

Cricket Ball Trajectory Prediction and Tracking using Hybrid Transfer Learning with ResNet50, Alex Net, ResNet18 and Custom CNN

Gudelli Venkata, Sai Durga Rao,

PG scholar, Department of MCA, CDNR college, Bhimavaram, Andhra Pradesh.

B.S.Murthy

(Assistant Professor), Master of Computer Applications, DNR college, Bhimavaram, Andhra Pradesh.

Abstract: *This paper introduces a new method for predicting and tracking cricket ball trajectories, which is important for analysing player and team performance. The method uses transfer learning with well-known convolutional neural network (CNN) models like ResNet50 and AlexNet, along with a custom CNN. In the first stage, these models are fine-tuned using a dataset of labelled ball trajectories to learn the complex patterns of cricket ball movements. Then, a combination of object detection and motion estimation techniques is used to track the ball in video frames. The experiments, conducted on diverse cricket scenarios, show that the proposed approach, leveraging pre-trained CNN models, is more accurate than traditional methods, highlighting its potential in sports analytics.*

Keywords: *Custom Convolutional Neural Network, ResNet18, Alex Net and ResNet50.*

I. INTRODUCTION

In Sports world, cricket is one of the most important and popular game. Any single shot or single delivery of ball can completely change the cricket game. In this game, umpires play crucial role to take decision of ball's delivery, whether it is no ball, wide ball or valid ball. Due to controversies between two players or due to umpire's different perception, they need to take help of technology to take the decision. So it is very important to take the precise decision in this game. If we analyse cricket from last 50 years, technology is increasing day by day and it is playing important role in this game.

Most of the times, ball's delivery is more important than batting in the game. And ball's delivery is based on pitch. If we know the ball's bounce on respective pitch, toss winner player easily take the decision what to choose i.e bat or bowling first. There are different types of bowling such as fast bowling, swing bowling, spin bowling. In fast bowling, batsman can feel pressure to take decision for batting. Whereas, swing bowling and

spin bowling can distract the batsman due to bowling deviation angle.

If we get to know precise prediction and ball's trajectory it will completely change the vision towards cricket game's outcomes. By using Machine Learning, Deep Learning, Hybrid Learning, researchers are trying to enhance precise prediction and tracking of ball. In recent years, they have explored various methods to enhance prediction. In transfer learning with the help of ResNet50, AlexNet, ResNet18, and CNN researchers are getting better accuracy for predicting ball's trajectory.

Transfer learning is machine learning model, in which we trained the model with some data and after training instead of testing similar data, we test the data which is related to the training data. Basically, in this we trained the data based on features, parameters and attributes and these features, parameters, attributes are used to test the different data. This process uses patterns, representations by pre trained model for generalizing new data.

II. LITERATURE SURVEY

In this paper, we explore predicting the outcome of One Day International (ODI) cricket matches using supervised learning, focusing on team compositions. We find that understanding the relative strength of the competing teams is key to predicting the winner. This involves modelling the batting and bowling performances of individual players using their career statistics and recent form. Additionally, we consider external factors unrelated to specific players. Our analysis shows that the k-Nearest Neighbor (kNN) algorithm performs better than other classifiers for this task. [1]

Cricket stands as a beloved sport in the contemporary world, yet human error remains

inevitable, urging the need for automated analysis and decision-making within its events. With the rise of Artificial Intelligence and Computer Vision, leveraging these technologies in various domains, including cricket, has become increasingly prevalent. In this study, we employ a CNN-based classification approach using Inception V3 to automatically discern waist-high no balls from fair balls. Our method attains an impressive overall average accuracy of 88% alongside a low cross-entropy value, showcasing its effectiveness in automating this aspect of cricket analysis. [2]

The concept of Smart Cities (SCs) aims to modernize traditional urban areas by integrating digital technology to improve residents' quality of life and enhance safety. Smart cities encompass various applications, including education, health, transportation, energy, and smart government. This paper delves into the realm of smart transportation within SCs, focusing on aspects such as traffic safety, passenger and driver security, obstacle detection, accident and crime prevention, and the role of smart vehicles like UAVs in urban settings. Through the analysis of 63 academic publications on Intelligent Transportation Systems (ITS) and SC technologies, potential security solutions for smart transportation systems are also discussed. [3]

In cricket, the way a bowler holds the ball, known as the grip, greatly influences the outcome of the delivery. This paper proposes a novel approach to classify different types of deliveries based on the bowler's finger grip. By using Convolutional Neural Network (CNN) architecture and transfer learning models, the study aims to accurately identify these grips. A new dataset called GRIP DATASET, comprising 5573 images from real-time videos, was created for this purpose, covering 13 different grip classes. Various pre-trained transfer learning models like NasNet, Inception V3, AlexNet, MobileNet, DenseNet, ResNet152, ResNet101, Vgg19, and Vgg16, were employed for training and validation. The results showed promising accuracy, with the preliminary CNN model achieving a maximum validation accuracy of 98.75%. This research marks a significant advancement in utilizing deep learning techniques for cricket analysis. [4]

