ABSTRACT

This mini-project is focused on implementing the research paper that explores the application of Convolutional Neural Networks (CNNs) in categorizing cricket batting shots, leveraging transfer learning techniques. By utilizing the Shot-Net dataset consisting of 4586 images, it aimed to classify four distinct batting strokes. Through rigorous experimentation, we identified the InceptionV3 network as the most effective model architecture, achieving an impressive 98.84% accuracy on the validation dataset. Our study highlights the significant potential of deep learning methodologies in sports analytics, particularly in understanding complex player actions. By employing transfer learning on pre-trained models like InceptionV3, we can leverage existing knowledge to enhance classification accuracy. This research contributes to advancing the field of sports analytics by harnessing the power of deep learning algorithms for nuanced tasks like cricket shot classification.

TABLE OF CONTENTS

INTRODUCTION

Cricket, often referred to as a gentleman's game, is a popular sport played by millions worldwide. Originating in England in the 16th century, cricket has evolved into one of the most widely followed and passionately played sports across the globe. It is characterized by its unique blend of strategy, skill, and endurance, played between two teams of eleven players each. With the advent of advanced technology and data analytics, cricket has witnessed a paradigm shift in how the game is analyzed and understood. One significant development in this realm is the integration of Convolutional Neural Networks (CNNs) into cricket analytics. CNNs, a class of deep learning algorithms inspired by the human visual cortex, have revolutionized computer vision tasks and found applications in various domains, including sports. The integration of CNNs in cricket has not only enhanced the way the game is analyzed but has also opened up new avenues for player development, strategy formulation, and fan engagement. As technology continues to evolve, CNNs are expected to play an increasingly vital role in shaping the future of cricket, providing innovative solutions to challenges and unlocking new opportunities for growth and innovation in the sport.

REQUIREMENT SPECIFICATION

Dataset:

In this implementation, we utilized the Cricket-shot dataset sourced from Kaggle, comprising 4586 images categorized into four distinct cricket batting shots. Additionally, the dataset includes augmented images, incorporating transformations such as **flipping, rotation, scaling, shearing, and translation**, enhancing its diversity and robustness.

GPU Usage:

To expedite the training process of our Convolutional Neural Network (CNN), we leveraged additional GPU support. Specifically, we utilized the GPU P100 type, provided by Kaggle Notebook, to harness its computational power and accelerate the model training, thereby optimizing the efficiency of our implementation.

METHODOLOGY for IMPLEMENTATION

InceptionV3

Inception V3 is a deep convolutional neural network (CNN) architecture developed by Google. It is renowned for its effectiveness in image classification tasks, particularly in the realm of computer vision. Introduced as part of the Inception family of models, Inception V3 incorporates several innovative features, including inception modules that utilize various filter sizes within the same layer to capture diverse patterns and features at multiple scales. This architecture also employs batch normalization and factorized convolutions to enhance training stability and computational efficiency, respectively. With its deep architecture and advanced design principles, Inception V3 has been widely adopted and serves as a benchmark model in the field of deep learning for image recognition tasks.

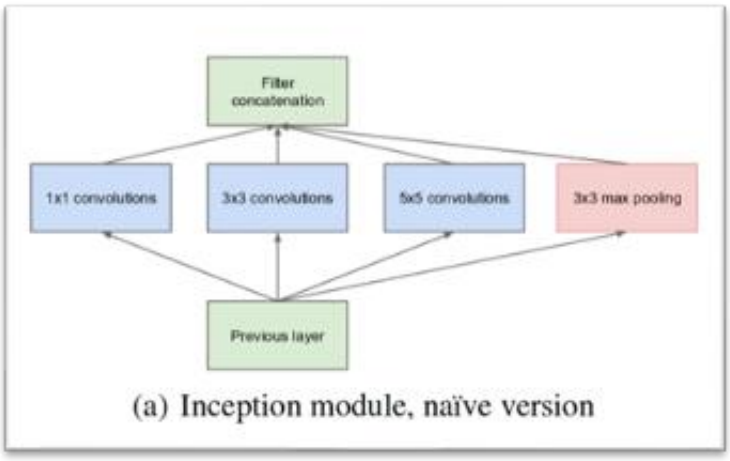


Fig 1: Inception network architecture

Training the model

Entire dataset was split into three parts (train_set, validation_set, test_set) with the split ratio of 70:15:15. Image resizing was performed to provide uniformity between the images. All the images were resized to **256x256x3** pixels. One Hot Encoding was performed on target variables to convert string values into numerical datatype. To adapt the model to our task, six additional dense layers were appended, with node counts ranging from 1024 to 32, followed by an output layer with 4 nodes representing our 4 output classes. **ReLU** activation functions were

employed for the dense layers, while the output layer utilized a **softmax** activation function. To prevent overfitting, three dropout layers with dropout probabilities of 0.2, 0.15, and 0.1 respectively were inserted between the dense layers.

