



IJITCE

ISSN 2347- 3657

International Journal of Information Technology & Computer Engineering

www.ijitce.com



Email : ijitce.editor@gmail.com or editor@ijitce.com

PREDICTING ACCURACY OF PLAYERS IN THE CRICKET USING MACHINE LEARNING

D.PUSHPA¹, PURELLA MEGHANA², POTLAPALLY JAHNAVI³, MADKA VARUN⁴, RISHIKA AGARWAL⁵, MODI

^{2,3,4,5,6} UG Students, Dept of CSE, MALLA REDDY INSTITUTE OF ENGINEERING AND TECHNOLOGY(AUTONOMOUS), Dhulapally, Secundrabad, Hyderabad, Telangana, India.

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. The paper 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, DenseNet, MobileNet, AlexNet, Inception V3, and NasNet 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.

Keywords: VGG16, VGG19, Cricket Bowling and CNN

1. INTRODUCTION

The potentiality of deep learning has reached a new horizon. In recent years, significant increase in research on different fields using Convolution Neural Network (CNN) has been noticed. One remarkable factor of it is that CNN has remarkable accuracy in performing characteristics are extracted by the initial convolution layer forwarded to the next layer. Thus its architecture

complicated classification by recognizing the complexities in images. Another most notable factor is the convolution operation that it performs over images using different filters. Invariant

provides the ability to generate the feature vector for further analysis

Not only in academia, but the use of CNN has also gone beyond it. CNN has already shown its potential for huge improvements in the field of medical imaging. For instance, its application in processing reconstructed images in low-cost CT [1] and lung pattern classification from CT scan images . In addition to that, massive companies such as Google, Microsoft, etc have their researchers exploring more about deep learning. The performance of AlexNet in the 2012 ImageNet competition [4] was significant. CNN architectures are uplifting major progress in Computer Vision (CV) which increased its applications in fields like robotics, drones, security, and autonomous cars. Many Artificial Intelligence researchers, in recent times, are trying to apply deep learning in Cricket: a sophisticated sport providing researchers notable volumes of visual and statistical data to work with.

Cricket has strong infrastructures and outstanding popularity in many South Asian countries such as India, Pakistan, Bangladesh, etc and in other parts of the world like England, Australia, New Zealand, etc. Its popularity has established itself as a particular sector in which many experts and researchers are spending efforts to analyze different fields of it. Artificial intelligence and Data Science have already created huge impacts and are being widely used in Cricket .

Moreover, many electronic sports broadcasters and newspapers always crave for modern technology to analyze this game for their consumers. In modern-day cricket, the finger grip on the bowl that a cricket bowler uses play a crucial role in the outcome of the delivery being bowled. And the enigmatic behavior of the ball bowled by the bowler towards the batsman is significantly dependent on the grip of the ball . Even world-class cricket batsmen like Virat Kohli, Sachin Tendulkar had missed the trick of the ball on numerous

occasions. So it is immensely important to understand the grip on the ball a bowler can use. So this research was intended to have major positive impacts in the following ways:

1. This system will be beneficial for live cricket match television broadcasters to show the type of grips used by a bowler, during a live match, for their viewers. In the latter portion of this paper, it is demonstrated how the proposed system can instantly detect grips from live webcam videos which eventually supports this claim. Moreover, the high accuracy of our system to correctly identify those 13 grips is also good evidence. Thus, this work may attract television broadcasters to incorporate this new technology in their broadcasting system and to get an upper hand among their competitors by presenting a much better informative visual of cricket match for their viewers.

2. This system will help to analyze the variety of grip used by a new bowler in the opponent team that is difficult to play. In other words, while training, it is often difficult to analyze a new bowler from the opposing team, basically when they are performing well since there is a scarcity of videos of the bowler for video analysis. Moreover, it is also not feasible to analyze the videos manually with the naked eye. So the proposed scheme will help to identify the types of grip within those 13 classes of grips automatically and effectively from the limited number of recorded videos of that particular bowler available. As a result, this system is expected to be a good resource for players while training by video analysis before a match. This paper presents a high-quality implementation of different convolutional architectures to categorically identify the grip of a cricket bowler while bowling a delivery.

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, MobileNet, AlexNet, NasNet, ResNet 101, ResNet 152 and DenseNet were used to get trained with GRIP DATASET which consists of 5573 image frames extracted and augmented artificially from videos. This study is expected to help researchers to apply deep learning image processing potential to explore cricket bowling and also provide a better method for experts in cricket to analyze intelligently with Artificial Intelligence.