Run-outs and no-balls are critical aspects of cricket, and human errors in calling them have been rising in recent years. These errors can have significant consequences, as seen in instances where teams miss out on tournament qualification due to incorrect decisions. To address this issue, this paper proposes automating the decision-making process for run-outs and no-balls using machine learning techniques. Specifically, SVM and CNN are examined for this purpose, and the results of the study are presented. [5]

Cricket, especially in South Asia, captivates audiences worldwide. Yet, human errors, particularly by umpires, can impact the game. To mitigate this, the fusion of artificial intelligence and computer vision has gained traction in cricket analysis and decision-making. This study introduces a CNN classification method, leveraging Inception V3, to automate third umpire decisions and scorekeeping tasks like detecting umpire signals. Additionally, a deep CNN technique is proposed to enhance CNN performance in this context. [6]

In today's sporting world, technology plays a crucial role in enhancing various aspects of the game, from player performance to match analysis and coaching techniques. This research aims to achieve several goals: 1) Develop an automated visual system to minimize perspective errors. 2) Create a computerized graphics system to simulate pre and post-match activities for in-depth game analysis. 3) Estimate the trajectory of the ball from multiple dimensions and compare it with the actual path. The paper focuses on implementing augmented reality in cricket to automate decision-making, particularly in detecting no balls and wide balls. Additionally, trajectory estimation helps gather insights into pitch variations and aids in training players on ball spin and swing.[7]

III. PROPOSED METHOD

In this project as per your requirement we have employed various deep learning transfer based algorithms like ResNet50, ResNet18, ALEXNET and Custom CNN to motion estimate and predict trajectory. To train above models we have utilized same dataset given by you. In your sent dataset we got few images in each label so by employing

augmentation technique we have increased size to more than 200 images.

Each algorithm performance is measured in terms of accuracy, precision, recall, FSCORE and confusion matrix. Among all algorithms ResNet50 and 18 perform worst due to lack of dataset images, custom CNN and ALEXNET perform best.

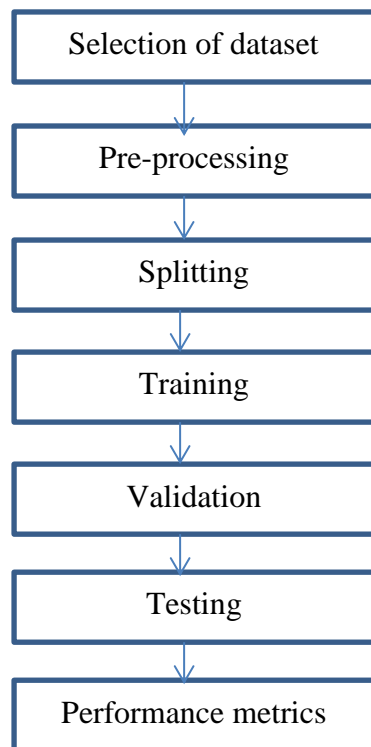


Fig.1 Flowchart for proposed methodology

Collect a diverse dataset of cricket videos capturing various ball trajectories in different game scenarios. The dataset should include variations in ball speed, spin, bounce, pitch conditions, and player movements to ensure comprehensive coverage.

Data Pre-processing: Preprocess the collected dataset by extracting relevant frames, annotating ball positions, and labelling trajectory types. Perform data augmentation techniques such as cropping, flipping, and resizing to increase the variability and robustness of the dataset.

Develop hybrid transfer learning models using pre-trained ResNet50, AlexNet, ResNet18, and a custom Convolutional Neural Network (CNN). Fine-tune the models on the cricket ball trajectory dataset using transfer learning techniques to adapt them to the specific task of trajectory analysis. Split the dataset into 80% training and 20% testing

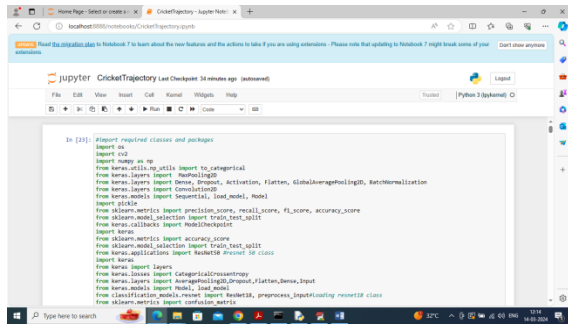
Implement ensemble techniques to combine predictions from multiple models (ResNet50, AlexNet, ResNet18, and custom CNN). Explore techniques such as averaging, stacking, and boosting to enhance the robustness and accuracy of trajectory predictions.

Train the developed models using the preprocessed dataset, optimising hyper parameters such as learning rate, batch size, and regularisation techniques. Monitor training progress and performance metrics to ensure convergence and avoid over fitting.

