CHAPTER 1

INTRODUCTION

1.1 Background

As part of our college internship program, we Rucha Pathak, Sourav Shinde, and Anuj Pensalwar had the opportunity to work as Web Developer Interns at Avenirya Solutions Pvt. Ltd. The internship began in March 2024 and lasted for six months, concluding in july 2024. The working model was hybrid, combining both offline and online interactions, and our regular working schedule was from Monday to Friday. Throughout this period, we were mentored by Mr. Yash Kolnure, who brought invaluable insights from his experience in software systems and project development.

Avenirya Solutions Pvt. Ltd. Solutions is a technology-focused company dedicated to enhancing education and business operations through innovative digital platforms. The company specializes in building custom mobile applications and web applications, AI-powered tools, and intelligent automation solutions tailored for academic institutions and enterprises. Their mission is to empower educators and learners by simplifying complex digital workflows and enabling effective e-learning environments.

During our internship, we worked collectively on the design and development of a comprehensive Cricket Stroke Detection using CNN and Deep learning, a popular and open-source platform based on machine learning. This internship aimed to develop a system that can detect and classify cricket shots using a trained CNN model. The model is trained using labeled image data of different shot types, such as Pull, Sweep, Drive, and Leg Glance/Flick. The end product not only identifies the type of shot from a static image but also features a user-friendly GUI to interact with the system seamlessly.

At the end of this internship, we were proud to have received a Certificate of Completion from Avenirya Solutions Pvt. Ltd., acknowledging our contribution and the successful completion of the project.

1.2 Project Overview

Cricket is one of the most popular sports in the world, especially in countries like India. In this sport, batting strokes such as the cover drive, pull shot, straight drive, and sweep play a very important role in a player's performance. Recognizing and analyzing these strokes can help players improve their game and coaches to train them better. But manually identifying these strokes from images is a time-consuming and tiring process. To solve this problem, we have developed a project called "Cricket Stroke Detection Using CNN", where CNN stands for Convolutional Neural Network.

The main idea behind this project is to build a smart system that can automatically detect different cricket strokes from images using deep learning. Deep learning is a part of Artificial Intelligence (AI) that helps computers learn from images, sound, or text, just like humans do. CNN is a special type of deep learning model that is very powerful in recognizing images and patterns. That is why we have used CNN in our project.

To begin with, we collected a dataset of cricket images showing different batting strokes. We extracted important frames from those images where the player is clearly performing a stroke. Each frame was then labeled according to the stroke being played, like "cover drive," "pull shot," or "sweep." After collecting the data, we preprocessed it by resizing the images to the same size and applying techniques like rotation, flipping, and zooming to increase the variety of data. This helps the model to learn better and become more accurate.

The core of our project is the CNN model. It has several layers that work together to detect the stroke. First, the convolutional layers find features like edges, bat angle, and body movement. Then pooling layers reduce the size of the image while keeping important features. After that, we use dense layers that help the model understand the image and make decisions. In the final layer, we use a function called softmax, which tells the model which stroke is most likely being played in the image.

The dataset used for the project contains thousands of labeled cricket shot images grouped into four classes Pull, Sweep, Drive, and Leg Glance/Flick. These images were resized to 64x64 pixels and converted to grayscale format to standardize the input to the CNN model. Image processing was carried out using OpenCV and PIL libraries. A front-end interface was developed using Python's Tkinter library. This GUI allows the user to load an image, display it on the screen, and then classify it using the trained CNN model. The GUI also includes a login and registration system using SQLite

for authentication. Users can also visualize training accuracy and loss graphs through Matplotlib.

Fig1.1 shows our project “Cricket Stroke Detection Using CNN” is a smart and innovative use of AI in the field of sports. It not only saves time but also provides accurate and automatic analysis of cricket strokes. This project shows how technology can help in improving performance and make sports analysis more efficient and intelligent.

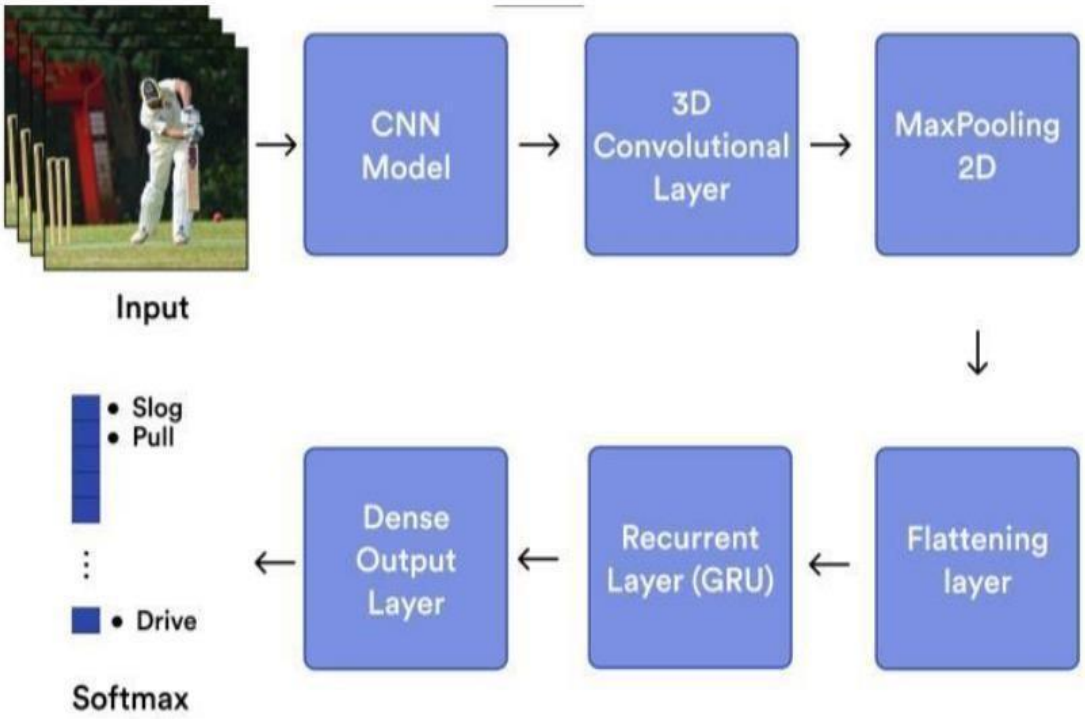

Fig.1.1 System block diagram

1.3 Motivation

Cricket is not just a sport it is an emotion for millions of fans across the world, especially in India. Every cricket lover admires the elegant shots played by batters such as the cover drive, pull shot, or straight drive. These strokes are not only beautiful to watch but also very important for scoring runs and building a good innings. To master these strokes, players need regular feedback, and coaches need to observe their technique very carefully. However, manually analyzing every stroke from images is time-consuming and may also lead to human errors.

With the advancement of technology, especially in the field of Artificial Intelligence (AI) and Deep Learning, it is now possible to automate this process. Our

motivation to work on this project came from the idea of using AI to help in cricket coaching and analysis. We thought if a computer can learn to recognize faces and objects from images, why can't it learn to recognize cricket strokes too?

This inspired us to use Convolutional Neural Networks (CNNs), which are powerful tools for image classification. CNNs can detect patterns, shapes, and features in images, and classify them into different categories. So, by training a CNN model with images of different cricket strokes, we can make it intelligent enough to identify strokes automatically.

Our goal is to make the analysis process faster, easier, and more accurate. This system can support:

- Coaches by giving instant feedback on a player's batting style.
- Young players who may not have access to experienced coaches.
- Cricket academies and training centers to improve their training methods.
- Match broadcasters who want to highlight specific strokes during a live match.

Additionally, our motivation was also based on the idea of solving a real-world problem using technology. As computer engineering students, we always look for opportunities to apply our technical knowledge to practical use cases. This project gave us a chance to combine our love for cricket with our interest in deep learning.

In the future, we can improve the system further by including bowling and fielding action detection, using human pose estimation, and even applying it in live matches. This project is just a starting point towards building smart and intelligent sports analysis systems using AI.

1.4 Problem Definition And Objectives

The project aims to develop an automated and intelligent system capable of detecting and classifying different cricket batting strokes from image frames using advanced deep learning techniques, specifically Convolutional Neural Networks (CNN). The primary objective is to reduce the need for manual analysis by providing an accurate, efficient, and real-time stroke recognition solution. This system is designed to classify various cricket strokes based on visual input, enabling coaches, analysts, and players to receive immediate feedback. A robust CNN model is trained on a diverse and well-labeled dataset to ensure accurate classification even under real-world conditions such as

varying lighting, backgrounds, and motion blur.

One of the key goals is to maintain high accuracy and low latency, making the system practical for live environments like coaching sessions or match analysis. In addition to visual classification, the system offers audio feedback using text-to-speech, making it more accessible and interactive, especially for real-time use. The user interface is designed to be simple and intuitive, allowing users to upload images, view predictions, and access historical logs for deeper analysis. To enhance model generalization and reduce overfitting, data augmentation techniques such as flipping, rotation, and brightness adjustment are applied during training. Lastly, the project emphasizes a modular architecture that allows easy updates and integration with more complex image recognition or sports analytics systems in the future. This combination of real-time performance, accessibility, and modular design ensures that the system is not only functional today but also scalable for future developments.

1.5 Project Scope & Limitations

The scope of the Cricket Stroke Detection Using CNN project encompasses several key areas aimed at building an intelligent, scalable, and practical solution for stroke classification in cricket. The core objective is to detect and classify a variety of batting strokes such as cover drive, pull shot, sweep, and straight drive based on input images. By leveraging the power of Convolutional Neural Networks (CNNs), which are highly effective in extracting spatial features from visual data, the system can accurately identify distinct stroke patterns in cricket images. The project is designed to work with still images, making it suitable for both recorded footage analysis and potential real-time applications, such as live coaching or match broadcasting in the future.

The system covers all the major phases of machine learning model development, including dataset collection, image preprocessing, CNN model training, testing, and performance evaluation. One of its key use cases is in coaching environments, where it can assist players and coaches by automatically recognizing batting strokes, thereby eliminating the need for time-consuming manual video analysis. Furthermore, the system is designed to be expandable, with the potential to evolve beyond batting strokes to recognize other cricket actions like bowling, wicket-keeping, or fielding techniques, making it versatile for broader applications in the sport.

Real-world applications of this project are significant. It can be effectively implemented in cricket academies, sports analysis centers, live broadcasting systems, and training platforms, providing valuable insights and automated feedback. In future iterations, the system can be enhanced with advanced features such as live video processing and pose estimation, which would enable even more accurate and dynamic real-time stroke detection. This broad and future-ready scope makes the project not only impactful in the current context but also adaptable to evolving technological and sporting needs.

