PokerEye

AI-Powered Poker Hand Recognition System

Product of Aydie's Avenue

Aydie's Avenue

aydie.in

Mangalore University

St Philomena College, Puttur

Detailed Project Report On

PokerEye

AI-Powered Poker Hand Recognition System

Submitted in partial fulfilment of the requirements for the Degree of
Bachelor of Science (B.Sc)

Aditya Dinesh K (U05PH21S0028)
Jithesh B (U05PH21S0032)
Ahammad Sabik (U05DP21S0115)
Team Code: MINPRO24001

Under the Guidance of

Prof. Varija Das
Department of Computer Science
St Philomena Collage, Puttur, DK, 574202

Department of Computer Science
St Philomena Collage, Puttur, DK, 574202

2023 - 2024

St Philomena College

Philonagara, Darbe, Puttur, 574202

Dept of Computer Science

CERTIFICATE

Certified that this is a bonafide record of the Mini Project Work entitled **"Poker Eye"** carried out by Aditya Dinesh K(U05PH21S0028), Jithesh B (U05PH21S0032) and Ahammad Sabik (U05DP21S0115) of III B.Sc during the academic year 2023-24 for the partial fulfilment of the requirement to award a Bachelor of Science(B.Sc) Degree by Mangalore University.

Prof. Varija Das
Head of the Dept

APPROVED

PRINCIPAL

Submitted to the University Examination on _____ at St Philomena College, Puttur Examination Center.

EXAMINERS

Project External

Examiner Supervisor

Declaration

We hereby declare that the Mini Project Work entitled “Poker Eye,” submitted to Mangalore University, is an original work conducted by us under the guidance of Prof. Varija Das during the academic year 2023-24. This report is submitted to the University in partial fulfillment of the requirements for the award of the Bachelor of Science degree. We confirm that this dissertation is an authentic record of the work carried out by us and has not been submitted previously to any other institute or university.

Aditya Dinesh K
UUCMS: U05PH21S0028

Jithesh B
UUCMS: U05PH21S0032

Ahammad Sabik
UUCMS: U05DP21S0115

Acknowledgement

We extend our sincere gratitude to everyone who directly or indirectly assisted us in completing this project. Without their support, this achievement would not have been possible.

Our heartfelt thanks go to Rev. Dr. Antony Prakash Monteiro, Principal of St. Philomena College, Puttur, for his unwavering support. We are especially grateful to Prof. Varija Das, Head of the Computer Science Department, for her invaluable guidance and assistance. We also appreciate all the professors and college staff for their support and for providing the necessary information.

Special thanks to our friends for their help, support, and constructive feedback. Lastly, we thank our parents for their financial and emotional support, and we acknowledge each other's teamwork and dedication, which made this success possible.

Date:
Place: Puttur

Aditya Dinesh K

Jithesh B

Ahammad Sabik

Index

Sl	Content	Page
1	Introduction	1
2	Objective	2
3	Project Category and Methodology	2
4	System Architecture	2
5	Process Description	3
6	Hardware and Software Requirements	4
7	Training Data	5
8	MongoDB	14
9	What is Poker and more about it	17
10	Context Flow Diagram	19
11	Data Flow Diagram	21
12	Detailed Analysis of the Code and Algorithm	24
13	PokerEyeDetector.py (Main Function)	24
14	PokerHandFunction.py	26
15	MongoDBTest.py	29
16	Running the Model	30
17	Scope of the Solution	32
18	Work Plan and Timeline	33
19	Expected Outcomes	35
20	Accuracy Testing	37
21	Limitations	39
22	Future Work	41
23	Conclusion	43

1. Introduction

Poker, a game of strategy and chance, hinges on players' ability to recognize and evaluate their hand combinations. Traditionally, this relies on visual identification of cards, which can be time-consuming and prone to errors, especially in fast-paced games. This project proposes an innovative solution - an AI-powered system that automates poker hand recognition in real-time using computer vision and machine learning techniques.

1.1 Background and Significance

Poker has evolved from a leisure activity to a competitive sport with professional leagues and online tournaments. Mastering hand recognition is crucial for strategic decision-making. However, relying solely on human observation introduces limitations. Players might struggle with:

- **Speed:** Identifying complex hand combinations like straight flushes or full houses can be challenging under time pressure.
- **Accuracy:** Fatigue, distractions, or rapid gameplay can lead to misidentification of cards, potentially impacting strategic decisions.
- **Consistency:** Maintaining peak focus and accuracy throughout a long game is demanding.

1.2 Problem Addressed

This project addresses the aforementioned limitations by developing an AI system that automatically recognises poker hands from a live video stream. This eliminates the need for manual identification, offering several advantages:

- **Increased Speed:** Real-time hand recognition allows quicker decisions and improved gameplay efficiency.
- **Enhanced Accuracy:** The AI model, trained on a vast dataset of poker card images, can identify hands with high accuracy, minimizing errors.
- **Reduced Fatigue:** The system frees players from the mental strain of constant hand recognition, allowing them to focus on higher-level strategies.
- **Improved Consistency:** The model maintains consistent performance throughout a game, regardless of external factors.

2. Objectives

The primary objective of this project is to develop a functional AI system capable of recognizing standard poker hand combinations (e.g., straight flush, full house) displayed in a live video feed. To achieve this, the project will:

- To leverage Python programming language for development, commonly used in AI applications.
- To employ deep learning techniques to train an AI model capable of analyzing and classifying poker hand images.
- To integrate the trained model with a computer vision system to capture real-time video input from a webcam or external camera.
- To design a user-friendly interface to display the identified hand combination in real time.

3. Project Category and Methodology

This project falls under the category of Artificial Intelligence (AI), specifically **Computer Vision**.

3.1 Development Approach

- **Programming Language:** Python is a versatile language widely used for AI development due to its extensive libraries and ease of use.
- **Model Training:** Deep learning techniques, a subset of AI, will be employed to train the model. Popular deep-learning frameworks like Ultralytics and OpenCV will be utilized. These frameworks provide pre-built functions and algorithms for efficient model development.
- **Computer Vision:** OpenCV, an open-source library for computer vision, will be used to capture video input from a camera and process the video frames for analysis by the AI model.
- **Development Life Cycle (SDLC):** The Agile methodology will be adopted for project development. Agile promotes an iterative and flexible approach, allowing for continuous improvement and adaptation based on testing and feedback.