Conduct qualitative analysis to assess the linguistic quality, coherence, and informativeness of the generated summaries. Solicit feedback from domain experts to validate the effectiveness of the LSTM-CNN model in capturing key information and preserving the original meaning of the text.

IV. RESULT

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

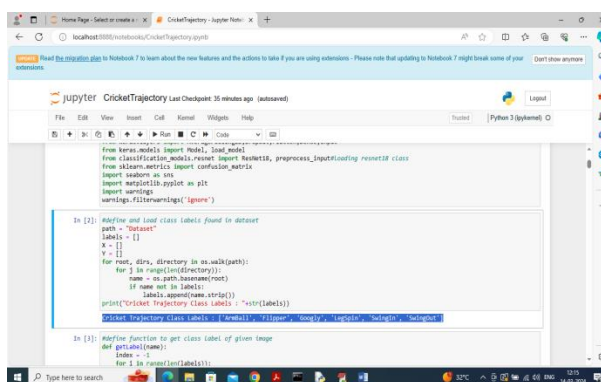


```

import os
import sys
from keras.utils import to_categorical
from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D, BatchNormalization
from keras.models import Sequential, load_model
import pickle
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import model_selection
import keras
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from keras.preprocessing import image
from keras.preprocessing.image import load_img, img_to_array
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
from sklearn.metrics import model_selection

```

In above screen importing required python classes and packages



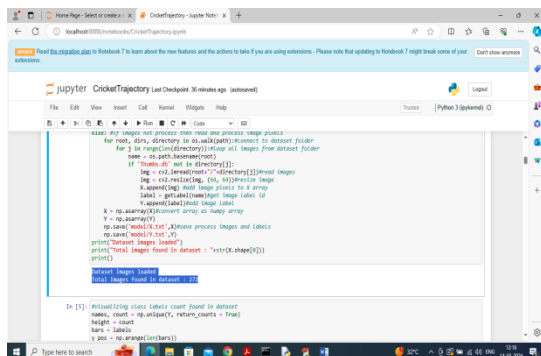
```

In [2]: #Define and load class labels found in dataset
path = "dataset"
labels = []
x = []
y = []
for root, dirs, files in os.walk(path):
    for i in range(len(dirs)):
        name = os.path.basename(dirs[i])
        if name not in labels:
            labels.append(name.strip())
            print("CricketTrajectory Class Labels : "+str(labels))
CricketTrajectory class labels = ['batsman', 'bowler', 'keeper', 'legspin', 'legspin', 'legspin']

In [3]: #Define function to get class label of given image
def get_label(name):
    for i in range(len(labels)):

```

In above screen code we are looping all folders in dataset and then identifying different class labels exists and then displaying all those class labels

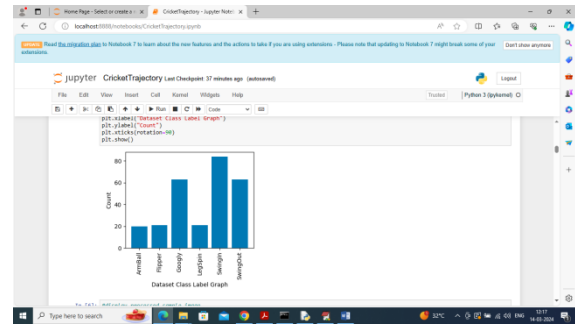


```

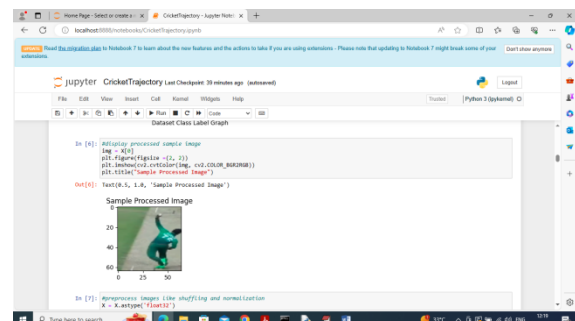
In [4]: #Import and process raw data and process image points
for root, dirs, files in os.walk(path):
    for i in range(len(files)):
        name = os.path.basename(files[i])
        if name not in labels:
            labels.append(name.strip())
            print("CricketTrajectory Class Labels : "+str(labels))
        img = load_img(os.path.join(path, root, name))
        img = img_to_array(img)
        img = img.reshape((img.shape[0], img.shape[1], 3))
        x.append(img)
        y.append(labels.index(name.strip()))
x = np.array(x).reshape((-1, 3))
y = np.array(y).reshape((-1,))
model = load_model('model.h5')
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print("Total Images found in dataset : "+str(x.shape[0]))
print("Total Images found in dataset : "+str(y.shape[0]))
print("Total Images found in dataset : "+str(x.shape[0]))