Limitations

1. **Dataset Dependency**

   The model's accuracy depends heavily on the quality and size of the dataset. Limited or imbalanced data can reduce performance.

2. **Lighting and Background Variations**

   Changes in lighting conditions, camera angles, and complex backgrounds in images may affect the model's ability to correctly detect strokes.

3. **Limited Stroke Types**

   The current model may recognize only a fixed set of predefined cricket strokes and might not identify new or uncommon stroke variations.

4. **Real-time Processing Constraints**

   Real-time detection requires high computational power and optimized algorithms, which may not be achievable on all devices.

5. **Similar Stroke Confusion**

   Some cricket strokes have very similar movements or bat positions, which can confuse the model and lead to misclassification.

6. **Lack of Pose Information**

   Without integrating human pose estimation, the model relies mainly on visual features and may miss subtle biomechanical differences in strokes.

7. **Environmental Noise**

   Noise such as player motion blur or crowd movement in images can affect stroke detection accuracy.

8. **Hardware Requirements**

   Training and running CNN models require GPUs or high-performance hardware,

which may not be accessible for all users.

1.6 Methodology

The proposed image recognition system follows a vision-based approach utilizing deep learning techniques to identify and classify images in real time. The overall methodology can be divided into the following key stages: The first step in the project is to collect cricket images that show various batting strokes. From these images, we extract key frames where the player is actively performing a stroke. These frames serve as our dataset images. Since raw images may have different sizes and qualities, we preprocess them by resizing to a fixed dimension (for example, 224x224 pixels) to maintain uniformity. We also normalize pixel values to a specific range (such as 0 to 1) to help the CNN model learn better. Finally, we label each image with the correct stroke category (like cover drive, pull shot, sweep) so the model knows what to learn during training.

Deep learning models like CNN require large and varied datasets to perform well. However, collecting a massive dataset of cricket stroke images can be difficult. To overcome this, we apply data augmentation techniques that artificially increase the size and diversity of the dataset. These techniques include rotating the images by small angles, flipping them horizontally, zooming in or out, and adjusting brightness or contrast. Augmentation helps the model become more robust to variations in player position, lighting, and camera angle, which often happen in real-world images.

The core of the project is training the Convolutional Neural Network (CNN) model. CNN consists of multiple layers: convolutional layers extract important features like edges, bat shape, and player posture from the images; pooling layers reduce the spatial size and highlight the strongest features; and fully connected (dense) layers use these features to classify the image into a specific cricket stroke category. We split the dataset into training and testing sets to evaluate model performance. The training process involves feeding the images to the CNN, calculating the prediction error using a loss function, and updating model weights using backpropagation and an optimizer like Adam. We train the model over multiple epochs until the accuracy stabilizes or reaches a satisfactory level.

To make the cricket stroke detection system accessible and easy to use, we develop a web application using the Flask framework. Flask is a lightweight Python web framework that allows integration of machine learning models with a simple web

interface. In the app, users can upload cricket images. The app processes the input, extracts frames if it's a video, and sends these frames to the trained CNN model for stroke classification. The results are then displayed on the web page, showing which stroke was detected in the uploaded media. This interface makes the system practical for coaches, players, and cricket enthusiasts without needing to run code manually.

1.7 Training and Work

As part of our internship, a team of three students collaboratively worked on a project titled "Cricket Shot Detection System Using CNN". The primary goal was to develop an intelligent image classification system capable of detecting various cricket shots such as Pull, Sweep, Drive, and Leg Glance/Flick using deep learning techniques. Guided by our mentor, each team member contributed to different aspects of the project. One member focused on model development, designing and training a Convolutional Neural Network (CNN) tailored for image classification. Another member was responsible for GUI creation, developing the front-end interface using Tkinter to ensure user-friendly interaction. The third member handled database and user flow, implementing authentication using SQLite and organizing the structure and logic of the application. Through this collaboration, we successfully integrated all components into a functional and user-accessible system.

The Cricket Shot Detection System Using CNN was developed using a well-integrated technology stack that combined deep learning, image processing, GUI design, data visualization, and lightweight database management. The core of the system is powered by TensorFlow and Keras, which were used to build and train the Convolutional Neural Network (CNN). Keras provided a high-level API for quickly creating and experimenting with different CNN architectures, while TensorFlow served as the backend engine for model training and performance optimization. For preprocessing and manipulating image data, OpenCV was employed. It allowed efficient resizing, grayscaling, and normalization of cricket images before they were passed to the CNN. In the GUI, the PIL (Python Imaging Library) was used for loading and displaying the uploaded images, enabling seamless integration with Tkinter components.

The graphical user interface (GUI) was developed using Tkinter, Python's standard GUI library. Tkinter made it easy to build user-friendly screens for login,

image upload, prediction, and graph display. It supports event-driven programming and integrates well with other Python modules, making it an ideal choice for desktop-based applications. To manage user authentication (login and registration), SQLite3 was used as the database. It is a serverless, file-based relational database system, suitable for lightweight applications. It helped store and validate user credentials without requiring any external setup.

To evaluate and visualize the model's training performance, Matplotlib was used to plot training and validation accuracy and loss graphs. These visual insights were useful both for understanding model behavior and for demonstration purposes within the GUI. In addition, NumPy played a critical role in handling image arrays and performing numerical operations, such as reshaping and normalization. Other utility modules like OS and tkinter.filedialog were used to access directories, read images, and navigate the file system.

Overall, this technology stack was carefully selected to ensure the system remains lightweight, modular, and scalable, while providing a complete and functional end-to-end solution from data preprocessing to deep learning-based prediction and user interaction.

CHAPTER 2
SOFTWARE REQUIREMENT SPECIFICATION

2.1 Assumption And Dependencies

The Cricket Stroke Detection Using CNN project is built on several fundamental assumptions that define the scope and performance expectations of the system. First, it assumes that the input images are clear and focused, capturing the batsman executing a stroke without any major obstructions, motion blur, or distractions in the frame. This visual clarity is essential for the CNN to accurately detect the spatial patterns associated with each stroke. Secondly, the system is trained on a limited and predefined set of common cricket stroke types such as cover drive, pull shot, and sweep. It does not account for variations or strokes outside of this trained category set, meaning that new or rare stroke types may not be correctly classified.

Another key assumption is that the input images are captured under consistent lighting conditions. Significant shadows, glare, or poor lighting could distort features within the image, leading to incorrect predictions by the model. The project also expects images to be taken from standard cricket viewing angles typically side-on or front-on as these angles are most commonly used in training and provide the best visibility of the stroke. Unusual or extreme camera positions could hinder the model's ability to recognize the correct stroke pattern.

Furthermore, the system assumes that the frame extraction process from video or image sequences accurately captures the moment when the stroke is being played. If irrelevant frames (e.g., before or after the stroke) are used, the model's performance may decline. Finally, the implementation of the system presumes the availability of sufficient hardware capabilities, such as a machine with a decent GPU, to handle the training of the CNN model and allow real-time or near real-time inference. These assumptions are crucial in ensuring the reliability and efficiency of the stroke detection system under practical conditions.

The Cricket Stroke Detection Using CNN project is built upon several software and hardware dependencies that ensure its functionality, efficiency, and performance. The core programming language used is Python, chosen for its extensive libraries and community support in both machine learning and web development domains. For deep learning tasks, the project relies heavily on TensorFlow and Keras, which are used to

design, train, and evaluate the Convolutional Neural Network (CNN) model. Additionally, NumPy is employed for numerical computations and array-based data manipulations during preprocessing stages.

For image-related operations, OpenCV is a critical dependency. It is used to extract frames from images and apply various preprocessing steps such as resizing, grayscale conversion, and filtering. It also supports data augmentation tasks. On the web development side, the project uses Flask, a lightweight Python web framework, to build the user interface that allows image uploads, handles backend logic, and displays stroke prediction results in real time.

To enhance the generalization ability of the CNN model, Keras's Image DataGenerator is used for data augmentation, which applies random transformations like rotation, flipping, zoom, and brightness variation to the training images, simulating diverse real-world conditions. Lastly, from a hardware perspective, the project assumes and benefits from the use of a system equipped with a GPU, particularly one that supports NVIDIA CUDA, to accelerate the training process and allow for efficient model inference. These dependencies collectively support the development, training, and deployment of an accurate and responsive stroke detection system.

2.2 Functional Requirements

The Cricket Stroke Detection Using CNN project aims to revolutionize sports analytics and training by automating the identification of cricket batting strokes using advanced deep learning techniques. The system starts with a user-friendly web interface where users coaches, analysts, or players can upload images capturing various cricket strokes. Upon upload, the system processes the images to extract the most relevant frames that highlight the player's posture and movement during the stroke. This step ensures that only the most informative visual data is passed on for analysis, reducing noise and improving prediction accuracy.

The extracted frames are then subjected to rigorous preprocessing. This includes resizing all images to a consistent input size required by the CNN model, normalizing pixel values to standardize lighting and contrast variations, and applying data augmentation techniques such as rotation, flipping, and cropping to enhance the model's robustness to different playing conditions and camera angles. These refined frames are then passed to a pre-trained Convolutional Neural Network specifically developed to recognize and classify a variety of cricket strokes. The CNN model,

trained on a comprehensive and labeled dataset of professional cricket strokes, is capable of distinguishing between multiple stroke types such as cover drive, square cut, pull shot, leg glance, straight drive, and sweep, among others. Once the model completes its inference, the detected stroke type is displayed in real time on the web interface. The system provides immediate visual feedback, making it useful for live training sessions or post-match analysis. In addition to stroke identification, the platform displays key performance metrics such as the confidence score of predictions, processing time per image, and overall system accuracy to help users evaluate the reliability of results.

To ensure a smooth user experience, the web application is designed with simplicity and accessibility in mind. It supports drag-and-drop uploading, clear navigation paths, and responsive design for use across various devices. Error handling mechanisms are also integrated to manage a range of potential issues whether it's an unsupported file format, blurry or poorly lit images, or prediction failures. Users are guided through corrective steps with clear and informative alerts. Beyond individual predictions, the system can also maintain a session-wise history of analyzed images and their results, enabling users to review and compare multiple strokes over time. This makes it a powerful tool not just for real-time detection, but also for long-term performance tracking and player development. Future enhancements may include video support, stroke speed estimation, and integration with wearable devices for more advanced biomechanics analysis. Overall, this project combines cutting-edge artificial intelligence with practical usability to deliver an impactful, real-world solution for cricket coaching, broadcasting, and sports analytics.

2.3 External Interface Requirements

The system provides a web-based user interface designed to be accessible via common web browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. The interface will be simple and intuitive, allowing users to upload cricket-related media easily. Users can upload either images showing batting strokes by selecting files from their device or by using drag-and- drop functionality. After processing, the interface will clearly display the detected cricket stroke type along with additional information like confidence score if applicable. The UI will also provide real-time visual feedback and allow users to listen to voice announcements of the detected strokes.