4. System Architecture

The system will comprise three key modules that interact seamlessly to achieve hand recognition ***(see Figure 4.1)*:**

- **Video Capture Module:** This module utilises OpenCV to capture real-time video from a webcam or external camera. Each video stream frame will be extracted and sent for further processing.

- **AI Model:** The core of the system, this AI model is trained on a dataset of labelled poker card images. The model analyzes each video frame, identifying the hand combination present based on its training data.
- **User Interface:** This module displays the hand combination recognized by the AI model in real-time. Players can easily view the identified hand combination for strategic decision-making.

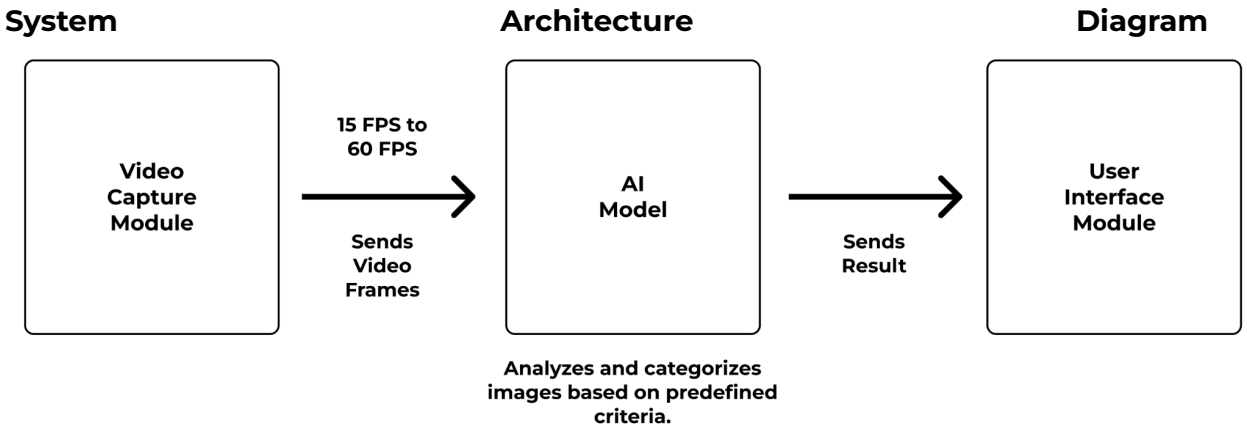


Figure: 1 System Architecture Diagram

5. Process Description

The system operates through a sequential process:

1. **Video Capture:** The Video Capture Module initiates by capturing video frames from the chosen camera source (webcam or external camera).
2. **Pre-processing:** Each captured frame undergoes pre-processing to prepare it for analysis by the AI model. This may involve resizing the frame to a standard size, normalizing colour variations, and applying image filtering techniques to enhance the clarity of the cards.
3. **Model Prediction:** The pre-processed frame is then fed into the AI model. The model analyzes the frame, comparing it to the patterns and features learned from the training data. Based on this comparison, the model predicts the most likely hand combination present in the frame.
4. **Result Display:** The predicted hand combination is sent to the User Interface module. This module translates the model's prediction into a human-readable format, such as text or symbols, and displays it on the user's screen. Users can instantly see the identified hand combination in real-time, aiding their strategic decisions.

6. Hardware and Software Requirements

6.1 Software

- **Development Environment:** Python (version 3.10.0)
- **Development IDE:** Pycharm, Google Collab, Jupyter Notebook
- **Deep Learning Library:** YoloV8, PyTorch, Ultralytics, Hydra-Core
- **Computer Vision Library:** OpenCV, CVZone, TorchVision
- **MongoDB (Database) Library:** PyMongo 4.7.3
- **Additional Libraries:** Matplotlib, Numpy, Pillow, PyYAML, Requests, Scipy, Torch, Tqdm, Filterpy, Scikit-image, Lap

6.2 Hardware

This document outlines the system requirements for training and running a neural network on both Windows and macOS. These specifications are recommended only if you are training and running the model on a local computer; no server hardware is used.

6.3 Windows System:

- **Training:**
    - Minimum 8GB of RAM
    - 6-core CPU (or equivalent processing power)
    - NVIDIA GTX GPU (or equivalent dedicated graphics card, RTX recommended for faster training)
- **Running:**
    - Minimum 4GB of RAM
    - 4-core CPU
    - GPU Recommended (GTX 1650 or higher for faster and smoother execution)
- **Training Time:**
    - 50 epochs (iterations): 6-8 hours (depending on CUDA GPU architecture)

6.4 Mac System:

- **Training: (Recommended)**
    - Apple Silicon Chip with 8-core CPU, 10-core GPU (Metal Architecture), 10-core Neural Engine, and 16GB of Unified Memory (8GB minimum)
- **Training: (Alternative)**
    - Intel Mac with dedicated Radeon GPU (CUDA compatible) with 8GB of VRAM

- **Running:**
    - 8GB Unified Memory with M1 Chip (Metal Architecture) or 8GB RAM with Intel i5 9th Gen (CUDA Architecture)
- **Training Time:**
    - 50 epochs (iterations): 8-10 hours (depending on Metal GPU architecture)

6.5 Resources used during running the model:

- GPU: 5% to 15%
- CPU: 5% to 8%

6.6 Training IDE and Resources

- **Google Collab:** It's essentially a cloud-based version of Jupyter Notebook. Instead of installing it on your computer, you access Colab through a web browser. This means you can work on your project from any device with an internet connection, and you don't need a powerful computer of your own. Plus, Colab sometimes offers access to special hardware like GPUs that can significantly speed up certain tasks. So, while both Colab and Jupyter Notebook let you combine code and explanations in a single document, Colab offers the convenience of cloud-based access and potentially more powerful computing resources.
- **Google Drive:**
    - Store & access: Keep your Colab notebooks (like Jupyter notebooks) and project data in Drive for easy access from anywhere.
    - No data juggling: Upload datasets to Drive and use them directly in Colab, eliminating repetitive downloads.
    - Share your work: Save results (graphs, reports) to Drive for effortless sharing with collaborators.