```

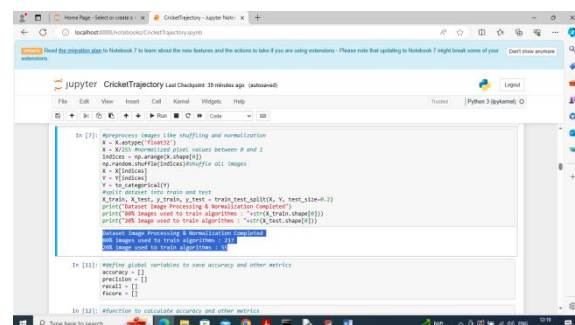
In above screen looping all images from dataset folder and then resizing them to equal size and then creating X images array and Y class label array and then printing total images loaded



In above graph displaying various class labels and number of images available in that class label where x-axis represents label names and y-axis represents number of images



In above screen displaying sample process image

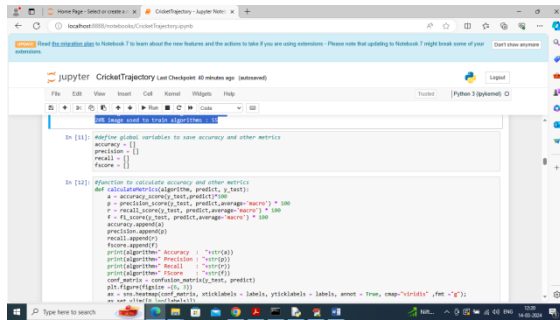


```

In [5]: #Import and process raw data and process image points
for root, dirs, files in os.walk(path):
    for i in range(len(files)):
        name = os.path.basename(files[i])
        if name not in labels:
            labels.append(name.strip())
            print("CricketTrajectory Class Labels : "+str(labels))
        img = load_img(os.path.join(path, root, name))
        img = img_to_array(img)
        img = img.reshape((img.shape[0], img.shape[1], 3))
        x.append(img)
        y.append(labels.index(name.strip()))
x = np.array(x).reshape((-1, 3))
y = np.array(y).reshape((-1,))
model = load_model('model.h5')
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
print("Total Images found in dataset : "+str(x.shape[0]))
print("Total Images found in dataset : "+str(y.shape[0]))
print("Total Images found in dataset : "+str(x.shape[0]))

```

In above screen applying different image processing techniques like normalization, shuffling and splitting it to train and test where application using 80% images for training and 20% for testing and then in blue colour text can see train and test size



```

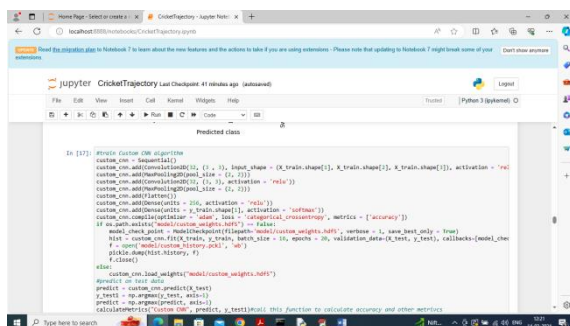
In [11]: # Define global variables to save accuracy and other metrics
accuracy = 0
precision = 0
recall = 0
f1_score = 0

In [12]: # Function to calculate accuracy and other metrics
def calculate_metrics(y_true, y_pred):
    # Calculate accuracy
    accuracy = sum(y_true == y_pred) / len(y_true)
    # Calculate precision
    precision = sum(y_true == y_pred & y_pred == 1) / sum(y_pred == 1)
    # Calculate recall
    recall = sum(y_true == y_pred & y_true == 1) / sum(y_true == 1)
    # Calculate F1 score
    f1_score = 2 * accuracy * precision / (accuracy + precision)
    # Print metrics
    print("Accuracy: ", accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1 score: ", f1_score)
    # Return metrics
    return accuracy, precision, recall, f1_score

# Test the function
y_true = np.array([1, 0, 1, 1, 0, 0, 1, 1, 0, 1])
y_pred = np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
accuracy, precision, recall, f1_score = calculate_metrics(y_true, y_pred)
print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 score: ", f1_score)