After experimenting with various optimizers, we settled on the **Adam optimizer** and categorical cross-entropy loss function to minimize the loss. During training, a batch size of 32 was utilized, meaning that each iteration involved randomly selecting 32 images for model training. The model underwent a total of **10 epochs** to complete the training process.

RESULTS

Once the model had been trained successfully on the training_dataset, it is subjected to make predictions using test_dataset. For assessing the model prediction, we have applied a variety of evaluation methods:

Plotted **accuracy chart** of a deep learning model that gives a visual depiction of the model's performance in terms of accuracy on a particular task or dataset. From the accuracy chart provided (Fig 1.2), it's clear that the validation accuracy is rising in tandem with the training accuracy. This trend suggests that the model is generalizing well to unseen data, mitigating the risk of overfitting.

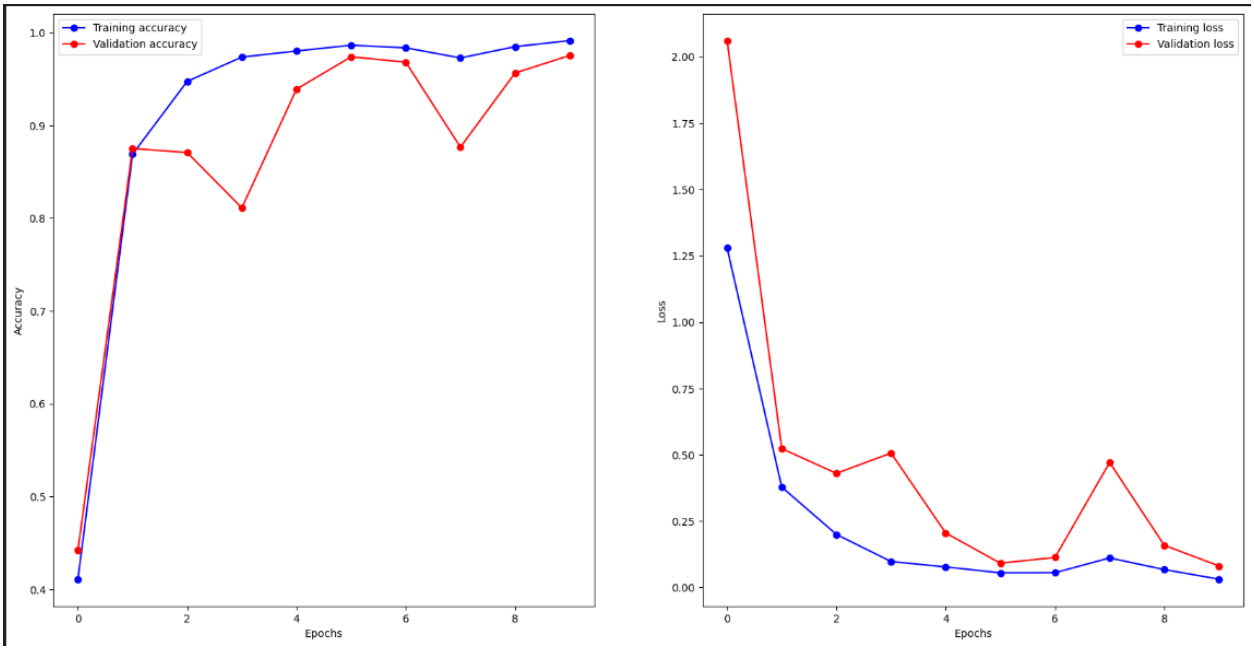


Fig 2: Accuracy chart

Confusion matrix was used to summarizes the performance of a deep learning model on a classification assignment. They reveal the types and frequencies of misclassifications made by the model, enabling identification of patterns and areas for improvement. This insight can guide further model refinement and feature engineering efforts.