The backend processing and model training require a computing environment capable of handling intensive calculations. Ideally, this includes a computer or server equipped with a GPU (Graphics Processing Unit), such as an NVIDIA CUDA-enabled GPU, to accelerate deep learning tasks. The users accessing the web app can do so from various devices including desktop computers, laptops, tablets, or smartphones, as long as they have a modern browser and internet connectivity. The system must also accommodate different screen sizes and resolutions to ensure usability across devices.

The project's core components are built using Python and utilize various libraries and frameworks for functionality. The web application backend is developed with Flask, which manages HTTP requests and communicates with the CNN model for stroke classification. Communication between the UI and backend can be handled through RESTful API endpoints or direct function calls within Flask routes. The software stack includes TensorFlow or Keras for deep learning, OpenCV for images processing, and additional libraries for data augmentation and preprocessing. This modular software design allows easy updates or replacements of components without affecting the overall system.

2.4 Analysis Models: SDLC Model to be applied

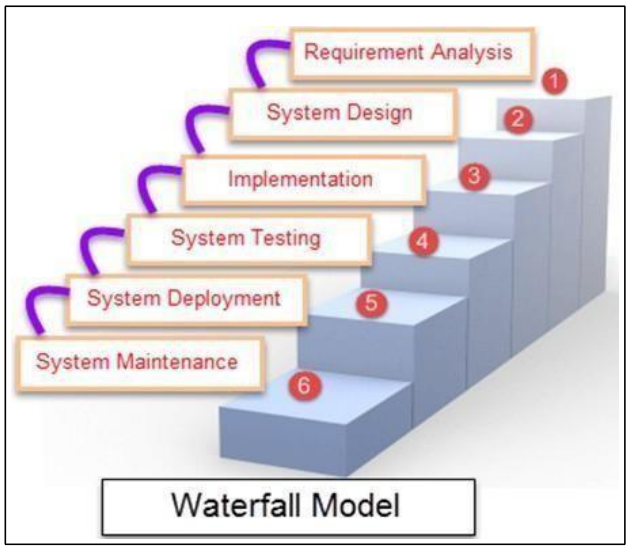

Fig 2.1: SDLC Model Diagram

Fig 2.1 shows the Cricket Stroke Detection Using CNN project, the Waterfall Model is applied as the Software Development Life Cycle (SDLC) model. The Waterfall Model is a linear and sequential approach, where each phase must be completed before moving on to the next. This model is suitable for projects with well-defined requirements and

objectives, as it emphasizes planning, documentation, and disciplined execution. In this phase, all the functional and non-functional requirements of the system are gathered. For this project, it involves understanding the objectives such as detecting and classifying cricket strokes using CNN, real-time result display, and handling various input images. Stakeholders (like coaches or analysts) define what the system should achieve. Based on the requirements, the overall architecture and design of the system are created. This includes deciding on the deep learning framework (e.g., TensorFlow/Keras), the CNN architecture, data flow diagrams, web application layout, and integration points between front-end and back-end components. In this phase, the actual coding and development of the system take place. The web application is built using suitable technologies (such as HTML, CSS, JS, and Flask or Django), and the CNN model is developed and trained using labeled cricket stroke images. This also includes integrating the model with the application for prediction functionality.

Once implemented, the system undergoes rigorous testing. This includes Unit Testing for individual components (e.g., CNN, image preprocessing) Integration Testing for checking model and web interface interaction Functional Testing to validate the stroke detection results Performance Testing to ensure real-time processing. Post-deployment, the system enters the maintenance phase. This involves updating the model as more data becomes available, fixing bugs, improving prediction accuracy, and ensuring the application remains secure and efficient over time. By applying the Waterfall Model, the development of the Cricket Stroke Detection System follows a structured path with clear milestones and deliverables at each phase. This model ensures that all essential components from requirements to maintenance are addressed methodically, making it ideal for academic or early-stage projects where requirements are stable and clearly understood.

CHAPTER 3

SYSTEM DESIGN

3.1 System Architecture

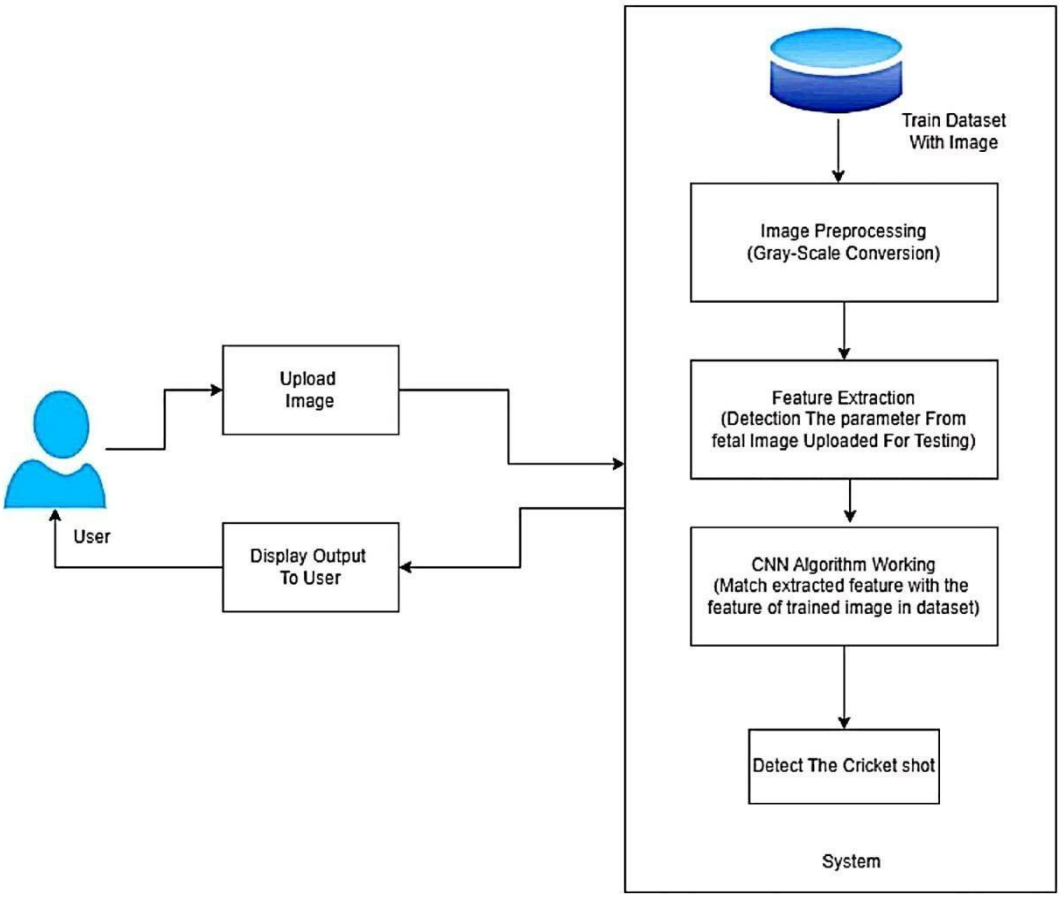

Fig 3.1: System Architecture

Fig 3.1 illustrates the process of sign recognition using a deep learning model. The steps involved the process begins when the user provides an image as input to the system. This image typically captures a cricket player performing a batting stroke. These images can be collected from live footage, training sessions, or recorded matches. This step is crucial, as the quality and content of the input image directly affect the accuracy of the stroke classification model. Before feeding the image into the deep learning model, it undergoes several preprocessing steps to improve its quality and make it suitable for training or prediction. These steps help the model focus on important visual features and reduce irrelevant information or inconsistencies. In this step, the colored input image is converted to a grayscale image. This is done because color information is often

not necessary for detecting shapes or patterns in cricket strokes. Grayscale images contain only intensity information (brightness), which simplifies the data by reducing it from three color channels (RGB) to one. This helps decrease computational complexity while retaining essential structural details like outlines and motion.

This technique enhances the contrast of the grayscale image. Histogram equalization spreads out the most frequent intensity values, making features in the image more distinguishable. For example, the edges of a bat, the player's limbs, or the ball might become clearer. This step improves visibility in images with poor lighting or uneven exposure, ensuring that the model receives clearer inputs for learning or prediction. Normalization is the process of scaling pixel values to a standard range, typically between 0 and 1. This is done by dividing all pixel values (which typically range from 0 to 255) by 255. Normalization ensures that all input data is on the same scale, which is essential for deep learning models to converge properly during training. It also helps prevent large pixel values from disproportionately influencing the model's learning process.

3.2 Data Flow Diagram

In Data Flow Diagram Figure 3.2, 3.3 and 3.4, we Show that flow of data in our system in DFD0 we show that base DFD in which rectangle present input as well as output and circle show our system, In DFD1 we show actual input and actual output of system input of our system is text or image and output is rumor detected like wise in DFD 2 we present operation of user.