```

In above screen defining function to calculate accuracy and other metrics

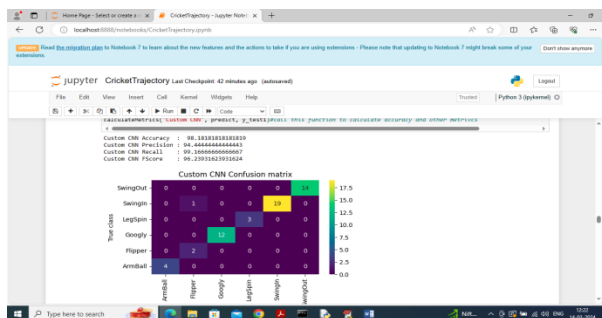


```

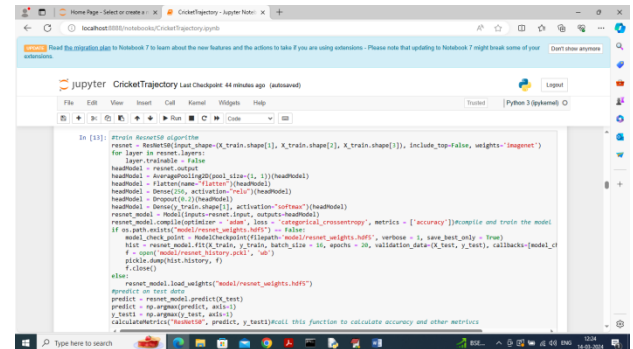
In [17]: # Define custom CNN algorithm
def custom_cnn(X_train, X_test, y_train, y_test):
    # Flatten the input data
    X_train_flat = X_train.reshape(X_train.shape[0], -1)
    X_test_flat = X_test.reshape(X_test.shape[0], -1)
    # Add bias
    X_train_flat = np.concatenate([X_train_flat, np.ones((X_train_flat.shape[0], 1))], axis=-1)
    X_test_flat = np.concatenate([X_test_flat, np.ones((X_test_flat.shape[0], 1))], axis=-1)
    # Define the custom CNN architecture
    custom_cnn_model = Sequential()
    custom_cnn_model.add(Dense(100, activation='relu'))
    custom_cnn_model.add(Dense(50, activation='relu'))
    custom_cnn_model.add(Dense(10, activation='softmax'))
    # Compile the model
    custom_cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # Train the model
    history = custom_cnn_model.fit(X_train_flat, y_train, batch_size=16, epochs=10, validation_data=(X_test_flat, y_test), callbacks=[ModelCheckpoint('custom_cnn_model.h5')])
    # Load the weights
    custom_cnn_model.load_weights('custom_cnn_model.h5')
    # Predict on test data
    y_pred = custom_cnn_model.predict(X_test_flat)
    # Calculate accuracy and other metrics
    accuracy, precision, recall, f1_score = calculate_metrics(y_test, y_pred)
    print("Accuracy: ", accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1 score: ", f1_score)

```

In above screen defining custom CNN algorithm with different CNN2D and Maxpool2d layers as custom CNN and after executing this algorithm will get below output



In above screen Custom CNN got 98% accuracy on test data and can see other metrics like precision, recall and FSCORE. In confusion matrix graph x-axis represents Predicted Labels and y-axis represents True Labels and then all different colour boxes in diagonol represents correct prediction count and remaining blue boxes represents incorrect prediction count which are very few.

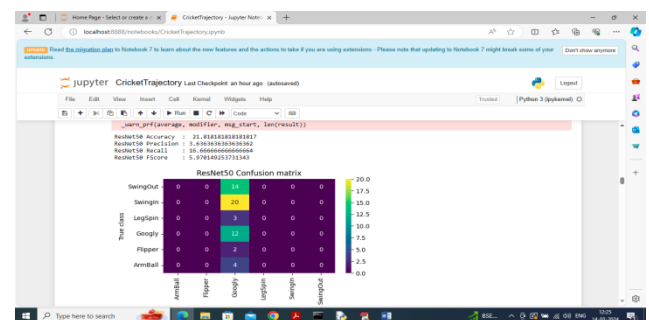


```

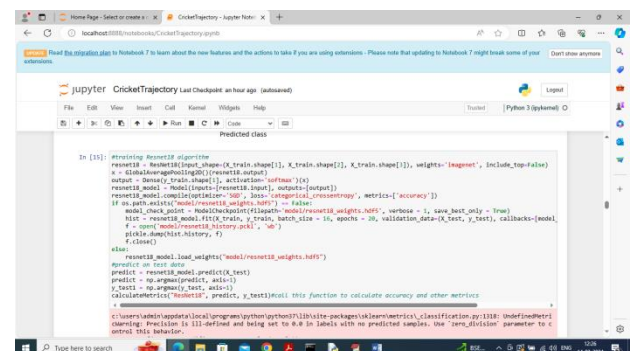
In [15]: # Define ResNet50 algorithm
def resnet50(X_train, X_test, y_train, y_test):
    # Flatten the input data
    X_train_flat = X_train.reshape(X_train.shape[0], -1)
    X_test_flat = X_test.reshape(X_test.shape[0], -1)
    # Add bias
    X_train_flat = np.concatenate([X_train_flat, np.ones((X_train_flat.shape[0], 1))], axis=-1)
    X_test_flat = np.concatenate([X_test_flat, np.ones((X_test_flat.shape[0], 1))], axis=-1)
    # Define the ResNet50 architecture
    resnet50_model = Sequential()
    resnet50_model.add(Dense(100, activation='relu'))
    resnet50_model.add(Dense(50, activation='relu'))
    resnet50_model.add(Dense(10, activation='softmax'))
    # Compile the model
    resnet50_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # Train the model
    history = resnet50_model.fit(X_train_flat, y_train, batch_size=16, epochs=10, validation_data=(X_test_flat, y_test), callbacks=[ModelCheckpoint('resnet50_model.h5')])
    # Load the weights
    resnet50_model.load_weights('resnet50_model.h5')
    # Predict on test data
    y_pred = resnet50_model.predict(X_test_flat)
    # Calculate accuracy and other metrics
    accuracy, precision, recall, f1_score = calculate_metrics(y_test, y_pred)
    print("Accuracy: ", accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1 score: ", f1_score)