7. Training Data

The AI model's success hinges on the training data's quality and quantity. This project will require a comprehensive dataset of poker card images labelled with their corresponding hand combinations. Here are some considerations for data collection:

- **Image Variety:** The dataset should encompass a wide range of card combinations, including high cards, pairs, two pairs, three-of-a-kind, straight flushes, full houses, four-of-a-kind, straight, flush and royal flush.
- **Image Quality:** Images should be clear and well-lit with minimal background clutter to ensure accurate model training.

- **Image Variation:** The dataset should include images captured from different angles and distances to simulate real-world scenarios where cameras might not be perfectly positioned.
- **Data Augmentation Techniques:** Techniques like image flipping, rotation, and colour jittering can artificially expand the dataset and improve the model's generalisation capabilities. This means the model will be able to recognize hands even if they appear slightly different from the training data.

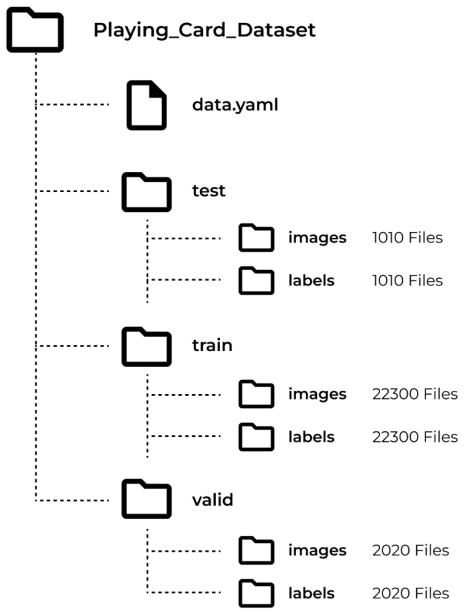
7.1 Training Datasets

Ultralytics is a leader in the field of computer vision, providing state-of-the-art solutions for object detection and image analysis. Their robust datasets are a cornerstone for AI and machine learning advancements, offering a wealth of data to support diverse projects. In my project, I utilized a comprehensive dataset from Ultralytics consisting of 42,000 images of playing cards. This dataset is meticulously organized, featuring a data.yaml file, 42,000 label files, and 12,000 training files. Such a detailed dataset is invaluable for training models with high precision and accuracy. By employing this dataset, my project aims to improve the accuracy and efficiency of algorithms for card recognition and categorization.

The availability of many images, coupled with precise labels and extensive training data, enables the development of robust models capable of handling real-world scenarios. The structure and quality of the dataset significantly streamline the training process, allowing for the creation of models that are both accurate and reliable. Ultralytics' datasets are essential for pushing the boundaries of computer vision technology, and their playing cards dataset is a prime example of how comprehensive data can drive innovation and improve the performance of AI applications in practical, real-world contexts.

7.2 Files of Training Datastes

We store our dataset in a designated Google Drive folder to seamlessly integrate it with our Google Colab Notebook. This centralized storage system ensures easy access to the dataset directly from the notebook environment, facilitating efficient data loading and manipulation during model development. By utilizing Google Drive as our storage solution, we maintain organization and accessibility, streamlining our workflow for effective collaboration and experimentation in our projects.



(The Playing_Card_Dataset is meticulously organised within Google Drive. It consists of categorised subfolders: test, train, and valid. Each of these subfolders contains both an "images" folder, housing the respective images, and a "labels" folder, containing the corresponding label files. Additionally, the dataset includes a data.yaml file at the root level, providing essential metadata. This structured approach ensures efficient access and management of the playing card data for seamless integration into AI and machine learning projects.)

Figure: 7.2 Training Dataset Folder

7.2.1 Data.yaml

The data.yaml file serves as a crucial configuration file, specifying essential information about the dataset. It defines the paths to the training, validation, and testing data, facilitating easy access within the code. Additionally, it specifies the number of classes (nc) and their corresponding names, providing essential metadata for model training and evaluation. Moreover, the file includes parameters for integration with external platforms like Roboflow, streamlining data management and collaboration. Overall, the data.yaml file ensures consistency, clarity, and efficiency in handling the dataset, enhancing the reproducibility and scalability of the machine learning pipeline.

Data.yaml Code:

```
path: ../drive/MyDrive/DataSets/Playing_Cards
train: ../train/images
val: ../valid/images
test: ../test/images

nc: 52
names: ['10C', '10D', '10H', '10S', '2C', '2D', '2H', '2S', '3C', '3D', '3H', '3S',
'4C', '4D', '4H', '4S', '5C', '5D', '5H', '5S', '6C', '6D', '6H', '6S', '7C', '7D',
'7H', '7S', '8C', '8D', '8H', '8S', '9C', '9D', '9H', '9S', 'AC', 'AD', 'AH', 'AS',
'JC', 'JD', 'JH', 'JS', 'KC', 'KD', 'KH', 'KS', 'QC', 'QD', 'QH', 'QS']

roboflow:
  workspace: augmented-startups
  project: playing-cards-ow27d
  version: 4
  license: Public Domain
  url: https://universe.roboflow.com/augmented-startups/playing-cards-ow27d/dataset/4
```

7.2.2 Test, Train, Valid Folder

The "test" folder is utilized for assessing data accuracy post-training. The "train" folder houses new data for model training, referencing the test and validation sets. Lastly, the "valid" folder serves for data validation purposes, ensuring the robustness and reliability of the trained models. These folders collectively support the iterative process of model development and evaluation.

1. **Test folder:** This folder is essential for evaluating the performance of the trained YOLO model. It contains a separate set of images that the model has not seen during training or validation. By testing the model on unseen data, we can assess its ability to generalize to new scenarios and detect objects accurately.
2. **Train folder:** The train folder is where the bulk of the training data resides. These images are used to train the YOLO model to recognize and localize objects within the images. The training process involves iterating through the images multiple times (epochs), and adjusting the model's parameters to minimize the difference between predicted and ground truth bounding boxes and classes.
3. **Valid folder:** Similar to the test folder, the valid folder contains images that are reserved for validation purposes. During training, a portion of the training data is set aside for validation to monitor the model's performance on unseen data and prevent overfitting. The model is periodically evaluated on the validation set to determine if it is improving or if adjustments to the training process are necessary.