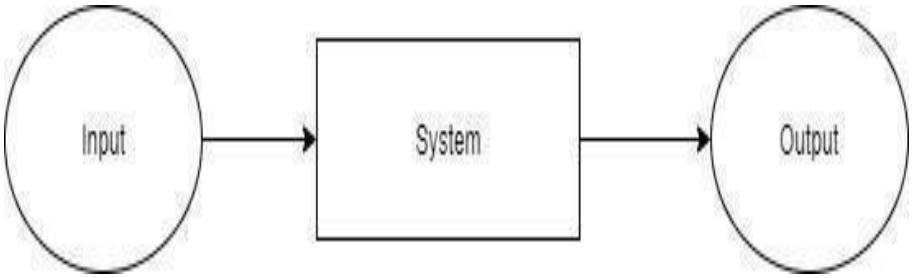

Fig 3.2: Data Flow diagram 1

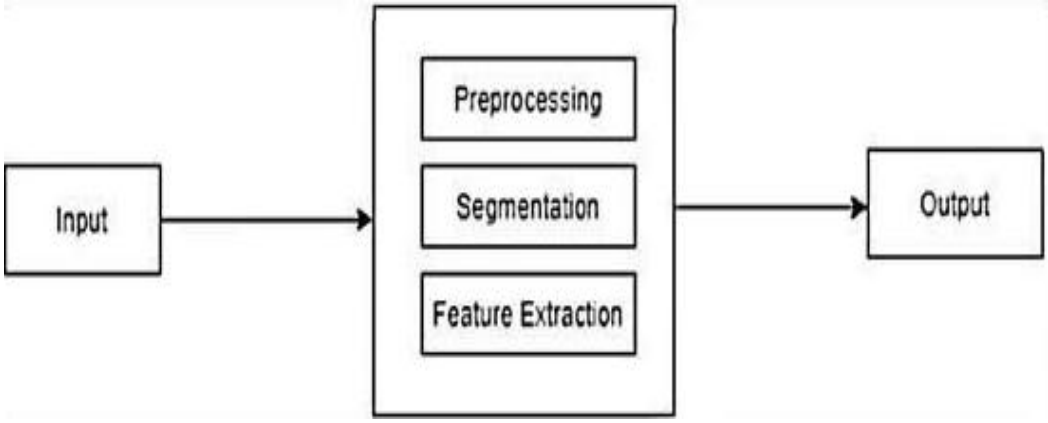

Fig 3.3: Data Flow diagram 2

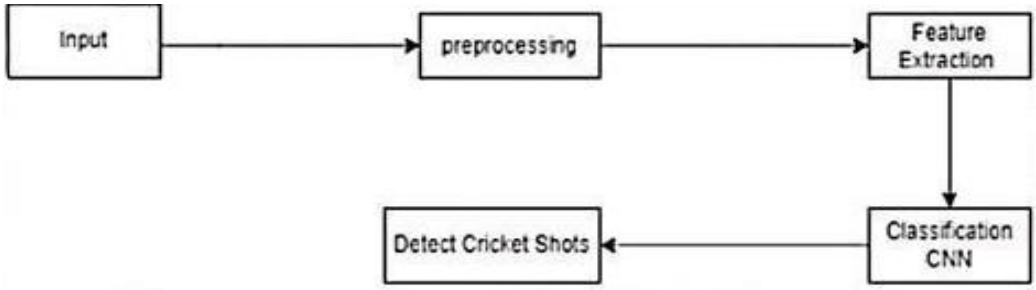

Fig 3.4: Data Flow diagram 3

Input: This is the starting point of the process. The input is typically a raw image such as a photograph or a frame from a video containing the object of interest. In your project, this would be an image showing a cricketer performing a batting stroke. Preprocessing: This is the first internal step, where the raw input image is prepared for further analysis Simplifies the image by removing color, retaining only intensity information. Enhances contrast to make features like bat and body outline clearer. Reduces unwanted data that may affect accuracy. Scales pixel values to a uniform range (usually 0 to 1), which improves the learning performance of deep learning models. Segmentation: In this stage, the image is divided into meaningful regions or parts. The goal of segmentation is to isolate the region of interest such as the player's body, bat, or specific motion patterns from the rest of the image. Separating objects based on pixel intensity. Edge Detection Identifying boundaries of shapes like arms or bat. Segmentation helps focus the analysis on the relevant portions of the image, ignoring background or irrelevant details. Feature Extraction: Once the relevant segments are isolated, the next step is to extract features key pieces of information that help identify patterns or objects in the image. After preprocessing, segmentation, and feature extraction, the final result or output is generated. In your project, this

would be the predicted cricket stroke type (e.g., cover drive, pull shot, sweep). The output may also include a confidence score or other relevant information for feedback or analysis.

3.3 Class Diagram

This fig 3.5 represents the internal structure and workflow of the Cricket Stroke Detection Using CNN project, illustrating how different components of the system interact to process and classify a cricket batting stroke from an image. The process begins with the User Image class, which handles the image input from the user. This image, containing a cricket stroke, is then passed to the Preprocessing class where it undergoes enhancement techniques such as grayscale conversion, normalization, and noise reduction to improve its quality and prepare it for further analysis. After preprocessing, the image moves to the Feature Extraction class, where essential visual characteristics such as edges, bat orientation, and body posture are identified and isolated. These extracted features are then sent to the Segmentation class, which divides the image into meaningful parts, such as separating the player and bat from the background, allowing the system to focus only on relevant regions of interest.

Once segmentation is complete, the resulting image is sent to the Classification class, where a Convolutional Neural Network (CNN) is used to analyze the segmented image and classify the type of cricket stroke being performed, such as a cover drive, pull shot, or sweep. The output of the classification is then passed to the Result class, which is responsible for displaying or storing the final prediction. This class provides feedback to the user by showing the identified cricket stroke, potentially along with other metrics like prediction confidence. Overall, this diagram reflects a clear, modular design where each class performs a specific role in a linear, step-by-step process, making the system easier to build, debug, and extend. It effectively combines image processing, machine learning, and user interaction to automate the recognition of cricket strokes from visual input. The output of the classification is then passed to the Result class, which is responsible for displaying or storing the final prediction. This class provides feedback to the user by showing the identified cricket stroke, potentially along with other metrics like prediction confidence. Overall, this diagram reflects a clear, modular design where each class performs a specific role in a linear, step-by-step process, making the system easier to build, debug, and extend. It effectively combines image processing, machine learning, and user interaction to automate the recognition of cricket strokes from visual input.

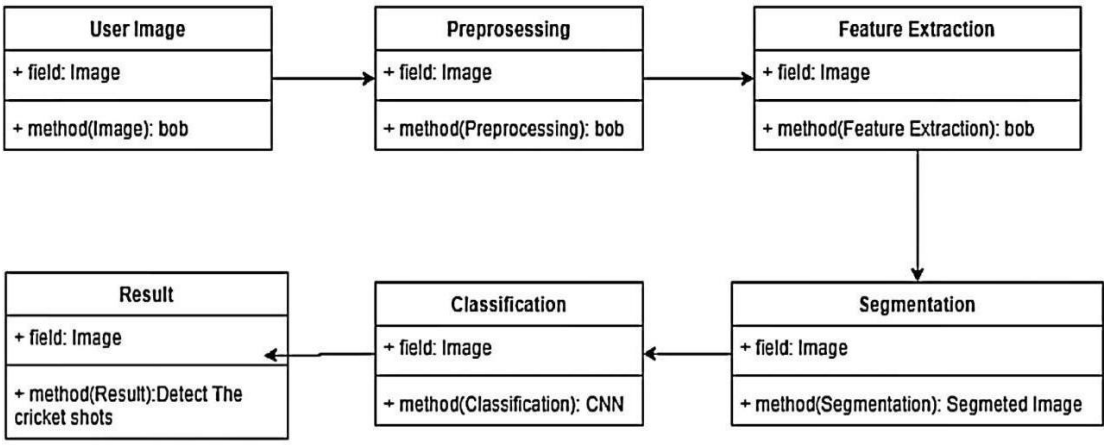

Fig 3.5 Depicts the flow of data in the system / Class Diagram

3.4 UML Diagrams

Unified Modeling Language is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct and document the artifacts of a software intensive system. UML is process independent, although optimally it should be used in process that is use case driven, architecture-centric, iterative, and incremental. The Number of UML Diagram is available.

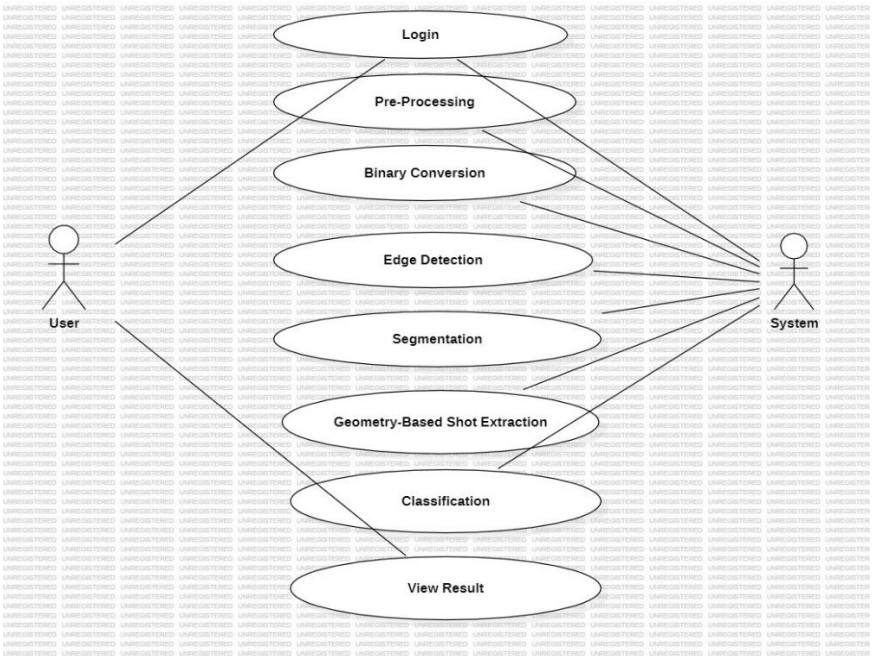

Fig 3.6 Use case Diagram

Figure 3.6 An use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. This diagram outlines the major functional steps involved, from user interaction to final result generation. It begins with the Login/Registration step, where the user accesses the system through secure authentication. Once logged in, the user provides an image input, triggering the Pre-Processing stage, where the image is prepared for analysis by cleaning and resizing. Following this, the system performs Contrast Enhancement to improve the visibility of important features such as the bat and player movements. Then, the image undergoes Binary Conversion, simplifying the visual data by converting it into a binary format (black and white), which makes further processing more efficient. Edge Detection is the next step, helping the system identify the boundaries and outlines of key objects like the bat, arms, or ball. After edge detection, the Segmentation module isolates the regions of interest such as the player’s body and bat from the background to focus analysis on relevant parts. This is followed by Extraction of Geometry-Based Shots, where the system analyzes geometric features like bat angle, hand position, and swing pattern to better understand the stroke's nature. Next, the image is passed to the Classification phase, where a Convolutional Neural Network (CNN) processes the extracted features and classifies the type of cricket stroke like cover drive, pull shot, or sweep. Finally, the Result is generated and presented to the user, indicating the detected stroke along with any relevant prediction metrics.

The use case diagram for the Cricket Shot Detection System outlines the key interactions between the user and the system. It starts with Login/Registration, allowing secure access. Once logged in, the user uploads an image, triggering the Pre-Processing phase where the image is cleaned and resized. Then, Contrast Enhancement improves visibility, followed by Binary Conversion to simplify data. Edge Detection identifies outlines, and Segmentation isolates key regions like the bat and player. In Geometry-Based Shot Extraction, features such as bat angle and swing are analyzed. These features are then passed to the Classification module, where a CNN identifies the type of cricket shot (e.g., drive, pull, sweep). Finally, the Result is displayed to the user, showing the predicted shot type and accuracy.