2. LITERATURE SURVEY

1. CNN for Classification of Sports

There are many successful works with CNN for various sports classification tasks. For instance, proposed a model that is based on Alex Net CNN to classify shots into various types from cricket and soccer videos. Their model showed an accuracy of around 94% which is reported to be better than the accuracy obtained using K-Nearest Neighbors, Support Vector Machine, Extreme Learning Machine and Convolution Neural Network. , in contrast, developed a Deep CNN model to classify movements of beach volleyball players based on wearable sensor data. Their model had better accuracy compared to five more classification algorithm they used which suggest that the model is an efficient approach to recognize the sensor-based activity. Likewise, many other works related to sports player detection , ball detection , and match highlights detection are also available.

Grip Angle Effect on the Outcome of Spin Bowling

In this paper , research was performed where they used a sensor attached cricket ball to study the outcome of grip angle parameter on off-spin bowling performance. Their main purpose was to scrutinize whether a standard grip on the ball is better than narrow and wide grips. According to the information of the smart ball they claimed that standard grip is more productive compared to narrow and wide grip in terms of their bowling performance parameter.

Identification of cricket batting shot from videos with deep learning.

This paper, used saliency and optical flow to get static and dynamic cues and on Deep CNN for extracting representations. They also have stated that their model gives better detection accuracy when they used a multiple class based SVM. Another paper suggested an analogous approach to analyze videos in order to identify batting shots in cricket using deep CNN where they trained their models using LSTM along with 2D CNN and also 3D. According to the result of their experiment, the 3D model had more accuracy compared to the 2D model on their validation data. Their dataset basically consist of around 800 short video clips

Umpire pose detection with deep learning

The following research , introduced a unique dataset, called SNOW, for umpire posture detection in cricket which may eventually help developing automatic cricket match highlights generation. They had classified four cricket events namely Six, Wide, No ball, and Out by capturing the action of the umpire from the frames of cricket videos. To extract features they used Inception V3 and VGG 19 and obtained results using an SVM classifier. In their experiment, VGG 19 showed better accuracy for classification compared to that of Inception V3.

Cricket outcome classification with deep learning

This research has used the CNN and LSTM Networks for the outcome classification of cricket. The test accuracy for their classification was 70%. They also made their dataset from collected video clips.

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, MobileNet, AlexNet, NasNet, ResNet 101, ResNet 152 and DenseNet were used to get trained with GRIP DATASET which consists of 5573 image frames extracted and augmented artificially from videos. This study is expected to help researchers to apply deep learning image processing potential to explore cricket bowling and also provide a better method for experts in cricket to analyze intelligently with Artificial Intelligence.

3. PROPOSED SYSTEM

In proposed, we are training different deep learning algorithms such as CNN, Alexnet, Resnet101, VGG16 and DenseNet to predict cricket ball Grip Type such as 'Arm, Swing, Carom, Flipper, Leg Break and Googly. In paper author has used 5000 images with 13 different grips but this dataset is not available on internet so we created 6 grips dataset by downloading images from Google. Even I mailed to paper authors to send dataset but they are not replying so I downloaded own images for 6 different grips and dataset contains more than 1000 images

MODULES

Collection of DataSet

The dataset, that was developed consists of frames extracted from different Real-Time Videos of various bowling experts in offline-mode from YouTube. There are 13 classes of bowling grip images - Inswing, OutSwing, Leg Break, Knuckle, Googly, Doosra, Flipper, Arm, Carrom, Slider, Top Spin, Leg Cutter and Off Cutter respectively. Videos where different bowlers and experts have shown the grips of different bowling techniques were considered. A python script was written using the OpenCV framework to extract the frames from videos automatically. The images were artificially augmented and used the zoom to focus on the grips of the bowlers. Keras Image Generator were used for this image augmentation, which, are set-up python generators that convert images to preprocessed tensors to be passed into our models.

Preliminary CNN

It shows a sequential model possesses two convolution layers at the beginning of 250 and 250 kernels respectively with dimensions of [3 X 3] and single stride. The Max Pooling layer of dimensions [3 X 3] then follows these convolution layers to manipulate the size of the input to be passed forward. Then following four layers are convolution layers with 150, 150, 100 and 100 kernels respectively of the same dimensions as in first 2 layers and then again Max Pooling of [3 X 3] followed the convolution layers. Finally, the output is flattened and then input into the Fully Connected Network of 100 neurons in the two hidden layers each and then finally through the Softmax Output Layer of 13 neurons according to the grip classes

VGG16 and VGG19

VGG16 is convolutional neural networks of which the architecture is highly successful in the analysis of visual data like images. The images are converted to [224 X 224] and are input in the VGG16 model. The kernels are of [3 X 3] dimensions with a depth of 64 in the first two convolution layers, 128 in fourth and fifth layers of convolution, 256 in seventh, eighth, and ninth layers of convolution and 512 in eleventh, twelfth and thirteenth layers all with strides of 1. The Max Pooling Layers in third, sixth, tenth, fourteenth and eighteenth possess a dimension of 2X2 and stride of 2. Then the output is flattened and inserted into 4096 units of hidden layers that are connected followed by a layer in the output which consist of softmax activation function