In summary, the test, train, and valid folders play distinct yet complementary roles in the training of YOLO models using Ultralytics' algorithms. They collectively contribute to the iterative process of model development, evaluation, and refinement, ultimately leading to the creation of accurate and reliable object detection systems.

7.3 Data Training using Google Collab

We utilize Google Colab for training our data, leveraging its powerful cloud-based computing resources and collaborative features to streamline the model training process effectively and efficiently.

7.3.1 Steps to train:

Selecting the runtime engine: In our project, we use Google Colab to train our models due to its powerful cloud-based computing capabilities and collaborative environment. To optimize performance, we select a GPU runtime, as GPUs are designed to handle intensive computational tasks more efficiently than CPUs. This choice significantly accelerates the data processing and model training phases.

To configure this, we navigate to the runtime settings in Google Colab and change the hardware accelerator to a Tesla T4 GPU. The Tesla T4 GPU offers enhanced computational power, which is crucial for processing large datasets and performing complex calculations required for training deep learning models. This setup not only speeds up the training process but also ensures that our models are trained more effectively, improving their accuracy and performance.

By leveraging Google Colab's GPU resources, we can handle the extensive computations involved in our project with greater ease and efficiency. This approach allows us to iterate faster, test different models and parameters, and ultimately achieve better results in a shorter time frame. The combination of Google Colab's collaborative features and the computational power of the Tesla T4 GPU makes it an ideal platform for our machine learning and data science tasks.

Code to check if the runtime:

```
!nvidia-smi
```

Output:

```
                    Thur April 25 13:40:07 2024
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   50C    P8              10W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

- **Mounting the Google Drive Folder:**

```
from google.colab import drive
drive.mount('/content/drive')
```

Importing the Drive Module: The code starts by importing the **drive** module from the **google.colab** package. This module includes functions specifically designed to handle interactions with Google Drive.

Mounting Google Drive: The **drive.mount('/content/drive')** function call mounts your Google Drive to a specified directory in the Colab environment, in this case, **/content/drive.**

Authorization Process: When the function is executed, it prompts the user to authorize access to their Google Drive. This involves following a provided link, signing into a Google account, and copying the authorization code back into the Colab notebook. This step ensures secure access to your files.

Accessing Google Drive Files: After authorization, the contents of your Google Drive become accessible from within Colab. You can navigate through directories, read from, and write to files in Google Drive as if they were part of the local file system in Colab.

Persistent Storage: Mounting Google Drive provides persistent storage, meaning files stored in Google Drive are retained across different Colab sessions. This is particularly useful for large datasets and model files that need to be accessed or saved regularly.

Enhanced Collaboration: By integrating Google Drive, it becomes easier to share files with collaborators. Shared files in Google Drive are readily accessible, facilitating seamless collaboration on projects hosted in Colab.

Large Storage Capacity: Google Drive typically offers more storage space than the local storage available in Colab. This is beneficial for handling large datasets, storing extensive output files, or maintaining versions of trained models.

Typical Use Cases: Researchers and developers commonly use this setup for machine learning and data science projects. It allows for easy access to training datasets, storage of intermediate results, and saving of final models. It also supports running experiments that require persistent and extensive storage solutions.

- **Package Installation**

The command **!pip install ultralytics** uses the pip package manager to install the Ultralytics library. The exclamation mark (!) indicates that this command is a shell command, not a Python statement, and should be executed in the Colab environment's terminal.

```
!pip install ultralytics
```

- **Ultralytics Library:** Ultralytics is known for developing advanced computer vision models, particularly the YOLO (You Only Look Once) object detection algorithms. Installing this package provides access to the latest implementations and features of these models.
- **Functionality:** The Ultralytics library includes pre-trained models, tools for training new models on custom datasets, and utilities for evaluating and deploying these models. It streamlines the process of setting up and running object detection tasks.
- **Dependencies:** The installation command ensures that all necessary dependencies for Ultralytics are also installed. This includes libraries for handling image processing, numerical computations, and machine learning operations.
- **Use Cases:** Researchers and developers use the Ultralytics library to leverage state-of-the-art object detection capabilities. It is suitable for applications ranging from automated surveillance and image analysis to research in computer vision and deep learning.
- **Integration with Colab:** By installing Ultralytics in Google Colab, users can take advantage of Colab's computational resources, such as GPUs, to efficiently train and deploy YOLO models. This integration supports rapid experimentation and model development.

7.4 Library Import Yolo

The statement **from ultralytics import YOLO** imports the YOLO class specifically from the Ultralytics library. This is essential for utilizing the YOLO object detection models provided by Ultralytics.

```
from ultralytics import YOLO
```

YOLO Class: YOLO (You Only Look Once) is a family of real-time object detection models known for their speed and accuracy. By importing the YOLO class, users can access these models and their functionalities directly within their Python environment.

Model Initialization: Once imported, the YOLO class can be used to initialize pre-trained YOLO models or to set up custom models for training on specific datasets. This provides flexibility in applying the models to various object detection tasks.

Functionality: The YOLO class offers methods for training, evaluating, and predicting objects in images and videos. It encapsulates complex operations into user-friendly functions, making it easier to implement advanced object detection workflows.

Use Cases: This import is particularly useful for tasks requiring real-time object detection, such as automated surveillance, autonomous driving, and various research applications in computer vision.

Integration with Colab: By importing the YOLO class in Google Colab, users can leverage Colab's computational resources, such as GPUs, to efficiently train and deploy YOLO models. This setup supports rapid prototyping and experimentation with high-performance object detection.

- **Check if the Yolo is working**

The command is executed in the terminal or shell environment within Google Colab.