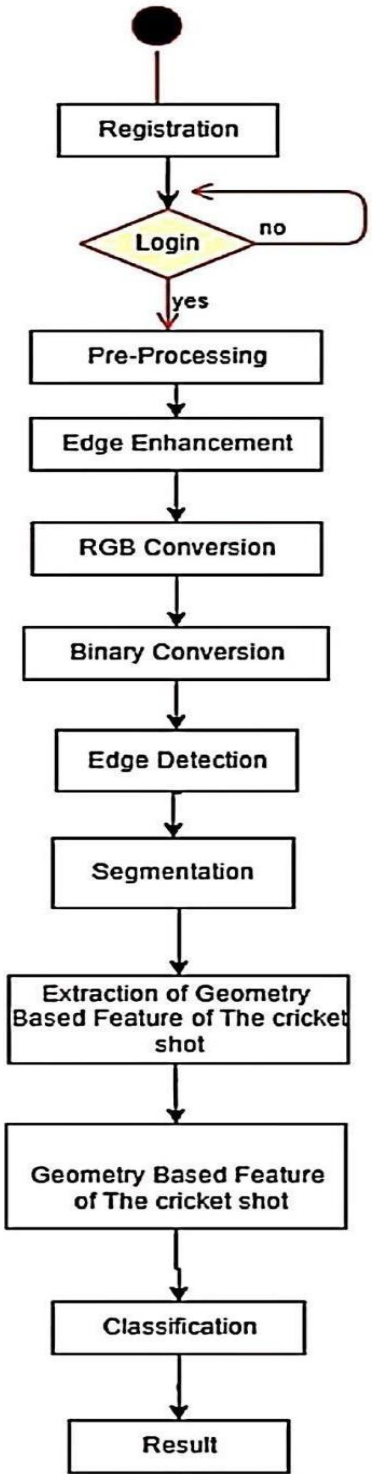

Fig 3.7: Activity Diagram

Figure 3.7 Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. The activity diagram represents the complete workflow of the Cricket Stroke Detection Using CNN project, outlining the sequential steps involved from user interaction to final stroke detection.

The process begins with the user accessing the system through a Registration step, followed by Login. If the login credentials are invalid, the system redirects the user back to the login prompt; if successful, the system proceeds to the image processing stages. The first technical step is Pre-Processing, where the input image is cleaned and prepared by removing noise and adjusting quality. This is followed by Edge Enhancement, which sharpens the edges of important objects within the image, such as the bat, ball, or limbs of the player. Next, the image undergoes RGB Conversion to ensure consistent color representation, followed by Binary Conversion, where the image is transformed into black and white to simplify the analysis and highlight essential shapes. After that, Edge Detection is applied to identify the boundaries and contours of relevant objects in the image. These detected features are passed to the Segmentation stage, where the system isolates areas of interest particularly the player and bat from the background. Once the image is segmented, the system performs Extraction of Geometry-Based Features, such as bat angle, arm position, and body posture. These features are then organized into a structured format referred to as the Geometry-Based Feature of the Cricket Shot, which serves as the input for the deep learning model.

The system then enters the Classification stage, where a Convolutional Neural Network (CNN) analyzes the extracted features and classifies the type of cricket stroke being performed, such as a pull shot, straight drive, or sweep. Finally, the outcome of this classification is presented to the user as the Result, providing accurate stroke identification along with any associated metrics. Overall, this activity diagram illustrates a well-structured, end-to-end process that combines image processing techniques with intelligent machine learning to achieve accurate and efficient cricket stroke detection.

CHAPTER 4
PROJECT IMPLEMENTATION

4.1 Overview of Project Modules

Data Collection

The dataset used in the Cricket Stroke Detection Using CNN project consists of a collection of cricket stroke image frames, each labeled with a specific stroke type such as cover drive, pull shot, straight drive, and others. These labeled images serve as the foundation for training the deep learning model to accurately recognize and classify different batting strokes. To prepare the images for effective model training and prediction, a series of image processing steps are applied. First, frames are extracted from the input images using OpenCV, a powerful computer vision library that allows for precise handling of visual data. Once the frames are extracted, they are resized to a fixed dimension, typically 224×224 pixels, to ensure uniformity across the entire dataset. This resizing is important because deep learning models require inputs of consistent shape.

Depending on the model architecture, the images may be either converted to grayscale, which simplifies the data and reduces computational requirements, or retained in RGB format to preserve color information that might be relevant to detecting subtle visual features. After this, normalization is performed on the pixel values, scaling them to a standard range of [0, 1]. This step improves model convergence during training and ensures that all pixel data is on the same scale, which is essential for neural networks to function effectively. Overall, this image processing pipeline is crucial as it brings consistency to the input format, enhances the accuracy and learning capacity of the model, and significantly reduces the computational cost during both training and inference stages.

CNN Model Development

The architecture design for the Cricket Stroke Detection Using CNN project involves a custom-built Convolutional Neural Network (CNN) tailored specifically for extracting meaningful features from cricket stroke images and classifying them into various stroke types. The CNN architecture is composed of several key layers that each play a vital role in the learning process. Convolutional layers form the core of the network and are responsible for detecting important spatial features and patterns in the input images,

such as the position of the bat, arm movement, and body orientation. These features are critical for distinguishing between strokes like cover drive, pull shot, and straight drive. Following the convolutional layers, Max Pooling layers are included to perform down-sampling, which reduces the spatial dimensions of the feature maps. This not only minimizes computational complexity but also helps retain the most prominent features while discarding irrelevant details.

To improve generalization and reduce the risk of overfitting where the model performs well on training data but poorly on unseen data Dropout layers are added. These layers randomly deactivate a portion of neurons during training, forcing the network to learn more robust and distributed representations. After sufficient feature extraction, the output from the convolutional layers is flattened and passed through Fully Connected layers (Dense layers), which combine all learned features to make a final classification decision about the type of cricket stroke. The entire model is implemented and trained using TensorFlow and Keras, two powerful and widely-used deep learning frameworks. These tools provide high-level APIs and efficient back-end computation, making it easier to build, train, and fine-tune the CNN model for optimal performance. Overall, this architecture is designed to efficiently capture the complexities of cricket stroke images and deliver accurate, real-time predictions.

Model Training

For the Cricket Stroke Detection Using CNN project, the dataset is strategically split to ensure effective training, validation, and testing of the model. Initially, 80% of the total data is allocated for the training phase, where the model learns to recognize patterns and features associated with different cricket strokes. Out of this training set, 20% is further reserved for validation, which allows for real-time monitoring of the model's performance during training and helps in tuning hyperparameters and preventing overfitting. In addition to this, a separate and untouched dataset is maintained for final testing to evaluate the model’s performance on completely unseen data, ensuring an unbiased assessment of its generalization capability.

A highly efficient gradient descent algorithm that combines the benefits of both AdaGrad and RMSProp, adapting the learning rate for each parameter to improve convergence speed and performance. The categorical cross-entropy loss function is used, which is ideal for multi-class classification problems like stroke detection, where the output can be one of several stroke types (e.g., cover drive, pull shot, etc.). Once training

is complete and the model achieves satisfactory accuracy, it is saved in a deployable format such as h5,keras or .pkl. These formats are compatible with various platforms and frameworks, enabling seamless integration of the trained model into the main web application for real-time cricket stroke detection.

Integration with Flask Web App

The web interface for the Cricket Stroke Detection Using CNN project is designed to be simple, responsive, and user-friendly, allowing seamless interaction between the user and the underlying machine learning model. The backend of the application is developed using Flask, a lightweight Python web framework that efficiently handles HTTP requests, routes, and server-side logic. On the frontend, HTML and CSS are used to build the structure and style of the interface, with the option to incorporate Tailwind CSS for modern, utility-first design components that enhance the visual appeal and responsiveness of the web pages.

Users are provided with flexible image input options, enabling them to either upload a single image or a frame extracted from a video. This ensures versatility, accommodating both still image analysis and frame-by-frame review of match footage. Once an image is uploaded through the interface, the system automatically loads the pre-trained CNN model and initiates real-time stroke classification. The uploaded image is passed through the same preprocessing pipeline used during model training, and the model predicts the type of cricket stroke depicted in the image. The predicted result is then displayed instantly on the interface, offering users immediate feedback. This real-time inference capability makes the web application highly practical for coaching, analysis, or demonstration purposes.

Model Evaluation and Improvement

In the Cricket Stroke Detection Using CNN project, performance evaluation plays a crucial role in assessing the effectiveness and reliability of the model. To monitor training progress and detect issues like overfitting or underfitting, accuracy and loss graphs are plotted over each epoch. These graphs help visualize how well the model is learning and how its performance changes over time. Additionally, a confusion matrix is used to gain deeper insight into the classification results by comparing actual stroke labels with predicted ones.

This matrix highlights where the model performs well and where it may be confusing one stroke type with another, making it a valuable diagnostic tool. To further enhance the model's accuracy and robustness, several improvements were implemented. Data augmentation techniques such as flipping, rotation, and scaling were applied to artificially expand the dataset and expose the model to a wider variety of stroke positions and angles, which improves generalization. Dropout layers were added as a form of regularization, reducing the risk of overfitting by randomly disabling a portion of the neurons during training, which forces the model to learn more generalized features. Additionally, hyperparameter tuning was performed to optimize training settings like the number of epochs, batch size, and learning rate. By carefully adjusting these parameters, the model's convergence and overall prediction accuracy were significantly improved, resulting in a more stable and reliable cricket stroke detection system.

Image Preprocessing for Prediction

In the Cricket Stroke Detection Using CNN project, every image or frame uploaded through the web application is subjected to the same preprocessing steps that were applied during the model's training phase. This is essential to ensure that the input format remains consistent and compatible with what the model expects. The first step is to resize the uploaded image to a fixed dimension, such as 224×224 pixels, which aligns with the input shape used during model training. This standardization allows the model to process inputs efficiently without shape mismatches.

If the model was trained on grayscale images, the uploaded image is also converted to grayscale during preprocessing. This reduces the input complexity by removing color channels and retaining only intensity information, focusing the model on patterns and edges rather than color. The final preprocessing step is normalization, where pixel values are scaled to a range between 0 and 1. This step ensures that all input values are on a uniform scale, which is critical for stable and effective neural network performance. By applying these preprocessing steps consistently during both training and inference, the system ensures reliable, accurate, and reproducible predictions across all user-uploaded images.

Cricket Stroke Prediction

Once the uploaded image has been preprocessed, it is passed into the Convolutional Neural Network (CNN) model for prediction. The model analyzes the image and returns two key outputs: the predicted stroke type and the prediction probability. The predicted

stroke type refers to the cricket shot that the model has identified such as a cover drive, pull shot, or straight drive based on the visual features extracted from the image. Alongside this, the model also provides a prediction probability, which indicates the model's confidence in its prediction. This is usually a value between 0 and 1, where a higher value signifies greater confidence.

These results are then displayed on the user interface in real time. To ensure a clear and engaging user experience, the results are presented with appropriate styling such as bold text, color-coded labels, or cards making the output both informative and visually appealing. This allows users to not only see which stroke has been detected but also understand how confidently the model arrived at that decision, helping in better analysis and interpretation of results, especially in training or coaching scenarios.