NasNet

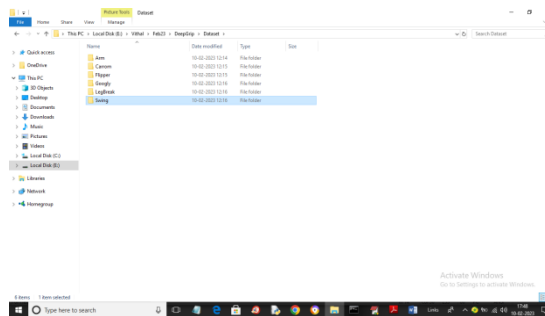
NasNet is a CNN architecture which is composed of cells that are enhanced by reinforcement learning [1]. The ImageNet database is used to train NasNet-Large with more than one million images and it can classify 1000 categories of objects. For our research purpose, the NasNet from Keras Applications were used with pre-trained weights. The input size of the images in the NasNet-Large model is [331 X 331].

4. RESULT ANALYSIS

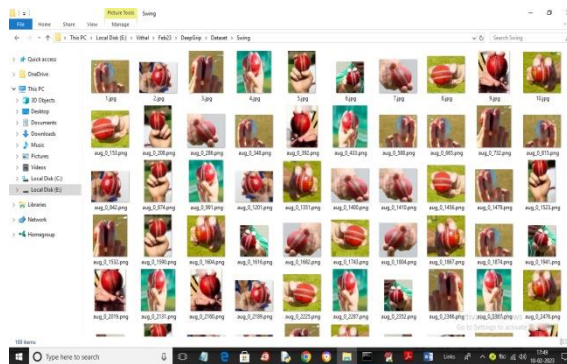
DeepGrip: Cricket Bowling Delivery Detection with Superior CNN Architectures

In this project we are training different deep learning algorithms such as CNN, Alexnet, Resnet101, VGG16 and DenseNet to predict cricket ball Grip Type such as 'Arm, Swing, Carom, Flipper, Leg Break and Googly. In paper author has used 5000 images with 13 different grips but this dataset is not available on internet so we created 6 grips dataset by downloading images from Google.

Even I mailed to paper authors to send dataset but they are not replying so I downloaded own images for 6 different grips and dataset contains more than 1000 images. In below screen we are showing dataset details

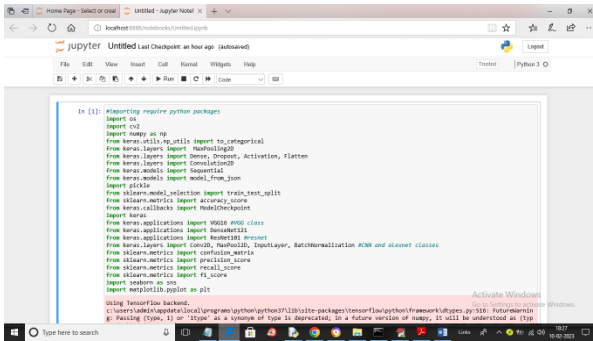


In above Dataset screen we have 6 different folders and just go inside any folder to view images like below screen



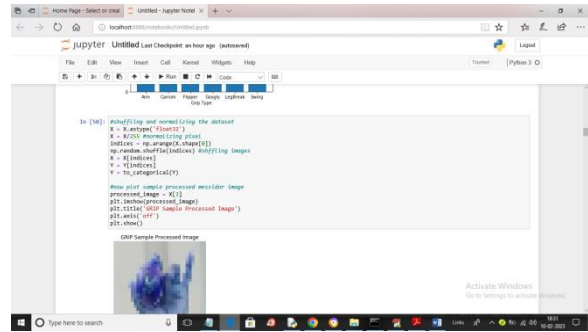
So by using above images we are training all deep learning algorithms and evaluating their performance in terms of accuracy, precision, recall, FSCORE and confusion matrix.

We have coded this project using JUPYTER notebook and below are the code and output screens with blue colour comments



```
In [1]: #Importing require python packages
import cv2
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.callbacks import Sequential
from keras.models import Model, Sequential
import sklearn
from sklearn.metrics import accuracy_score
from sklearn.metrics import MeanAbsoluteError
import keras
from keras.applications import VGG16, VGG19, InceptionV3
from keras.applications import ResNet50
from keras.layers import Conv2D, MaxPooling2D, InceptionV3, BatchNormalization, Flatten and GlobalAveragePooling2D
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
import seaborn as sns
import matplotlib.pyplot as plt
```

