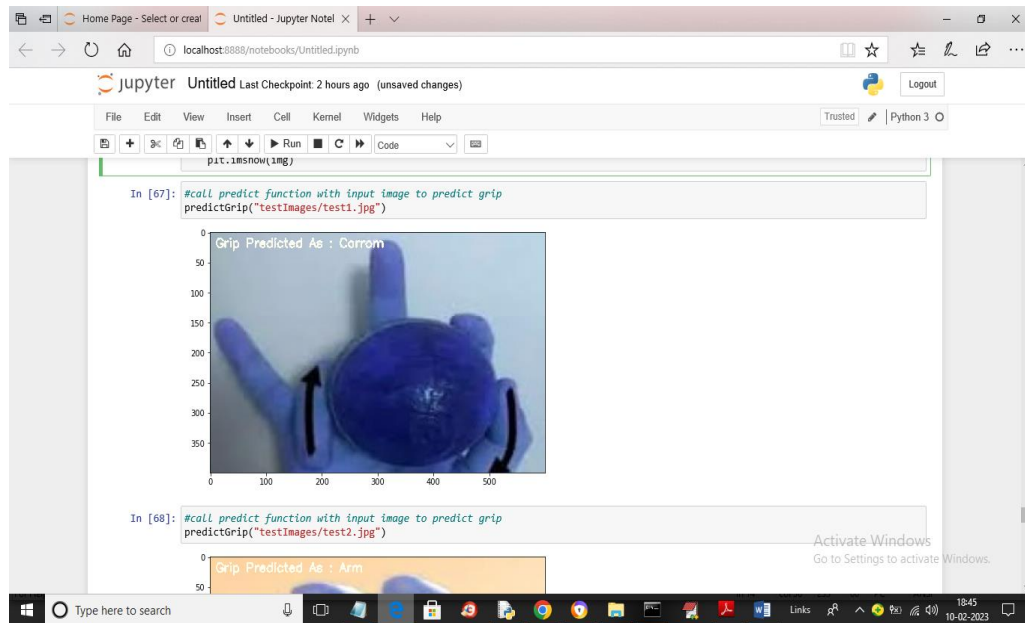
Screenshot 5.20: In above graph x-axis represents algorithm names and y-axis represents different metrics such as accuracy, precision, recall and FSCORE in different and in above graph we can see 4 algorithms has achieved accuracy of more than 95%

5.21 PERFORMANCE



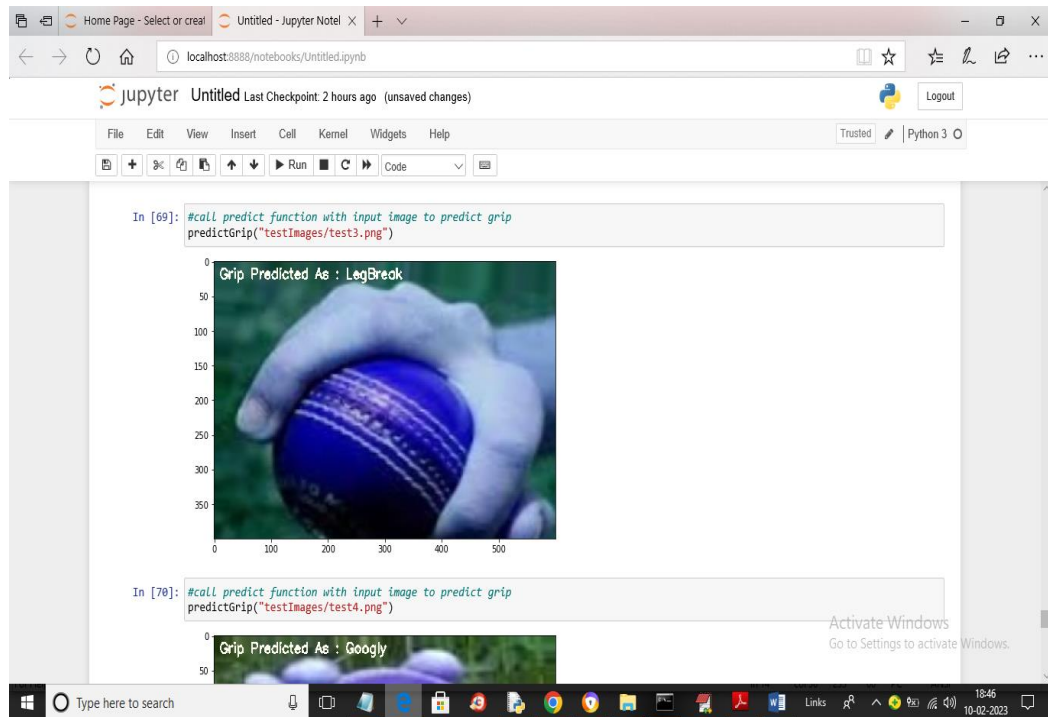
Screenshot 5.21 : In above screen we can see each algorithm performance in tabular format