Text Retrieval

In the Cricket Stroke Detection Using CNN project, after the model predicts a class index (such as Class 4), this numeric output is mapped back to a corresponding stroke name and its detailed description to make the result meaningful and easy to understand for users. This mapping is done using a reference file like a labels.csv, which associates each class index with a human-readable stroke name (e.g., "Straight Drive") and a concise stroke description (e.g., "A powerful front-foot shot played along the ground straight down the pitch"). This step is crucial because raw class numbers alone have no intuitive meaning to users. By translating the index into descriptive text, the system enhances interpretability and usability. This ensures that users not only see which stroke has been detected but also gain a quick understanding of what the stroke involves. This feature is especially beneficial for players, coaches, and analysts who may be using the application for training or performance review, as it provides immediate visual and contextual feedback about the detected cricket action.

Once the mapping is complete, both the stroke name and its description are clearly displayed on the user interface alongside the predicted image. This ensures that users not only see which stroke has been detected but also gain a quick understanding of what the stroke involves. This feature is especially beneficial for players, coaches, and analysts who may be using the application for training or performance review, as it provides immediate visual and contextual feedback about the detected cricket action.

4.2 Tools and Technologies Used

In the Cricket Stroke Detection Using CNN project, several key technologies and tools are utilized to build an efficient and accurate image classification system. OpenCV (Open Source Computer Vision Library) plays a vital role in the image processing phase. It is an open-source library widely used for real-time computer vision tasks. In this project, OpenCV is used to extract frames from cricket images, and to preprocess those images by resizing them to a fixed size (such as 224×224 pixels), converting them to grayscale if required by the model architecture, and normalizing pixel values to prepare them for input into the CNN model. OpenCV's comprehensive functionality makes it ideal for tasks like reading, writing, filtering, and analyzing image data efficiently.

To build and train the deep learning model, the project uses TensorFlow and Keras. TensorFlow, developed by Google, is a powerful open-source framework designed for creating and deploying machine learning and deep learning applications. Keras, which is a high-level API built on top of TensorFlow, simplifies the process of developing neural networks with easy-to-use interfaces. In this project, these tools are used to design, train, and deploy a Convolutional Neural Network (CNN) that can classify various cricket strokes from the input image frames. The model architecture includes convolutional layers for feature extraction, pooling layers for dimensionality reduction, dropout layers to prevent overfitting, and fully connected layers for final classification.

The entire project is implemented using Python, a high-level and versatile programming language that is highly popular in AI, machine learning, and web development domains. Python is used for data preprocessing, CNN model development, and the creation of a Flask-based web application that serves as the user interface. It also supports integration of features such as text-to-speech feedback, allowing the system to provide verbal responses to detected strokes using available Python libraries.

To improve model performance and reduce the risk of overfitting, the project employs data augmentation techniques. These techniques increase the diversity of the training data by applying random transformations to input images, helping the model learn more robust features. Using Keras's built-in utilities, transformations like rotation, shifting, flipping, and zooming are applied to the dataset. This not only expands the dataset but also ensures that the model is exposed to a wide variety of visual scenarios,

making it more generalizable to real-world cricket stroke images. Together, these tools and techniques form a cohesive, intelligent system for real-time cricket stroke detection and classification.

4.3 Algorithm Details

The Cricket Stroke Detection Using CNN project follows a structured, step-by-step development process to ensure accurate stroke recognition and smooth user interaction. The first phase, Step 1: Data Collection and Preprocessing, begins by gathering a dataset of cricket images that includes various labeled stroke types such as cover drive, pull shot, and straight drive. Using OpenCV, frames are extracted from these images with attention to clarity and variation to capture the nuances of each stroke. The images are then resized to a fixed dimension (e.g., 224×224 pixels) to ensure uniformity and compatibility with the CNN's input layer. To reduce computational complexity, images can optionally be converted to grayscale, especially if color information is not critical for classification. Finally, all pixel values are normalized to the range [0, 1], standardizing the input and enhancing the model's training performance.

In Step 2: CNN Model Development, a custom Convolutional Neural Network (CNN) architecture is designed specifically for image classification tasks. The model incorporates essential layers including convolutional layers to extract spatial features from the images, max pooling layers to reduce the dimensionality of the feature maps while retaining important features, dropout layers to mitigate overfitting by randomly deactivating neurons during training, and fully connected layers to process the extracted features and produce the final classification output. This model is implemented using Keras, a high-level deep learning API, with TensorFlow serving as the backend.

Moving to Step 3: Model Training, the processed dataset is divided into three subsets: 80% for training, 20% of that training data for validation, and a separate 20% of the full dataset reserved for final testing. The model is trained using the Adam optimizer, which adapts learning rates for each parameter, and the categorical cross-entropy loss function, ideal for multi-class classification problems. Dropout is used as a regularization method to improve the model's generalization. After training, the final model is saved in a deployable format such as .h5, .keras, or .pkl for future use and integration.

In Step 4: Integration with Web Application, a user-friendly interface is built using Flask for the backend and HTML/CSS (optionally styled with Tailwind CSS) for

the frontend. This web application allows users to either upload a cricket image or capture one using a webcam. Upon submission, the uploaded image is processed using the same preprocessing functions as those used during training to maintain input consistency. The trained model then performs prediction on the processed image, and the system displays the predicted stroke type and its description on the user interface in real time, offering a seamless and interactive user experience.

Finally, in Step 5: Cricket Stroke Label Repository, a mapping file such as stroke_labels.csv is maintained. This file links each numeric class index predicted by the model to a corresponding stroke name and a concise stroke description. For example, Class 2 may map to "Pull Shot" with the description "A horizontal bat stroke played against short-pitched deliveries." This mapping enhances the interpretability of the model's output by transforming raw predictions into meaningful, human-readable information, making the system not only intelligent but also user-centric and informative.

In Step 6: Model Evaluation and Improvement, the trained CNN model is evaluated using a separate test dataset to assess its performance. Key metrics such as accuracy, loss, and optionally a confusion matrix are calculated to measure how well the model can generalize to unseen data. To further improve performance and reduce overfitting, data augmentation techniques such as rotation, flipping, zooming, and brightness adjustments are applied to artificially expand the training dataset. Additionally, real-world testing is conducted using images captured from a webcam or mobile device to validate how the model performs in practical scenarios. Based on these tests, hyperparameters such as learning rate, batch size, and number of epochs are fine-tuned for optimal accuracy and reliability.

In Step 7: Image Preprocessing for Prediction, every new image whether uploaded or captured through the interface is processed using the same preprocessing pipeline used during training. This includes resizing the image to 224×224 pixels, optionally converting it to grayscale (if that was part of the model training), and normalizing pixel values to the [0, 1] range. These steps ensure that the input during inference matches the input format the model was trained on, maintaining consistency and prediction accuracy.

Next, in Step 8: Stroke Prediction, the preprocessed image is passed to the trained CNN model using a function such as model.predict(image), which returns the predicted class index and the confidence score or probability of the prediction. This

output helps in determining which stroke the model believes is present in the image and how confident it is in that classification. In Step 9: Text Retrieval, the numeric prediction is mapped to a human-readable stroke name and description using a label mapping file (e.g., stroke_labels.csv). This file connects each class index to meaningful text, such as "Cover Drive" along with a one-line explanation of the stroke, which is then displayed on the web interface.

Finally, in Step 10: Text-to-Speech Conversion, the retrieved text is converted into spoken output using the pyttsx3 library. This Python library enables text-to-speech synthesis, allowing the system to audibly communicate the detected stroke name and its description. Commands like engine.say(label_text) and engine.runAndWait() are used to speak the result. Additionally, user controls can be added in the web interface to adjust volume, speech rate, and enable/disable voice output, making the application more interactive and accessible for a wider range of users, including those with visual impairments or those seeking an audio-based experience. This comprehensive flow ensures accurate prediction, informative feedback, and an engaging user experience.

As shown in Fig.4.1 the proposed CNN Model is crafted to identify images. It employs a series of convolutional and pooling layers to detect spatial patterns and hierarchies effectively. Additionally, we integrate batch normalization and dropout layers to mitigate overfitting issues and enhance the model's ability to generalize to unseen data. This diagram illustrates the Convolutional Neural Network (CNN) architecture used in the Cricket Stroke Detection project. The architecture is structured to effectively extract visual features from cricket images and classify different types of strokes, such as cover drive, pull shot, or straight drive.

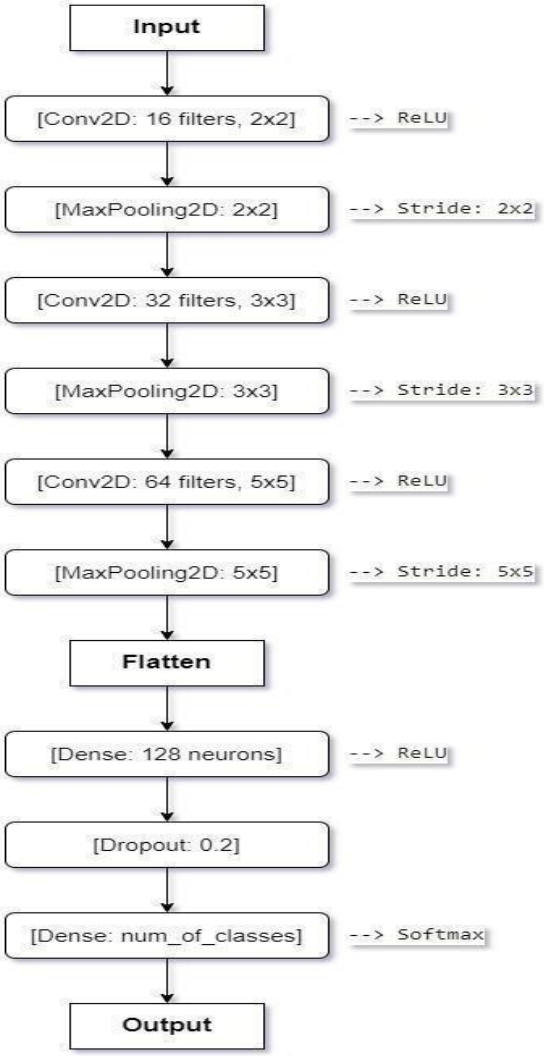

Fig 4.1: Proposed CNN Model

After these convolution and pooling operations, the output feature maps are flattened into a one-dimensional vector to prepare them for the fully connected layers. The Dense layer with 128 neurons applies another ReLU activation and serves as the first fully connected layer, learning high-level combinations of the features extracted earlier. To prevent overfitting, a Dropout layer with a rate of 0.2 is included, which randomly disables 20% of the neurons during training to improve generalization. The final layer is a Dense output layer with the number of neurons equal to the total number of stroke classes (e.g., 4, 5, or more), followed by a Softmax activation function that produces a probability distribution over the classes. Finally, the output represents the predicted stroke type, selected as the class with the highest probability. This structured CNN architecture is optimized for image classification and plays a central role in accurately identifying cricket strokes in the system.