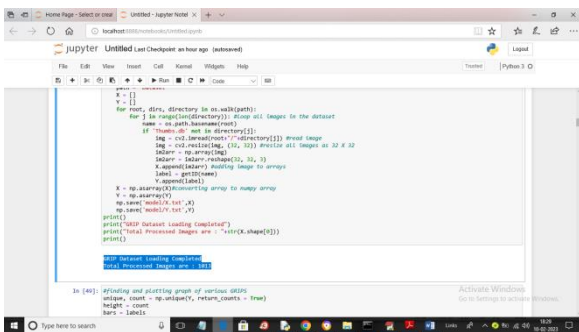
In above screen we are importing all require python classes and packages and you can read blue colour comments to know about classes



```
In [10]: #Loading and normalizing the dataset
x = x.astype('float32')
x /= 255. #normalizing pixel values
indices = np.arange(x.shape[0])
np.random.shuffle(indices)
x = x[indices]
y = y[indices]
y = to_categorical(y)

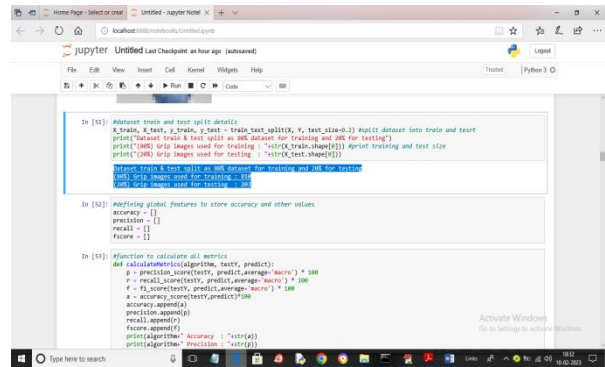
#now plot sample processed image (image processed_image = x[1])
plt.imshow(processed_image)
plt.title('GRIP Sample Processed Image')
```

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



```
for root, dirs, files in os.walk(path):
    for f in files:
        if f.endswith('.jpg'):
            img = cv2.imread(os.path.join(root, f))
            if img is not None:
                #resizing image
                img = cv2.resize(img, (32, 32))
                #adding X and Y array
                X.append(img)
                #getting labels
                labels.append(f.split('.')[0])
            else:
                print("Error loading image: ", f)
                continue
            #converting image to numpy array
            x = np.array(X)
            y = np.array(labels)
            #reshaping array to numpy array
            x = x.reshape((x.shape[0], x.shape[1], x.shape[2], 3))
            y = y.reshape((y.shape[0],))
            print("GRIP Dataset Loading Completed")
            print("Total processed images are: ", x.shape[0])
            print()
```

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

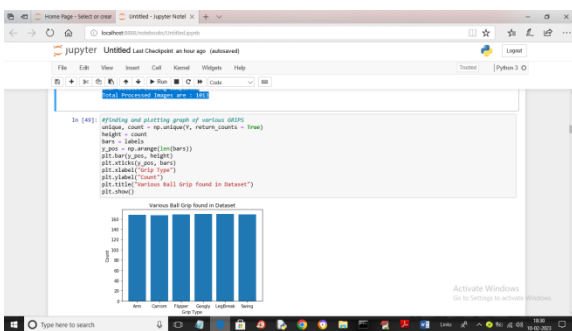


```
In [11]: #dataset train and test split details
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
print("dataset train & test split as 80% dataset for training and 20% for testing")
print("Total GRIP images used for training: ", x_train.shape[0])
print("Total GRIP images used for testing: ", x_test.shape[0])

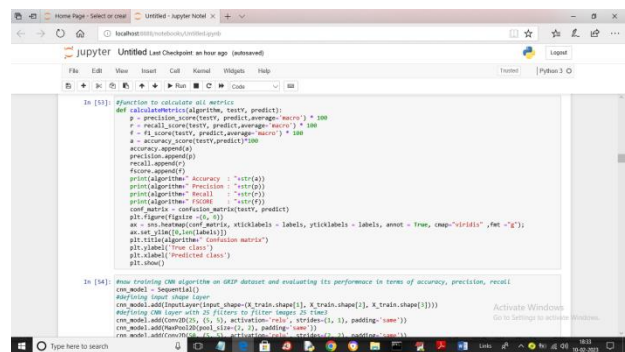
In [12]: #splitting global features to store accuracy and other values
accuracy = []
precision = []
recall = []
f1_score = []

In [13]: #function to calculate all metrics
def calculate_metrics(algorithm, test, predict):
    p = precision_score(test, predict, average='macro') * 100
    r = recall_score(test, predict, average='macro') * 100
    a = accuracy_score(test, predict) * 100
    accuracy.append(p)
    precision.append(r)
    recall.append(a)
    f1_score.append(f1_score(test, predict))
    print(algorithm, Accuracy: ", str(p))
    print(algorithm, Precision: ", str(r))
    print(algorithm, Recall: ", str(a))
    print(algorithm, F1Score: ", str(f1_score))
    conf_matrix = confusion_matrix(test, predict)
    plt.figure(figsize=(8, 8))
    ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap='magma', mt="Y")
    ax.set_xticklabels(x_labels)
    ax.set_yticklabels(y_labels)
    plt.xlabel('True class')
    plt.ylabel('Predicted class')
    plt.show()
```