```
!yolo task=detect mode=predict model=yolov8n.pt conf=0.25
      source='https://www.ultralytics.com/images/bus.jpg'
```

- **YOLO Task:** task=detect specifies that the task to be performed is object detection. This indicates that the model will identify and locate objects within the provided image.
- **Mode of Operation:** mode=predict indicates that the model is in prediction mode. This means the model will use a pre-trained YOLO model to predict objects in the given image.
- **Model Specification:** model=yolov8n.pt specifies the pre-trained YOLOv8n model to be used for the prediction. YOLOv8n is a lightweight, efficient version of the YOLOv8 model, designed for faster inference with good accuracy.
- **Confidence Threshold:** conf=0.25 sets the confidence threshold to 25%. This means that the model will only display predictions where it is at least 25% confident that it has correctly identified an object. Lower confidence detections will be ignored.
- **SourceImage:**source='https://www.ultralytics.com/images/bus.jpg' specifies the URL of the image to be processed. The model will download this image and perform object detection on it.
- **Prediction Process:** The command initiates the prediction process, where the specified YOLO model analyzes the image, detects objects, and outputs the results, typically including bounding boxes and class labels for detected objects.
- **Use Cases:** This command is useful for quickly applying a pre-trained YOLO model to a new image, allowing users to see the model's performance and results without needing to train a model from scratch.

Command Execution: The command is executed in the terminal or shell environment within Google Colab.