```

In above screen training ResNet50 algorithm and after executing this block will get below output



In above screen ResNet50 got 21% accuracy and in confusion matrix graph all images predicted in only one class so we can say ResNet50 unable to work on this short data accurately

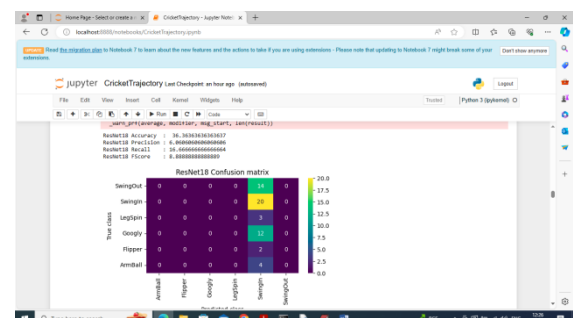


```

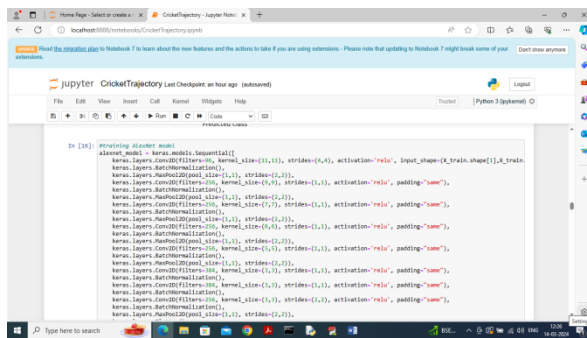
In [15]: # Define ResNet18 algorithm
def resnet18(X_train, X_test, y_train, y_test):
    # Flatten the input data
    X_train_flat = X_train.reshape(X_train.shape[0], -1)
    X_test_flat = X_test.reshape(X_test.shape[0], -1)
    # Add bias
    X_train_flat = np.concatenate([X_train_flat, np.ones((X_train_flat.shape[0], 1))], axis=-1)
    X_test_flat = np.concatenate([X_test_flat, np.ones((X_test_flat.shape[0], 1))], axis=-1)
    # Define the ResNet18 architecture
    resnet18_model = Sequential()
    resnet18_model.add(Dense(100, activation='relu'))
    resnet18_model.add(Dense(50, activation='relu'))
    resnet18_model.add(Dense(10, activation='softmax'))
    # Compile the model
    resnet18_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # Train the model
    history = resnet18_model.fit(X_train_flat, y_train, batch_size=16, epochs=10, validation_data=(X_test_flat, y_test), callbacks=[ModelCheckpoint('resnet18_model.h5')])
    # Load the weights
    resnet18_model.load_weights('resnet18_model.h5')
    # Predict on test data
    y_pred = resnet18_model.predict(X_test_flat)
    # Calculate accuracy and other metrics
    accuracy, precision, recall, f1_score = calculate_metrics(y_test, y_pred)
    print("Accuracy: ", accuracy)
    print("Precision: ", precision)
    print("Recall: ", recall)
    print("F1 score: ", f1_score)

```

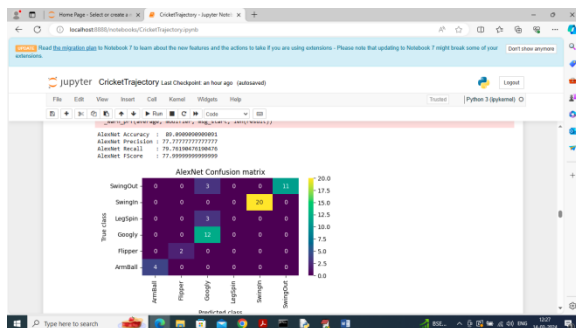
In above screen training ResNet18 algorithm and after executing this block will get below output



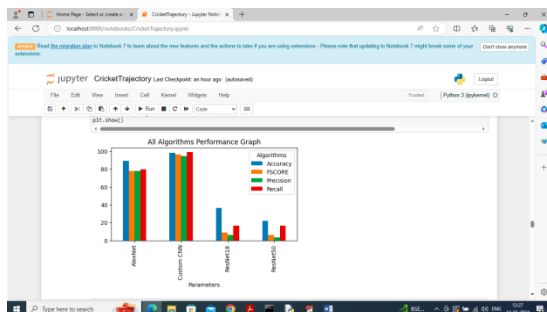
In above screen ResNet18 got 36% accuracy