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



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



```
In [13]: #function to calculate all metrics
def calculate_metrics(algorithm, test, predict):
    p = precision_score(test, predict, average='macro') * 100
    r = recall_score(test, predict, average='macro') * 100
    a = accuracy_score(test, predict) * 100
    accuracy.append(p)
    precision.append(r)
    recall.append(a)
    f1_score.append(f1_score(test, predict))
    print(algorithm, Accuracy: ", str(p))
    print(algorithm, Precision: ", str(r))
    print(algorithm, Recall: ", str(a))
    print(algorithm, F1Score: ", str(f1_score))
    conf_matrix = confusion_matrix(test, predict)
    plt.figure(figsize=(8, 8))
    ax = sns.heatmap(conf_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap='magma', mt="Y")
    ax.set_xticklabels(x_labels)
    ax.set_yticklabels(y_labels)
    plt.xlabel('True class')
    plt.ylabel('Predicted class')
    plt.show()

In [14]: #now training CNN algorithm on GRIP dataset and evaluating its performance in terms of accuracy, precision, recall
cm_model = Sequential()
cm_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
cm_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
cm_model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
cm_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
cm_model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
cm_model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
cm_model.add(Flatten())
cm_model.add(Dense(1000, activation='relu'))
cm_model.add(Dense(10, activation='softmax'))
```

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

```
In [14]: #How training CNN algorithm on CIFAR-100 dataset and evaluating its performance in terms of accuracy, precision, recall
import numpy as np
from tensorflow.keras import layers, models, metrics
from tensorflow.keras.preprocessing.image import ImageDataGenerator

#Data Generator
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True)

test_datagen = ImageDataGenerator(
    rotation_range=0,
    width_shift_range=0,
    height_shift_range=0,
    shear_range=0,
    zoom_range=0,
    horizontal_flip=False,
    vertical_flip=False)

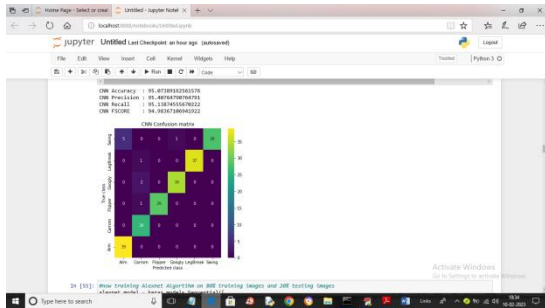
train_data = train_datagen.flow_from_directory('data/train',
                                             class_mode='categorical',
                                             batch_size=32)
test_data = test_datagen.flow_from_directory('data/test',
                                             class_mode='categorical',
                                             batch_size=32)

#Model Architecture
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(64, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(128, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(256, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Flatten(),
    layers.Dense(1000, activation='relu'),
    layers.Dense(100, activation='softmax')])

#Compile and Train
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_data, validation_data=test_data, epochs=30, verbose=1, save_best_only=True)

#Evaluate
accuracy = model.evaluate(test_data)
print('Accuracy: %f' % accuracy)
```

In above screen we are defining and training CNN model and after executing above block will get below output



In above screen with CNN we got 95% accuracy and we can see other metrics like precision, 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%

```
In [15]: #How training AlexNet algorithm on CIFAR-100 training images and testing on CIFAR-100 test images
import numpy as np
from tensorflow.keras import layers, models, metrics
from tensorflow.keras.preprocessing.image import ImageDataGenerator

#Data Generator
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True)

test_datagen = ImageDataGenerator(
    rotation_range=0,
    width_shift_range=0,
    height_shift_range=0,
    shear_range=0,
    zoom_range=0,
    horizontal_flip=False,
    vertical_flip=False)

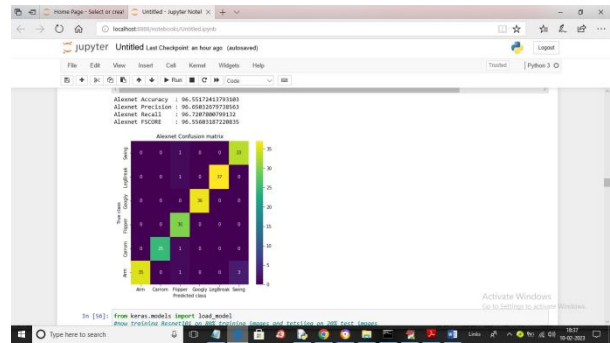
train_data = train_datagen.flow_from_directory('data/train',
                                             class_mode='categorical',
                                             batch_size=32)
test_data = test_datagen.flow_from_directory('data/test',
                                             class_mode='categorical',
                                             batch_size=32)

#Model Architecture
model = models.Sequential([
    layers.Conv2D(48, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(96, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(192, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(128, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(256, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Flatten(),
    layers.Dense(2048, activation='relu'),
    layers.Dense(2048, activation='relu'),
    layers.Dense(100, activation='softmax')])

#Compile and Train
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_data, validation_data=test_data, epochs=30, verbose=1, save_best_only=True)

#Evaluate
accuracy = model.evaluate(test_data)
print('Accuracy: %f' % accuracy)
```