```
!yolo task=detect mode=train model=yolov8n.pt epochs=50 imgsz=640
      data=../content/drive/MyDrive/DataSets/Playing_Cards/data.yaml
```

YOLO Task: task=detect specifies that the task to be performed is object detection. This indicates that the model will be trained to identify and locate objects within images.

Mode of Operation: mode=train indicates that the model is in training mode. This means that the YOLOv8n model specified by model=yolov8n.pt will be trained on a custom dataset for a specified number of epochs.

Model Specification: model=yolov8n.pt specifies the pre-trained YOLOv8n model to be used as the starting point for training. This model will be fine-tuned on the custom dataset to improve its performance on the specific task.

Training Epochs: epochs=50 specifies the number of training epochs. An epoch refers to one complete pass through the entire training dataset. In this case, the model will be trained for 50 epochs, allowing it to learn from the dataset multiple times.

Image Size: imgsz=640 sets the input image size during training to 640x640 pixels. This parameter determines the resolution at which images are processed by the model during training.

Dataset Specification:

data=../content/drive/MyDrive/DataSets/Playing_Cards/data.yaml specifies the YAML configuration file containing information about the custom dataset. This file defines the paths to the training, validation, and testing data, as well as the number of classes and their corresponding names.

Training Process: The command initiates the training process, during which the YOLOv8n model is fine-tuned on the custom dataset. The model iteratively adjusts its parameters to minimize the difference between predicted and ground truth bounding boxes and class labels.

Output and Progress Monitoring: Throughout the training, the command may display updates on training progress, including metrics such as loss values, average precision, and other performance indicators. These metrics help monitor the model's performance and guide adjustments to training parameters.

Checkpoint Saving: Depending on the configuration, the model may periodically save checkpoints during training. These checkpoints represent snapshots of the model's parameters at different stages of training and can be used to resume training or evaluate the model's performance on a separate dataset.

8. MongoDB

MongoDB is a NoSQL database designed for high performance, high availability, and easy scalability. Unlike traditional relational databases, MongoDB stores data in flexible, JSON-like documents, making it an ideal choice for applications requiring dynamic schemas, large-scale data, and real-time analytics.

8.1 Key Features

1. **Document-Oriented Storage**:
    - Data is stored in BSON (Binary JSON) format.
    - Each record called a document, can contain nested structures, arrays, and various data types, allowing for a rich representation of data.
2. **Schema Flexibility**:
    - MongoDB allows for dynamic schemas, meaning the structure of the documents in a collection does not need to be predefined. Fields can vary from document to document.
3. **Scalability**:
    - MongoDB supports horizontal scaling through sharding, which distributes data across multiple servers. This makes it possible to handle large volumes of data and high-traffic loads by distributing the load across multiple machines.
4. **Indexing**:
    - MongoDB supports a variety of indexing options to improve query performance. Indexes can be created on any field in a document, providing efficient ways to search through the data.
5. **Replication**:
    - To ensure high availability and data redundancy, MongoDB uses replica sets. A replica set is a group of MongoDB instances that maintain the same data set, providing automatic failover and data recovery.
6. **Aggregation Framework**:
    - The aggregation framework allows for complex data processing and transformation operations, such as filtering, grouping, and sorting. It is used to perform data analytics and aggregation tasks efficiently.
7. **Rich Query Language**:
    - MongoDB provides a powerful query language that supports CRUD (Create, Read, Update, Delete) operations. It also includes advanced querying capabilities like geospatial queries and text search.

8.2 Data Model

- **Database**: A container for collections, akin to a schema in relational databases. Each database has its own set of files on the file system.
- **Collection**: A group of documents, similar to a table in a relational database.
- **Document**: A set of key-value pairs, similar to a row in a relational database. Documents can contain nested objects and arrays.

8.3 Key Operations

1. **CRUD Operations**:
    - **Create**: Adding new documents to a collection.
    - **Read**: Retrieving documents based on query criteria.
    - **Update**: Modifying existing documents.
    - **Delete**: Removing documents from a collection.
2. **Indexing**:
    - Creating indexes on fields to improve query performance.
    - Indexes support efficient execution of queries and can be created on single or multiple fields.
3. **Aggregation**:
    - Performing complex data transformations and computations.
    - Operations like filtering, grouping, and sorting are performed using aggregation pipelines.
4. **Replication**:
    - Ensuring high availability by maintaining multiple copies of data across different servers.
    - Replica sets enable automatic failover, ensuring that if one server goes down, another can take its place without data loss.
5. **Sharding**:
    - Distributing data across multiple servers to handle large datasets and high traffic volumes.
    - Sharding partitions the data across shards, which can be spread across different servers, improving both read and write scalability.

8.4 Use Cases

1. **Content Management Systems (CMS)**:
    - Flexible schema design is ideal for managing various types of content without a fixed schema.
2. **Real-Time Analytics**:
    - Fast read and write operations support real-time data analysis and reporting.
3. **Internet of Things (IoT)**:
    - Efficiently handles high-velocity data streams from numerous devices.

4. **Mobile Applications**:
    - Synchronization capabilities and offline data storage support mobile use cases.
5. **E-Commerce Platforms**:
    - Manages complex product catalogues and user data with ease.

8.5 Advantages

- **High Performance**: Fast read and write operations are optimized for performance.
- **High Availability**: Replica sets provide data redundancy and automatic failover.
- **Flexible Schema**: Dynamic schema design allows for easy adaptation to changing data requirements.
- **Scalability**: Horizontal scaling through sharding supports large-scale applications.

8.6 Disadvantages

- **Complex Transactions**: Limited support for multi-document ACID transactions (though improved in later versions).
- **Memory Usage**: Can be high due to the rich document structure.
- **Learning Curve**: Requires understanding NoSQL paradigms and may be different from traditional relational databases.

8.7 Installation and Configuration

1. **Installation**:
    - MongoDB can be installed using package managers or by downloading binaries from the MongoDB website. Installation steps vary by operating system.
2. **Configuration**:
    - Configurations are managed via a configuration file, typically mongod.conf. Key parameters include data directory paths, log file paths and network interfaces.
3. **Starting MongoDB**:
    - MongoDB started as a service, which can be managed using system service commands. The service will manage data storage and retrieval operations.
4. **Connecting to MongoDB**:
    - Connections can be made using the MongoDB shell, a command-line tool, or through MongoDB drivers available for various programming languages, such as Python, Java, and Node.js.

MongoDB offers a robust, flexible, and scalable solution for modern application development. Its document-oriented model, combined with powerful

querying and indexing capabilities, makes it an excellent choice for a wide range of applications, from real-time analytics to content management and IoT systems. By leveraging MongoDB's features, developers can build applications that handle large volumes of data efficiently and adapt to evolving requirements.

9. What is Poker and more about it

Poker is a popular card game that combines skill, strategy, and luck. It involves betting and individual play, with the winner determined by the ranks and combinations of their cards. Poker is played in various formats, but the most common types are Texas Hold'em, Omaha, Seven-Card Stud, and Five-Card Draw.

9.1 Basic Rules of Poker

- Deck: Most poker games use a standard 52-card deck.
- Players: Poker can be played with 2 to 10 players.
- Objective: The goal is to win chips by either having the best hand at the showdown or convincing other players to fold their hands.
- Betting Rounds: There are multiple betting rounds in poker. The specific structure varies by game type but typically includes pre-flop, flop, turn, and river in community card games like Texas Hold'em and Omaha.

9.2 Betting Structures

- No-Limit: Players can bet any amount of their chips.
- Pot-Limit: Players can bet up to the amount currently in the pot.
- Fixed-Limit: Bets are made in fixed increments.

9.3 Hand Rankings

Poker hands are ranked based on the probability of their occurrence, with some hands being more valuable than others. Here are the standard poker hand rankings, from highest to lowest:

1. **Royal Flush:** A, K, Q, J, 10, all of the same suit.
    a. Example: A♠ K♠ Q♠ J♠ 10♠
2. **Straight Flush:** Five consecutive cards of the same suit.
    a. Example: 9♣ 8♣ 7♣ 6♣ 5♣
3. **Four of a Kind:** Four cards of the same rank.
    a. Example: A♠ A♥ A♦ A♣ 2♣
4. **Full House:** Three cards of one rank and two cards of another rank.
    a. Example: K♠ K♥ K♣ 3♦ 3♣
5. **Flush:** Five cards of the same suit, not in sequence.
    a. Example: J♣ 8♣ 5♣ 3♣ 2♣
6. **Straight:** Five consecutive cards of different suits.

    a. Example: 8♠ 7♦ 6♣ 5♦ 4♥
7. **Three of a Kind:** Three cards of the same rank.
    a. Example: Q♠ Q♥ Q♣ 7♦ 4♣