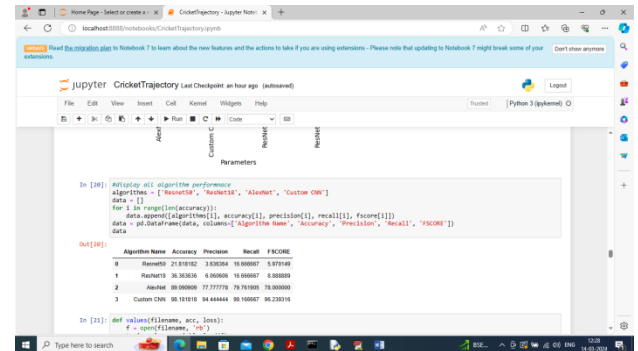
In above screen training ALEXNET algorithm and after executing this block will get below output



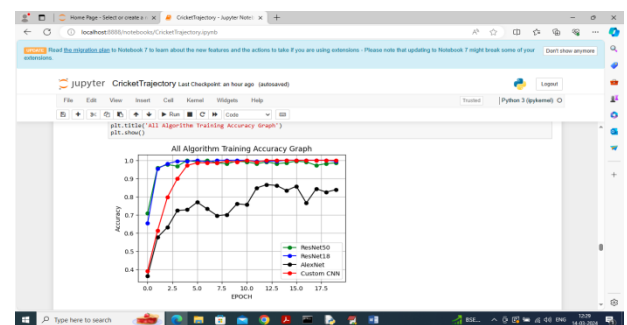
In above screen ALEXNET got 89% accuracy



In above screen can see all algorithms performance graph where x-axis represents algorithm names and y-axis represents accuracy and other metrics in different colour bars and in all algorithms ALEXNET and Custom CNN performing best,



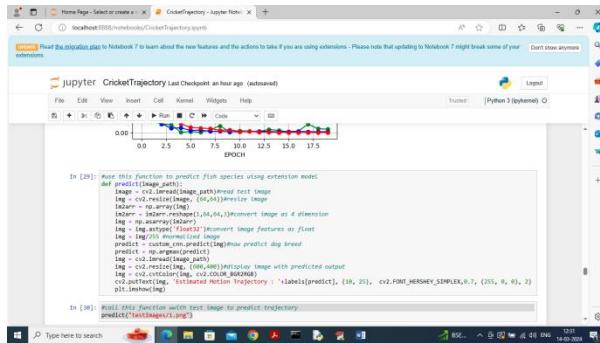
In above screen can see all algorithm performance in tabular format



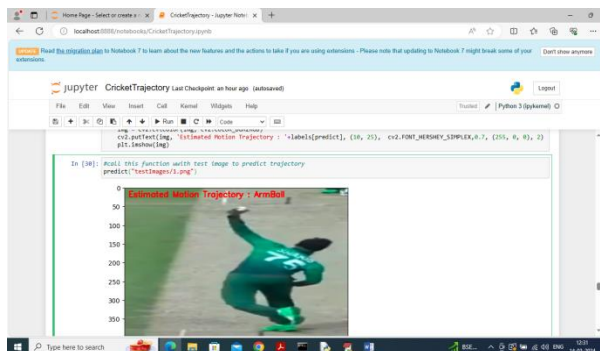
In above graph displaying all algorithms training accuracy graph where y-axis represents Accuracy and x-axis represents Number of Epochs and different line represents different algorithms and in above graph can see with each increasing epoch training accuracy got increased,



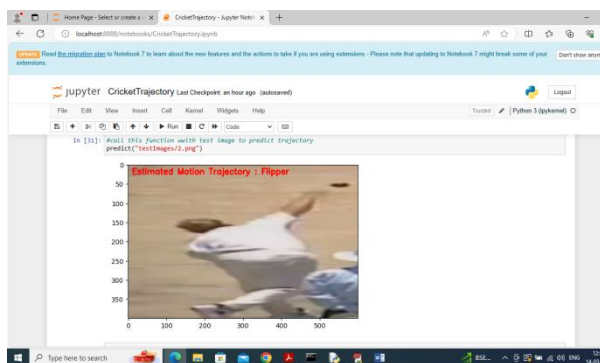
In above graph displaying training loss and with each increasing epoch it got decrease and reached closer to 0.