CHAPTER 5

SOFTWARE TESTING

5.1 Type Of Testing

1. Unit Testing

Unit testing is the process of testing the smallest parts of the application independently to ensure that each component performs its intended function. In this project, unit testing is used to test image preprocessing functions such as resizing, grayscale conversion, and normalization. For example, the function that resizes images to 32x32 pixels is tested separately to confirm that it consistently gives the correct output size. The normalization function is tested to make sure it properly converts pixel values to the [0, 1] range. These small tests help detect errors early in the development process. The CNN model's prediction function can also be tested independently to verify that it returns class probabilities in the expected format. Python libraries like unit test or pytest are commonly used to perform such unit tests.

2. Integration Testing

Integration testing focuses on verifying that different modules or components of the system work together seamlessly. In the cricket stroke detection system, integration testing ensures that the image preprocessing pipeline works correctly with the CNN model, and that the prediction output is properly passed to the web application and the text-to-speech module. For instance, an uploaded image should go through preprocessing, classification, and then voice feedback – all without any failure or incorrect transition. This type of testing confirms that the interaction between modules is stable and efficient, which is essential for providing real-time stroke recognition results on the web platform.

3. Black Box Testing

Black box testing is performed from the user's point of view, without any knowledge of the internal structure or code of the system. The goal is to ensure that the application functions correctly under various conditions. In this project, black box testing is used to test the overall system behavior, such as uploading a cricket stroke image and checking whether the correct stroke is predicted and announced. It also involves checking how

the system handles invalid input, such as uploading a blank or corrupted image. This type of testing helps verify that the system meets its requirements and provides a smooth user experience.

4. White Box Testing

White box testing involves testing the internal logic, code structure, and workflows of the application. It is used to verify whether all possible paths and branches in the code execute correctly. In the cricket stroke detection project, white box testing can be used to check whether all CNN layers are properly connected and functioning. It also includes verifying conditional logic in the Flask web application, such as how the system behaves when no image is uploaded or when the prediction fails. This kind of testing ensures that the internal code is optimized, well- structured, and free from hidden bugs.

5.2 Test Cases And Test Results

Test Case ID	Test Case	Test Case I/P	Actual Result	Expected Result	Test case criteria(P/F)
001	Enter The Wrong username or password click on submit button	Username or password	Error comes	Error Should come	P
002	Enter the correct username and password click on submit button	Username and password	Accept	Accept	P

Fig 5.1 Test Cases 1

The test case table shown in fig 5.1 is designed to validate the login functionality of the Cricket Stroke Detection web application. It includes two key test scenarios to ensure the system behaves correctly when users enter either invalid or valid credentials. In the first test case (Test Case ID: 001), the user enters an incorrect username or password and clicks the submit button. The test input is invalid login credentials, and the expected outcome is that the system should display an error message indicating the login attempt was unsuccessful. The actual result matches this expectation as an error appears, and therefore, the test case is marked as Pass (P).

In the second test case (Test Case ID: 002), the user provides the correct username and password and then clicks the submit button. In this scenario, the system is expected to accept the login attempt and grant access. As observed, the actual result aligns with the expected outcome the login is successful and this test case is also marked as Pass (P). Overall, both test cases confirm that the authentication mechanism of the application functions correctly. It appropriately denies access for incorrect credentials and allows access for valid ones, ensuring both security and user experience are upheld. This testing step is essential in the overall system validation process to ensure reliable login behavior before deploying the system for end users.

CHAPTER 6

RESULTS

6.1 Screenshots

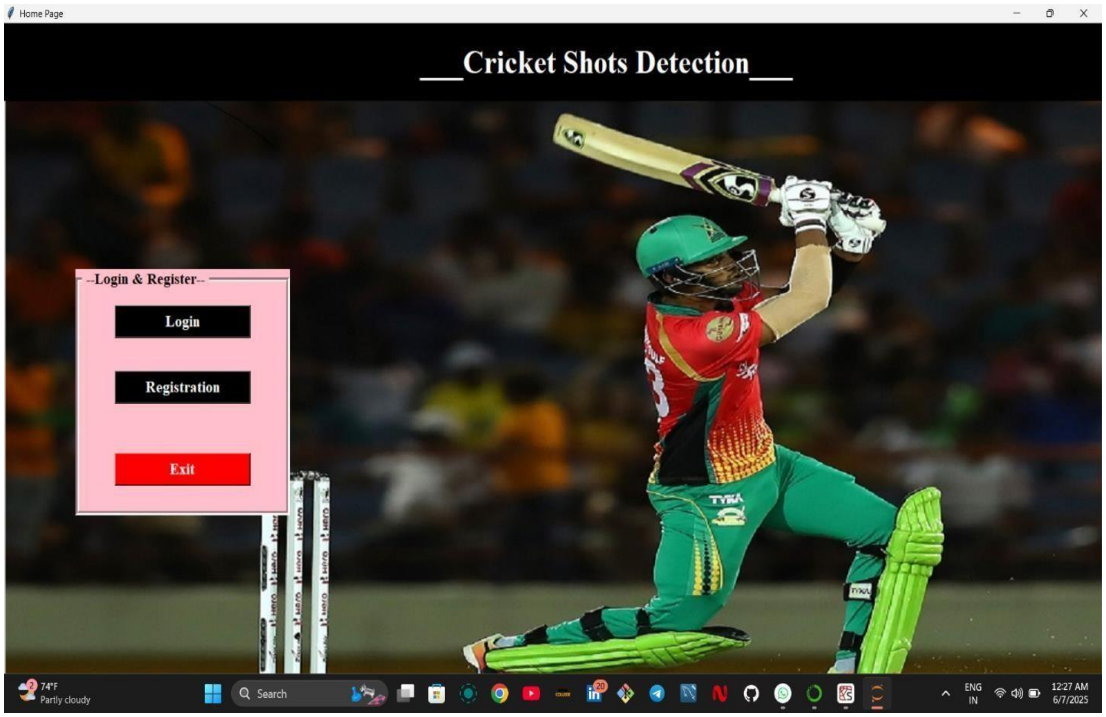

Fig 6.1 Output Screenshot: Initial Page

Fig 6.1 Shows the homepage or main GUI screen of a web application titled "Cricket Shots Detection." It appears to be the initial interface presented to users upon launching the web application, and it combines both a functional menu and an engaging visual background to create an intuitive user experience. Main Title: Centered in white and underlined text is the project title: "Cricket Shots Detection", clearly indicating the purpose of the application. Background Image: A high-resolution action image of a professional cricketer playing a cricket stroke is used as the background. This visually reinforces the core function of the application detecting and classifying batting strokes. Login/Register Panel: On the left-hand side of the screen, there's a dedicated rectangular box labeled "Login & Register", which contains the primary navigation buttons: Login Button: Clicking this button would likely redirect the user to a login form, where they can enter their username and password to access the system. Registration Button: This button probably opens a registration form allowing new users to create an account by providing their credentials and other necessary information. Exit Button: This red button is designed to close the application or exit the

interface when clicked.

Login: This function ensures that only registered users (e.g., administrators, analysts, or authorized personnel) can access the core features of the system such as image upload, stroke detection, and result visualization. Registration: Allows new users to sign up and store their information in the system's backend (likely stored in a database). It ensures user authentication. Exit: Safely closes the GUI and terminates the program, preventing any unnecessary resource usage or accidental window clutter.

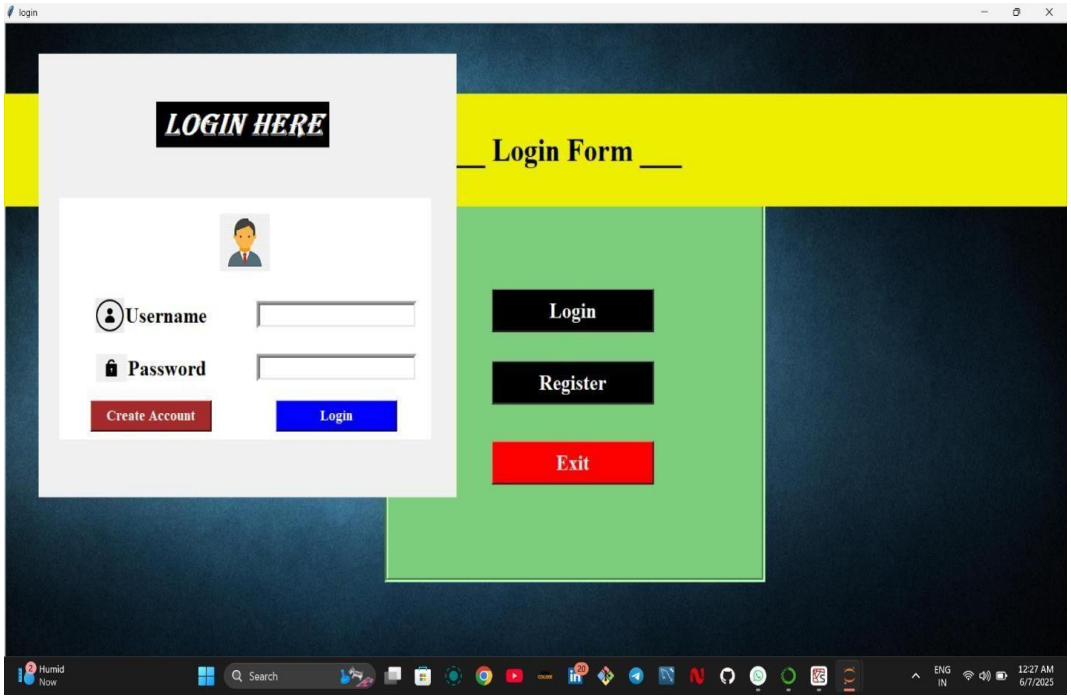

Fig 6.2 Output Screenshot: Login Page

Fig 6.2 displays the Login Interface of the Cricket Shots Detection application. It is designed to authenticate users before allowing access to the system's core features. The interface blends visual design with functional elements and is most likely built using Python's Tkinter library. This interface is crucial for user authentication and session management. By requiring login/registration, the system ensures, Only authorized users can upload images and perform stroke detection. It enhances security and enables logging of user actions, which is helpful for debugging or maintaining a usage history.