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



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

```
In [16]: #How training Resnet101 algorithm on CIFAR-100 training images and testing on CIFAR-100 test images
import numpy as np
from tensorflow.keras import layers, models, metrics
from tensorflow.keras.preprocessing.image import ImageDataGenerator

#Data Generator
train_datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True)

test_datagen = ImageDataGenerator(
    rotation_range=0,
    width_shift_range=0,
    height_shift_range=0,
    shear_range=0,
    zoom_range=0,
    horizontal_flip=False,
    vertical_flip=False)

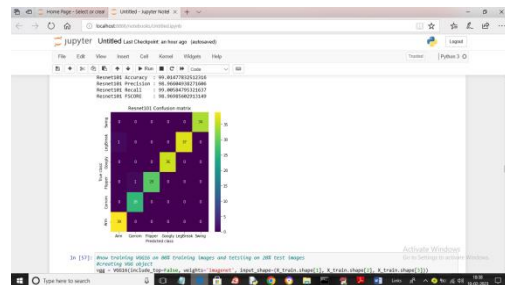
train_data = train_datagen.flow_from_directory('data/train',
                                             class_mode='categorical',
                                             batch_size=32)
test_data = test_datagen.flow_from_directory('data/test',
                                             class_mode='categorical',
                                             batch_size=32)

#Model Architecture
model = models.Sequential([
    layers.Conv2D(64, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(128, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(256, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(512, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Conv2D(1024, (3, 3), activation='relu', input_shape=(X_train.shape[1], X_train.shape[2], X_train.shape[3])),
    layers.MaxPooling2D((2, 2), activation='relu'),
    layers.Flatten(),
    layers.Dense(1000, activation='relu'),
    layers.Dense(100, activation='softmax')])

#Compile and Train
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_data, validation_data=test_data, epochs=30, verbose=1, save_best_only=True)

#Evaluate
accuracy = model.evaluate(test_data)
print('Accuracy: %f' % accuracy)
```

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

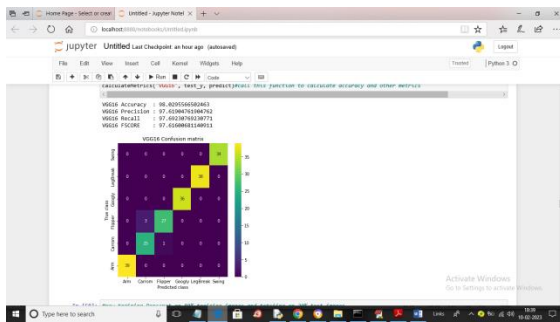


In above screen with Resnet101 we got 99% accuracy


```

In [17]: Now training VGG16 on CIF-100 training images and testing on CIF-100 test images
importing DenseNet subject
DenseNet = DenseNet121(top=False, weights='imagenet', input_shape=(X_train.shape[1], X_train.shape[1], X_train.shape[3]))
for layer in vgg.layers:
    layer.trainable = False
    vgg_model.add(DenseNet)
    vgg_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # Training VGG16
    vgg_model.fit(X_train, y_train, batch_size=32, epochs=30, validation_data=(X_test, y_test), callbacks=[model_checkpoint_callback])
    # Saving VGG16 weights
    vgg_model.save_weights('vgg_weights.h5')
    # Loading VGG16 weights
    vgg_model.load_weights('vgg_weights.h5')
    # Predicting on test images
    vgg_model.predict(X_test)
    
```