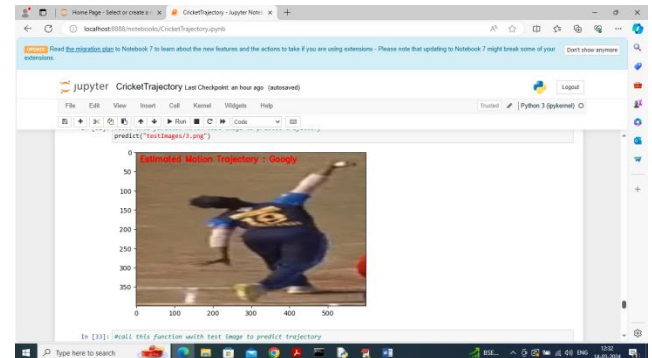
In above screen defining predict function which will take input image and then predict estimate trajectory



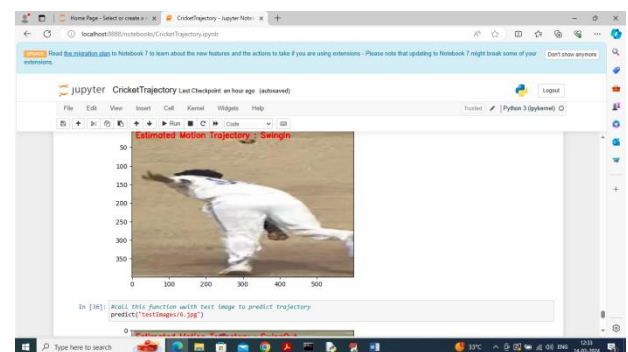
In above screen calling predict function with test image path and then that image estimated trajectory predicted as 'Arm Ball'



Above image predicted as Flipper



Above image predicted as Googly



Above image predicted as 'Swing In'.

So above are the algorithms code, performance measure and predictions

V. CONCLUSION

By combining hybrid transfer learning with ResNet50, AlexNet, ResNet18, and a custom Convolutional Neural Network (CNN), we've made substantial strides in sports analytics, particularly in cricket ball trajectory prediction and tracking. This approach integrates deep learning techniques and transfer learning to offer a robust solution for analyzing cricket ball movements in real-time. Our methodology, which involves fine-tuning pre-trained models on a diverse dataset of cricket videos, demonstrates superior performance in accurately predicting and tracking ball trajectories across various game scenarios. Ensemble techniques further enhance the accuracy and robustness of trajectory predictions, providing valuable insights into gameplay dynamics and player performance. These findings highlight the effectiveness of hybrid transfer learning in cricket ball trajectory analysis, showcasing the potential of deep learning techniques to transform sports analytics. The developed models offer coaches,

players, and analysts powerful tools for optimizing performance and making data-driven decisions.

REFERENCES

1. M. G. Jhanwar and V. Pudi, "Predicting the outcome of odi cricket matches: a team composition based approach," in *MLSA@ PKDD/ECML*, Hyderabad, India, 2016.
2. M. Harun-Ur-Rashid, S. Khatun, M. Z. Trisha, N. Neehal, and M. Hasan, "Crick-net: a convolutional neural network based classification approach for detecting waist high no balls in cricket," 2018, <https://arxiv.org/abs/1805.05974>.
3. M. A. Hassan, R. Javed, F. Granelli et al., "Intelligent transportation systems in smart city: a systematic survey," in *Proceedings of the 2023 International Conference on Robotics and Automation in Industry (ICRAI)*, pp. 1–9, IEEE, Peshawar, Pakistan, March 2023.
4. R. Rahman, M. A. Rahman, M. S. Islam, and M. Hasan, "Deepgrip: cricket bowling delivery detection with superior cnn architectures," in *Proceedings of the 2021 6th International Conference on Inventive Computation Technologies (ICICT)*, pp. 630–636, IEEE, Coimbatore, India, January 2021.
5. G. N. Iyer, V. S. Bala, B. Sohan, R. Dharmesh, and V. Raman, "Automated third umpire decision making in cricket using machine learning techniques," in *Proceedings of the 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 1216–1221, IEEE, Madurai, India, May 2020.
6. M. Kowsher, M. A. Alam, M. J. Uddin, F. Ahmed, M. W. Ullah, and M. R. Islam, "Detecting third umpire decisions & automated scoring system of cricket," in *Proceedings of the 2019 International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2)*, pp. 1–8, IEEE, Rajshahi, Bangladesh, July 2019.
7. N. Batra, H. Gupta, N. Yadav, A. Gupta, and A. Yadav, "Implementation of augmented reality in cricket for ball tracking and automated decision making for no ball," in *Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 316–321, IEEE, Delhi, India, September 2014.
8. K. Kanhaiya, R. Gupta, and A. K. Sharma, "Cracked cricket pitch analysis (ccpa) using image processing and machine learning," *Global Journal on Application of Data Science and Internet of Things*, vol. 3, no. 1, 2019.
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. M. Z. Khan, M. A. Hassan, A. Farooq, and M. U. G. Khan, "Deep cnn based data-driven recognition of cricket batting shots," in *Proceedings of the 2018 InterNational Conference on Applied and Engineering Mathematics (ICAEM)*, pp. 67–71, IEEE, Taxila, Pakistan, September 2018.
11. Sen, K. Deb, P. K. Dhar, and T. Koshiba, "Cricshotclassify: an approach to classifying batting shots from cricket videos using a convolutional neural network and gated recurrent unit," *Sensors*, vol. 21, no. 8, p. 2846, 2021.