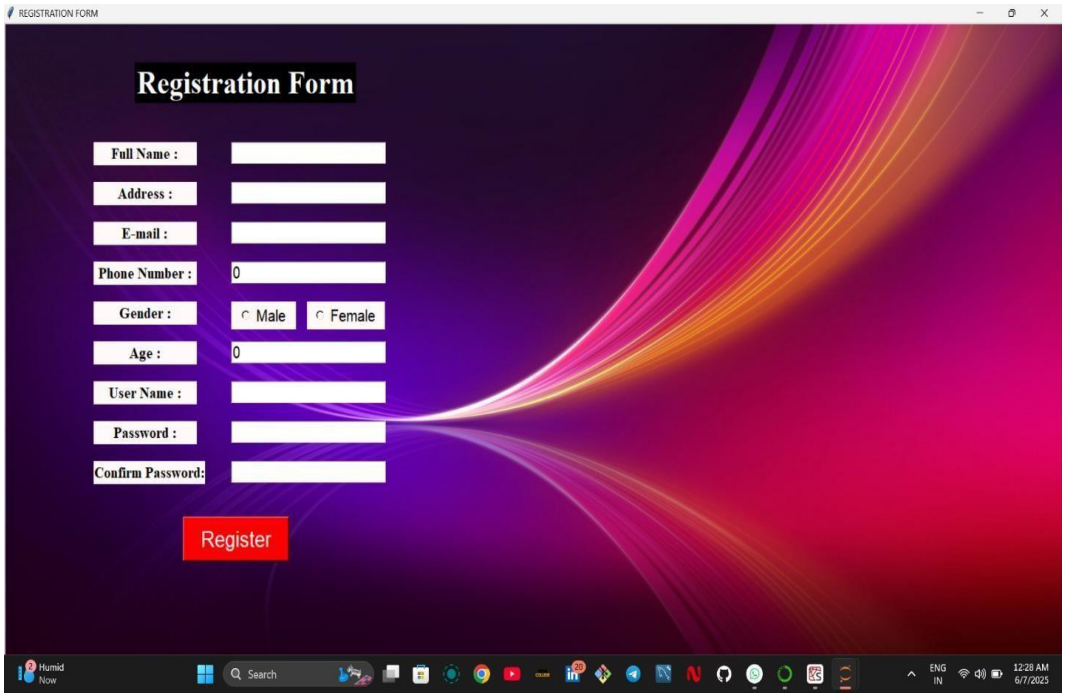

Fig 6.3 Output Screenshot: Registration Page

Fig 6.3 displays the Registration Form interface of the Cricket Shots Detection. This form serves as the primary entry point for new users to register themselves before accessing the main system features. At the top, the form is titled “Registration Form,” clearly indicating its purpose. Title Bar displays "REGISTRATION FORM" indicating the current active screen is the registration window. This registration interface is critical for enabling user management in the system. It supports: Secure login flow with unique user credentials, Data association, where uploaded images or classification results may be linked to specific users. This Registration Form GUI is the initial access point for new users in the Cricket Shots Detection project. By collecting basic personal and login details, it ensures a secure, personalized experience while laying the foundation for session handling, user-specific logging.

Fig 6.4 below showcases the main prediction interface of the Cricket Shots Detection. This screen appears after a user logs in and performs an image classification task to detect cricket strokes. The application interface is titled "Cricket Shots Detection", and it includes a structured layout for processing, viewing, and classifying images using a Convolutional Neural Network (CNN). On the left-hand side, there's a process panel that contains several function. Select Image allows the user to upload a cricket stroke image from their local system. Image preprocess applies preprocessing techniques such as resizing, grayscale conversion, and binary thresholding to prepare

the image for the model. Train model Triggers the training process (if enabled) using the CNN architecture to learn patterns from labeled stroke data. CNN prediction executes the trained CNN model to classify the uploaded image and predict the stroke type.

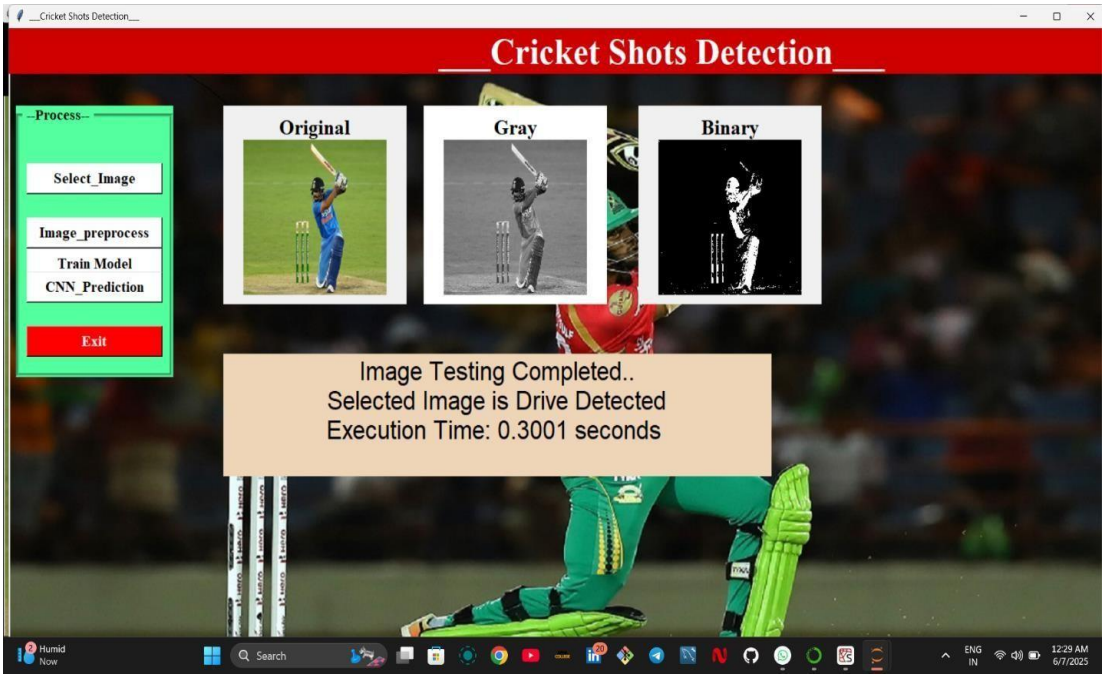

Fig 6.4 Output Screenshot: Result

At the center of the interface, three image panels display the image in different stages of processing original: The raw input image as uploaded by the user. Gray: The grayscale version of the image, where colors are removed to simplify features for analysis. Binary: The final binary (black and white) version used by the CNN for detection, where pixels are thresholded to highlight the silhouette or shape of the batsman. Below the image panels is a message box that displays the output of the CNN prediction. The text “Image Testing Completed” confirms successful execution, while the “Selected Image is Drive Detected” indicates that the predicted stroke is a drive shot. It also shows the execution time (0.3001 seconds), which reflects the system’s fast inference capability.

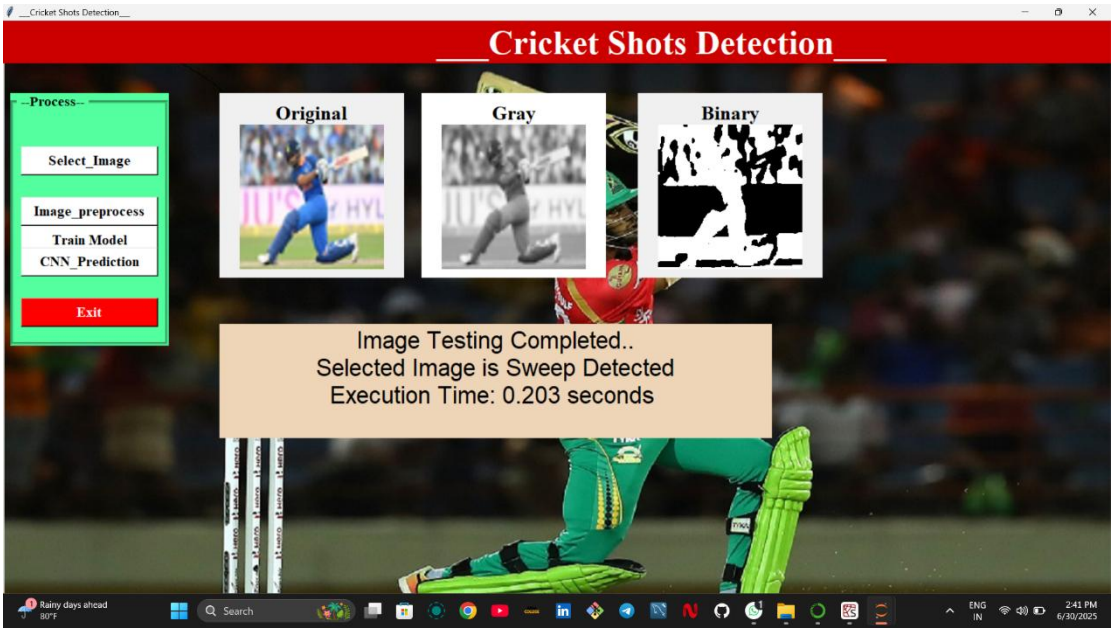

Fig 6.5 Output Screenshot: Result

Fig 6.6 Output Screenshot: Result

CONCLUSION

The process described in this proposes a technique for automatically analyzing cricket shots that uses deep learning strategies to reach a significantly better result. Convolutional Neural Networks are used in the described technique to utilize the video input of each cricket shot being played. A dataset of numerous distinct shots performed by a batsman is used to train the CNN model. The dataset is first preprocessed, and the preprocessed pictures are then normalized before being fed into the CNN model for training. Once the trained model is complete, it is utilized to do the testing utilizing input video that has been adequately preprocessed and normalized without first being exposed to CNN detection. To perform the cricket shot assessment, the results are efficiently categorized utilizing the decision- making process.

The development of the *Cricket Shot Detection System* represents the successful integration of machine learning, computer vision, and user interface design in a practical and engaging project. The use of a *Convolutional Neural Network (CNN)* for image-based classification of cricket shots has proven to be effective and showcases how deep learning can enhance the field of sports analytics. Throughout the internship period, a complete end-to-end system was developed from preprocessing cricket images to classifying them into specific shot types and displaying the result in a GUI.

REFERENCES

[1] A S. Rao, J. Gubbi, S. Marusic, and M. Palaniswami, "Crowd Event Detection on Optical Flow Manifolds," in IEEE Transactions on Cybernetics, vol. 46, no. 7, pp. 1524-1537, July 2024, DOI: 10.1109/TCYB.2015.2451136.

[2] D.Tang, "Hybridized Hierarchical Deep Convolutional Neural Network for Sports Rehabilitation Exercises," in IEEE Access, vol. 8, pp. 118969-118977, 2024, DOI: 10.1109/ACCESS.2020.3005189.

[3] TensorFlow and Keras Documentation
https://www.tensorflow.org
https://keras.io

[4] Kaggle Community Discussions & Datasets – Used as a source of understanding dataset formatting and preprocessing.

[5] Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). "ImageNet Classification with Deep Convolutional Neural Networks." Advances in Neural Information Processing Systems, 25.