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

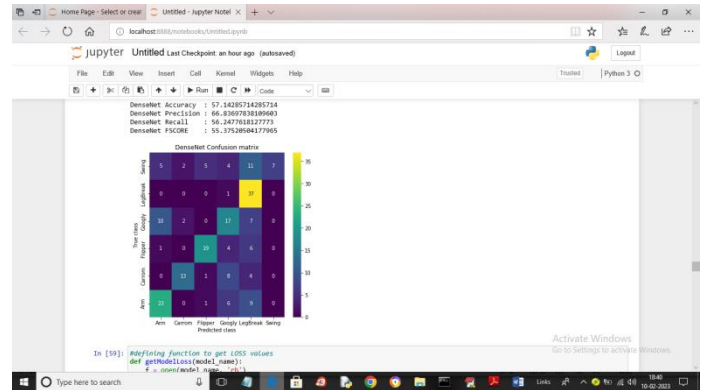


In above screen with VGG16 we got 98% accuracy

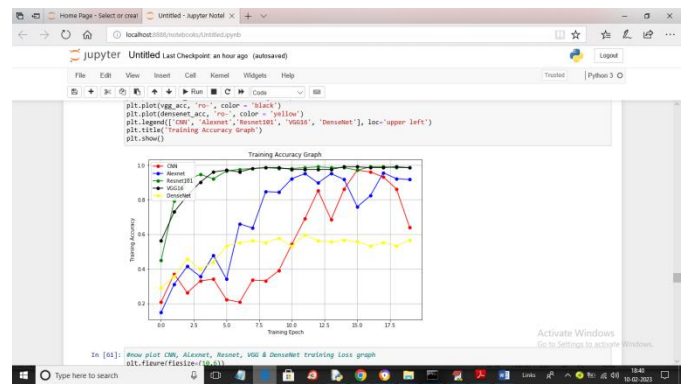
```

In [18]: Now training DenseNet on CIF-100 training images and testing on CIF-100 test images
importing DenseNet subject
DenseNet = DenseNet121(top=False, weights='imagenet', input_shape=(X_train.shape[1], X_train.shape[1], X_train.shape[3]))
for layer in densenet.layers:
    layer.trainable = False
    densenet_model.add(DenseNet)
    densenet_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # Training DenseNet
    densenet_model.fit(X_train, y_train, batch_size=32, epochs=30, validation_data=(X_test, y_test), callbacks=[model_checkpoint_callback])
    # Saving DenseNet weights
    densenet_model.save_weights('densenet_weights.h5')
    # Loading DenseNet weights
    densenet_model.load_weights('densenet_weights.h5')
    # Predicting on test images
    densenet_model.predict(X_test)
    
```

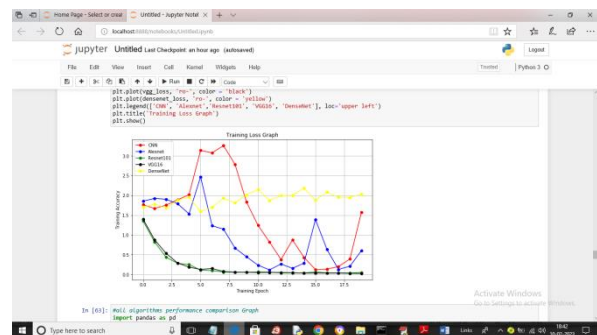
In above screen we are training DenseNet121 algorithm and after executing this algorithm will get below output



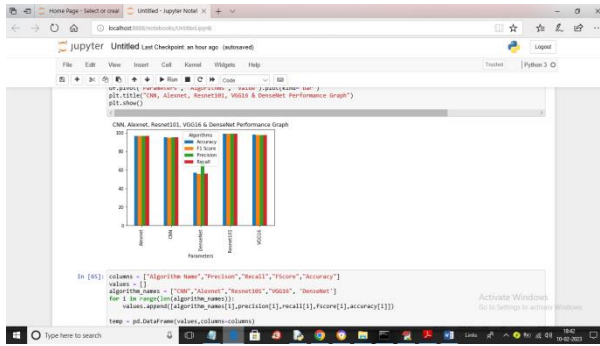
In above screen with DenseNet we got 57% accuracy



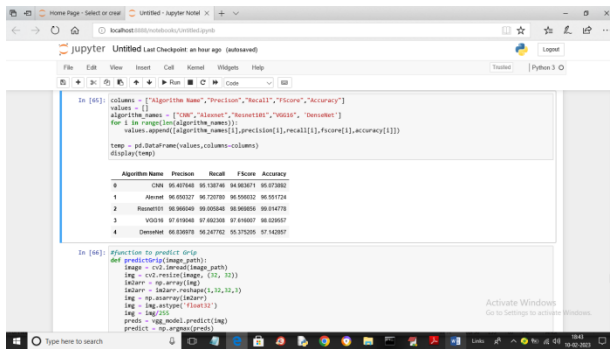
In above graph we are showing training accuracy for each algorithm and each different colour line represents accuracy of different algorithm where x-axis represents training epoch and y-axis represents accuracy and with each increasing epoch accuracy get increased