8. **Two Pair:** Two cards of one rank and two cards of another rank.
    a. Example: J♠ J♦ 5♣ 5♦ 2♣
9. **One Pair:** Two cards of the same rank.
    a. Example: 10♠ 10♦ 8♣ 4♦ 2♣
10. **High Card:** The highest card in the hand when no other hand combinations are made.
    a. Example: A♠ 10♦ 9♣ 6♠ 4♦ (Ace high)

9.4 Common Poker Variants

1. **Texas Hold'em:** Each player is dealt two private cards, and five community cards are dealt face up. Players use any combination of their two cards and the five community cards to make the best five-card hand.
2. **Omaha:** Similar to Texas Hold'em, but each player is dealt four private cards, and they must use exactly two of them, along with three of the five community cards, to make the best five-card hand.
3. **Seven-Card Stud:** Each player is dealt seven cards, three face down and four face up. Players use any five of their seven cards to make the best hand.
4. **Five-Card Draw:** Each player is dealt five cards, and they have the opportunity to exchange some or all of their cards for new ones from the deck to make the best hand.

9.5 Poker Strategy

Poker strategy involves understanding the odds, reading opponents, and making decisions based on incomplete information. Key elements include:

- **Position:** Being in a later position (acting after others) is advantageous as you have more information about their actions.
- **Bet Sizing:** Adjusting bet sizes based on the strength of your hand and the tendencies of your opponents.
- **Bluffing:** Representing a stronger hand than you have to induce your opponents to fold better hands.
- **Pot Odds and Equity:** Calculate the odds of completing your hand and compare them to the odds given by the pot size.

Poker is a complex and engaging game that combines elements of chance and skill. Understanding hand rankings, the rules of different variants, and basic strategies are essential for success.

10. Context Flow Diagram

The Poker Hand Recognition System is designed to automate the process of recognizing poker hands from a live video feed. The system leverages computer vision and machine learning techniques to process video frames, identify poker hands, and store and display the results in real time. The following documentation details the components and workflow of the system, as represented in the Context Flow Diagram (CFD).

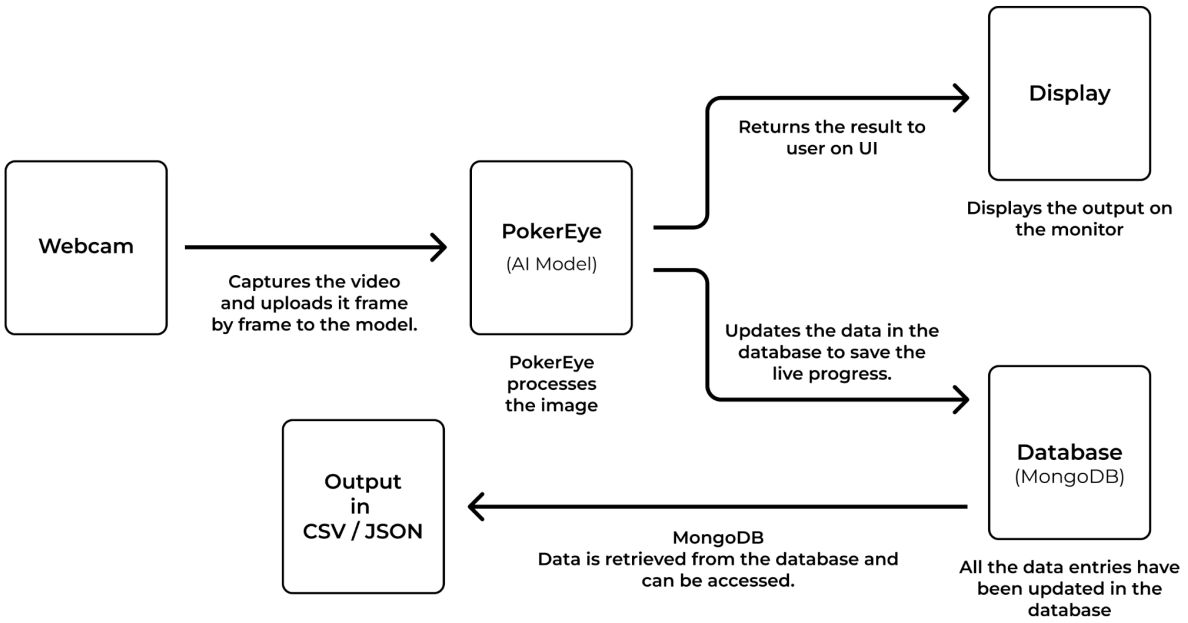


Figure: 10.1 Context Flow Diagram

10.1 Components and Workflow

1. **Webcam**
    - Function: The webcam captures live video of the poker game.
    - Process: It continuously uploads the captured video frame by frame to the AI model for further processing.
2. **PokerEye (AI Model)**
    - Function: PokerEye is the core AI model responsible for processing the video frames.
    - **Process:**
        - It receives video frames from the webcam.
        - Processes each frame to detect and recognize poker hands.
        - The processed data includes identifying the cards and determining the poker hand.
    - **Output:**
        - Updates the database with the recognized hand information.
        - Sends the result to the user interface for immediate display.

3. **Database (MongoDB)**
    - **Function:** The database is used for storing and retrieving the processed data.
    - **Process:**
        - It receives updates from PokerEye and stores the data entries.
        - Enables the retrieval of stored data for future reference and analysis.
    - **Output:**
        - Provides data access for generating output files and for displaying on the monitor.
4. **Display**
    - Function: The display module presents the results to the user in real time.
    - **Process:**
        - Receives the recognized poker hand results from PokerEye.
        - Displays the results on the monitor, providing immediate feedback to the user.
5. **Output in CSV/JSON**
    - **Function:** To provide a formatted output of the processed data.
    - **Process:**
        - Data is retrieved from MongoDB.
        - The data is then formatted into CSV or JSON files, which can be accessed and used for various purposes like reporting, analysis, or further processing.

10.2 Detailed Process Flow

1. **Video Capture and Upload**
    - The system begins with the webcam capturing the live video feed of the poker game. The captured video is uploaded to the AI model frame by frame to ensure continuous processing without lag.
2. **Image Processing and Hand Recognition**
    - Each video frame is processed by PokerEye. This AI model uses advanced computer vision algorithms to analyze the frames, detect the poker cards, and recognize the poker hands. The processing involves multiple steps including image segmentation, card identification, and hand evaluation.
3. **Database Update**
    - Once the poker hand is recognized, the information is sent to the MongoDB database. The database is updated with each recognized hand, ensuring that all data entries are recorded for future use. This step is crucial for maintaining a comprehensive record of the game.

4. **Real-Time Display**
    - Simultaneously, the recognized hand is sent to the display module. The results are shown on the user interface in real time, allowing the players and observers to see the identified poker hands instantly.
5. **Data Retrieval and Output Generation**
    - The stored data in MongoDB can be retrieved to generate output files. These files can be in CSV or JSON format, making the data accessible and usable for various applications. This feature is particularly useful for analysis, reporting, and reviewing the game data.

11. Data Flow Diagram

The Data Flow Diagram (DFD) provides a comprehensive overview of how the Poker Hand Recognition System operates, detailing the interaction between various components and modules. Below is a detailed explanation of each component and the overall workflow:

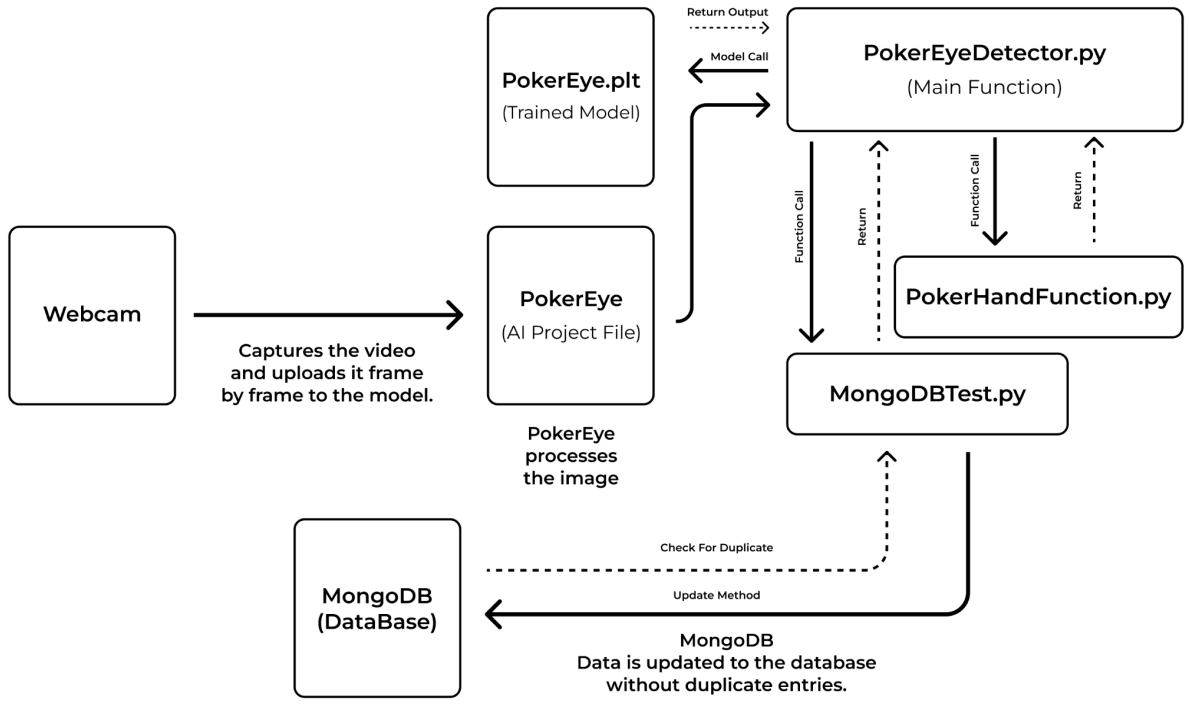


Figure: 11.1 Data Flow Diagram

11.1 Components and Workflow

1. **Webcam**
    - Function: The webcam captures live video of the poker game.
    - Process: It continuously uploads the captured video frame by frame to the AI model for further processing.
2. **PokerEye (AI Project File)**
    - Function: Acts as the central processing unit for the system, handling the primary image processing tasks.
    - Process:
        - Receives video frames from the webcam.
        - Utilizes the trained model (PokerEye.plt) to process each frame, identifying the poker cards and hands.
        - Calls various functions and modules to complete specific tasks in the recognition process.
3. **PokerEye.plt (Trained Model)**
    - Function: The trained machine learning model is used to detect and recognize poker hands from video frames.
    - Process:
        - Called by PokerEye (AI Project File) for model inference.
        - Processes the input frames and returns the recognized poker hands.
4. **PokerEyeDetector.py (Main Function)**
    - Function: Serves as the main control script that orchestrates the workflow.
    - Process:
        - Calls the trained model (PokerEye.plt) to perform hand recognition.
        - Interacts with other scripts like PokerHandFunction.py and MongoDBTest.py to handle various tasks.
        - Returns the output results after processing.
5. **PokerHandFunction.py**
    - Function: Contains specific functions related to poker hand recognition.
    - Process:
        - Called by PokerEyeDetector.py to perform detailed hand analysis.
        - Returns results of hand recognition to the main function.
6. **MongoDBTest.py**
    - Function: Handles database interactions, particularly with MongoDB.
    - Process:
        - Called by PokerEyeDetector.py to check for duplicates and update the database.
        - Ensures that data is stored without duplicates, maintaining data integrity.

        - Returns confirmation of database updates.
7. **MongoDB (Database)**
    - Function: Stores the processed data from the AI model.
    - Process:
        - Receives data updates from MongoDBTest.py.
        - Stores recognized hand data and ensure no duplicate entries are recorded.
        - Data can be retrieved for further use or analysis.

11.2 Detailed Process Flow

1. **Video Capture and Upload**
    - The process starts with the webcam capturing the live video feed. This video is uploaded frame by frame to the PokerEye (AI Project File) for continuous processing.
2. **Image Processing and Hand Recognition**
    - PokerEye (AI Project File) receives each frame and processes it using the trained model (PokerEye.plt). This model detects and recognizes the poker hands within each frame.
    - The recognition results are then handled by PokerEyeDetector.py, which manages the overall workflow and integrates various function calls.
3. **Function Calls and Integration**
    - PokerEyeDetector.py coordinates the recognition process by calling:
        - PokerEye.plt for model inference.
        - PokerHandFunction.py for detailed hand analysis.
        - MongoDBTest.py for database operations, ensuring data integrity.
4. **Database Operations**
    - MongoDBTest.py interacts with MongoDB to check for duplicate entries and update the database with new recognized hand data.
    - MongoDB ensures all data entries are unique and stores them for future retrieval.
5. **Return Output**
    - Once the database is updated and hand recognition is completed, PokerEyeDetector.py returns the output results.
    - These results can be displayed on a user interface or used for further analysis and reporting.

The Poker Hand Recognition System is a sophisticated integration of various components, each playing a crucial role in the overall functionality. The webcam captures the live game, the AI project file and trained model handle the recognition tasks, and the database ensures data integrity and storage. The well-coordinated function calls and integration ensure a seamless operation, making the system effective in real-time poker hand recognition. This detailed

workflow ensures accuracy, efficiency, and reliability in recognizing and processing poker hands from live video feeds.

11.3 Detailed Analysis of the Code and Algorithm

11.3.1 PokerEyeDetector.py (Main Function)