5.22 GRIP PREDICTION



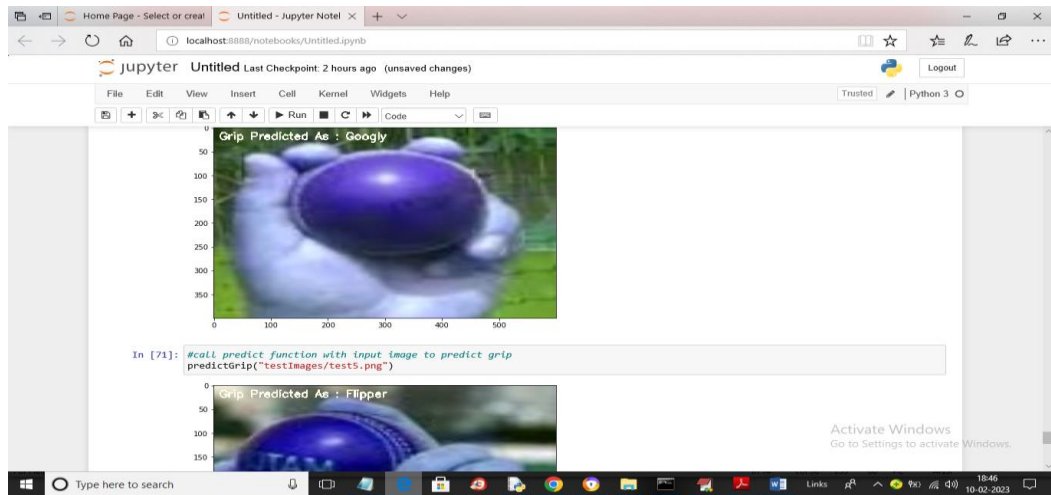
Screenshot 5.22 : In above screen grip predicted as ‘Carrom’ which you can see in white color text

5.23 GRIP PREDICTED



Screenshot 5.23 : In above screen grip predicted as Leg Break

5.24 PREDICTED OUTPUT



Screenshot 5.24: So in above screen we can see other predicted grip output

6. TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.2 TYPES OF TESTING

6.2.1 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

6.2.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes.

7. CONCLUSION AND FUTURE SCOPE

7. CONCLUSION & FUTURE SCOPE

7.1 PROJECT CONCLUSION

In this research a unique strategy was proposed to identify different types of delivery in cricket from offline real-time videos of cricket bowling. A new deep CNN model which was used as a preliminary model to train the dataset showed outstanding accuracy and also compared the model performance with several existing pre-trained transfer learning models. Furthermore, a completely new dataset that consists of over 5000 images, categorized into 13 different deliveries in cricket bowling, was also introduced in the process of this research. This research is likely to be very useful for cricket players and coaches for training with video analysis and also for TV broadcaster of live cricket match to introduce a new technology in their live broadcast. Finally, this research is also expected to serve as a motivation for researchers to apply deep learning to explore various actions and activities related to sports.

7.2 FUTURE SCOPE

In the future scope of this research is wide that may include wrong action detection in a live match, automatic commentary text generation related to bowling action and automatic player performance evaluation.

8. BIBLIOGRAPHY

8. BIBLIOGRAPHY

8.1 REFERENCES

- [1] H. Chen, Y. Zhang, M. K. Kalra, F. Lin, Y. Chen, P. Liao, J. Zhou and G. Wang, "Low-dose CT with a residual encoder-decoder convolutional neural network," IEEE Transactions on Medical Imaging, pp. 2524-2535, 2017
- [2] M. Anthimopoulos, S. Christodoulidis, L. Ebner, A. Christer and S. Mougiakakou, "Lung pattern classification for interstitial lung diseases using a deep convolutional neural network," IEEE transactions on medical imaging, vol. 35, no. 5, pp. 1207-1216, 2016.

8.2 GITHUB LINK

<https://github.com/Vishaljakkam/DeepGrip-Cricket-Bowling-Delivery-Detection-with-Superior-CNN-Architectures/blob/main/Code>