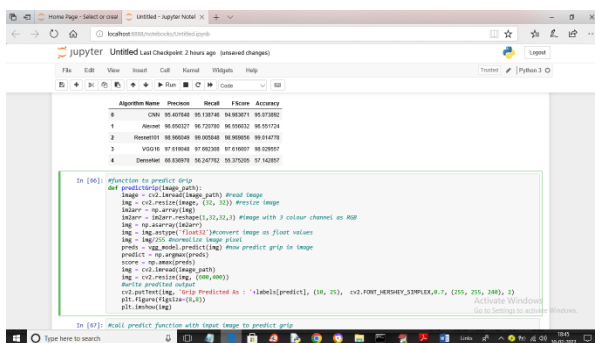
In above loss graph with each increasing epoch loss got decreased



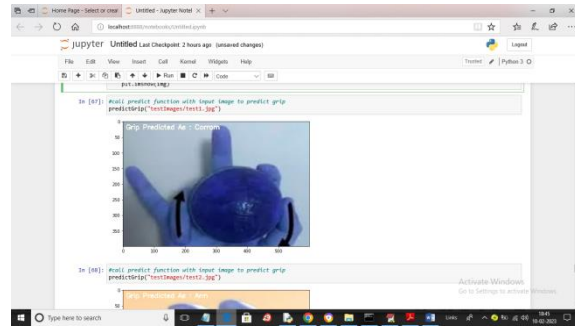
In above graph x-axis represents algorithm names and y-axis represents different metrics such as accuracy, precision, recall and FSCORE in different colour bars and in above graph we can see 4 algorithms has achieved accuracy of more than 95%



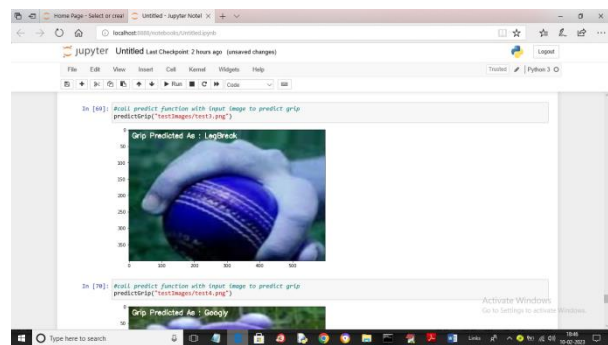
In above screen we can see each algorithm performance in tabular format



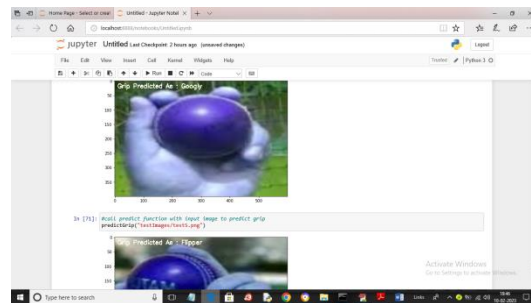
In above code we are reading input image and then predict type of grip like below output



In above screen grip predicted as 'Carrom' which you can see in white colour text



In above screen grip predicted as Leg Break



So in above screen we can see other predicted grip output

5. 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

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. Christe 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.
- [3] A. Khan, A. Sohail, U. Zahoor and A. S. Qureshi, "A Survey of the Recent Architectures of Deep Convolutional Neural Networks," *Artificial Intelligence Review*, pp. 1-62, 2020.
- [4] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional," in *Advances in neural information processing systems*, vol. 25, 2012, pp. 1097-1105.
- [5] T. Singh, V. Singla and P. Bhatia, "Score and winning prediction in cricket through data mining," in *International Conference on Soft Computing Techniques and Implementations (ICSCTI)*, 2015.
- [6] V. Subramaniaswamy, R. Logesh and V. Indragandhi, "Intelligent sports commentary recommendation system for individual cricket players," *International Journal of Advanced Intelligence Paradigms*, vol. 10, pp. 103-117, 2018.
- [7] A. Kaluarachchi and S. V. Aparna, "CricAI: A classification based tool to predict the outcome in ODI cricket.," in *International Conference on Information and Automation for Sustainability*, 2010.
- [8] F. K. Fuss, B. Doljin and R. E. Ferdinands, "Effect of the Grip Angle on Off-Spin Bowling Performance Parameters, Analysed with a Smart Cricket Ball," in *Multidisciplinary Digital Publishing Institute Proceedings*, 2018.
- [9] R. A. Minhas, A. Javed, A. Irtaza, M. T. Mahmood and Y. B. Joo, "Shot classification of field sports videos using AlexNet Convolutional Neural Network," *Applied Sciences*, vol. 9, no. 3, p. 483, 2019.
- [10] T. Kautz, B. H. Groh, J. Hannink, U. Jensen, H. Strubberg and B. M. Eskofier, "Activity recognition in beach volleyball using a Deep Convolutional Neural Network," *Data Mining and Knowledge Discovery*, vol. 31, no. 6, pp. 1678-1705, 2017.