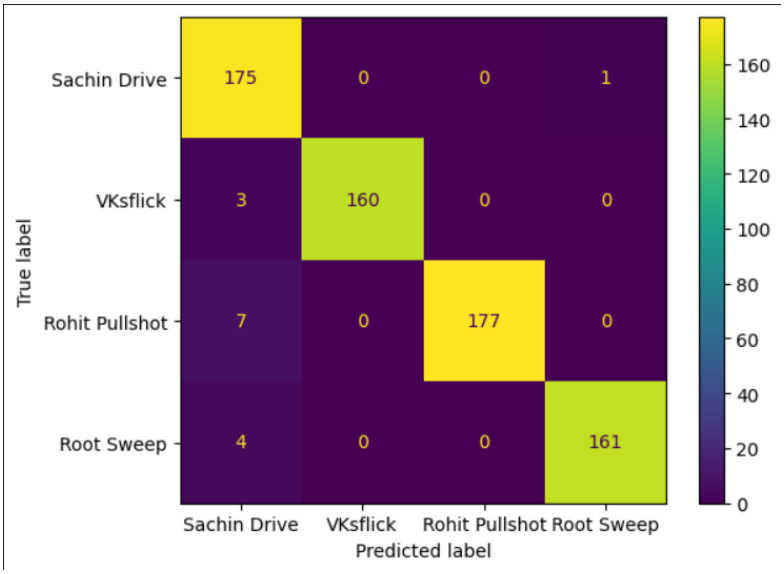


Fig 3: Confusion matrix

Model's **accuracy, precision, recall, and F1-score**, all of which are measures that are typically utilized to evaluate the performance of a classification model.

Accuracy: Accuracy is the proportion of correctly classified instances out of the total instances. It's a measure of overall correctness and is calculated as the ratio of correct predictions to the total number of predictions.

Precision: Precision measures the accuracy of positive predictions made by the model. It's the ratio of true positives (correctly predicted positive instances) to the sum of true positives and false positives (incorrectly predicted positive instances).

Recall: Recall, also known as sensitivity or true positive rate, measures the model's ability to correctly identify positive instances from all actual positive instances. It's the ratio of true positives to the sum of true positives and false negatives (positive instances incorrectly classified as negative).

F1-score: The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It's useful when class imbalance exists in the dataset. F1-score reaches its best value at 1 (perfect precision and recall) and worst at 0.

	Drive	Flick	Pull	Sweep
Precision	0.994318	0.993865	1.000000	0.993976
Recall	0.994318	0.993865	0.994565	1.000000
F1SCore	0.994318	0.993865	0.997275	0.996979
Total samples	176.000000	163.000000	184.000000	165.000000

Table 1: Precision, Recall, F1 Score

CODE

#importing the required libraries

```
import os
import cv2
import random
import matplotlib.pyplot as plt
from glob import glob
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout,
from tensorflow.keras.models import Model
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.losses import CategoricalCrossentropy
from tensorflow.math import confusion_matrix
```

#Loading the dataset

```
data_dir = '/kaggle/input/cricket-shot-dataset/data'
image_paths = []
class_labels = []
for class_name in labels:
    paths = glob(os.path.join(data_dir, class_name, "*.png"))
    image_paths.extend(paths)
    class_labels.extend([class_name] * len(paths))
df = pd.DataFrame(data={"image_path": image_paths, "label": class_labels})
labels = ['pullshot', 'drive', 'sweep', 'flick']
sizes = [1228, 1173, 1100, 1085]
fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%')
ax1.axis('equal')
ax1.set_title("Distribution of images")
plt.show()
```

#Splitting the dataset

```
from sklearn.model_selection import train_test_split
X = df
y = df[['label']]
df_train, df_temp, y_train, y_temp = train_test_split(X, y, stratify=y, test_size=0.30, random_state=40)
df_val, df_test, y_val, y_test = train_test_split(df_temp, y_temp, stratify=y_temp, test_size=0.5, random_state=40)
```

#Using ImageDataGenerator class for various pre-processing tasks

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

LR = 0.0003

train_generator =
ImageDataGenerator().flow_from_dataframe(dataframe=df_train,directory='./',x_col="image_pa
th", y_col="label", target_size=(256, 256), batch_size=b_size, shuffle=True,
class_mode='categorical', seed=42)

valid_generator =
ImageDataGenerator().flow_from_dataframe(dataframe=df_val,directory='./',x_col="image_path
", y_col="label", target_size=(256, 256), batch_size=b_size, shuffle=True,
class_mode='categorical', seed=42)

test_generator =
ImageDataGenerator().flow_from_dataframe(dataframe=df_test,directory='./',x_col="image_pat
h",y_col="label", target_size=(256, 256), batch_size=1, class_mode='categorical', shuffle=False)
```

#Defining the model

```
from tensorflow.keras.layers import Flatten, Dense, Dropout

from tensorflow.keras.models import Model

i_model = InceptionV3(input_shape=img_shape, include_top=False, weights='imagenet')
```

 # Add some layers to InceptionV3 model

```
x = Flatten()(i_model.output)

x = Dense(1024, activation='relu')(x)

x = Dropout(0.2)(x)

x = Dense(512, activation='relu')(x)

x = Dense(256, activation='relu')(x)

x = Dropout(0.15)(x)
```

 # Add the final output layer with softmax activation

```
i_output = Dense(num_classes, activation='softmax')(x)
```

 # Create the final model with InceptionV3 as the base and the added layers

```
i_final = Model(i_model.input, i_output)

LR = 0.0003
```

```
i_final.compile(optimizer=Adam(learning_rate=LR), loss=CategoricalCrossentropy(), metrics=['accuracy'])
```

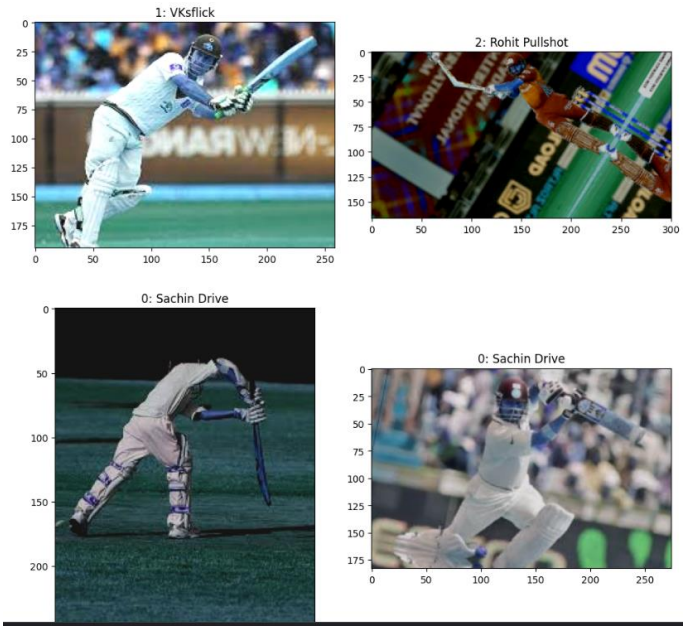
#Training the model

```
hist_incep=i_final.fit(train_generator, epochs=10, validation_data=valid_generator, callbacks=[early_stopping, reduce_lr])
```

#Testing the model

```
pred = i_final.predict(test_generator)

fig,ax=plt.subplots(2,2,figsize=[10,10])

n=len(df_test)

for ind in range(2):
    for ind1 in range(2):
        idx_im=random.randint(0,n)
        img=cv2.imread(df_test.iloc[idx_im]['image_path'])
        ax[ind][ind1].imshow(img)
        id_pred=pred[idx_im].argmax()
        ax[ind][ind1].set_title(f"{id_pred}: {name[id_pred]}")

plt.tight_layout()
```

Model predictions on random input images



CONCLUSION

We have successfully built a CNN model using Inception V3 network for classifying the cricket batting shots based on given input images. This model has classified the given images with significant accuracy enabling to be used in various applications. This model can help coaches and batsmen to improve their performance by giving them insights on the shots and suggestions for playing the perfect cricket shot. As part of future scope, we intend to apply this pose estimation technique to detect cricket players prone to injuries from the training videos. Furthermore, we can use other various techniques like transfer learning with the existing larger model (MobileNet, R-CNN) architectures along with hyper-parameter tuning. With the available Inception image classification model, we can develop a frontend for end-user interaction.