```
from ultralytics import YOLO
import cv2
import cvzone
import math
import PokerHandFunction
import MongoDBTest
from datetime import datetime

cap = cv2.VideoCapture(0)
cap.set(propId=3, value=1280)
cap.set(propId=4, value=720)

model = YOLO("../Yolo-Weights/playingCards.pt")



className = [
    '10C', '10D', '10H', '10S',
    '2C', '2D', '2H', '2S',
    '3C', '3D', '3H', '3S',
    '4C', '4D', '4H', '4S',
    '5C', '5D', '5H', '5S',
    '6C', '6D', '6H', '6S',
    '7C', '7D', '7H', '7S',
    '8C', '8D', '8H', '8S',
    '9C', '9D', '9H', '9S',
    'AC', 'AD', 'AH', 'AS',
    'JC', 'JD', 'JH', 'JS',
    'KC', 'KD', 'KH', 'KS',
    'QC', 'QD', 'QH', 'QS'
]



pokerHandRanks = {
    10: "Royal Flush", 9: "Straight Flush", 8: "Four of a Kind", 7: "Full House",
    6: "Flush", 5: "Straight", 4: "Three of a Kind", 3: "Two Pair",
    2: "Pair", 1: "High"
}
reversedPokerHandRanks = {value: key for key, value in pokerHandRanks.items()}

while True:
    success, img = cap.read()
    resluts = model(img, stream=True)
    hand = []

    cvzone.putTextRect(img, 'Poker Eye', (25, 60), scale=1.3, thickness=2,
                       colorT=(31, 18, 71), colorR=(201, 233, 107), font=cv2.FONT_HERSHEY_PLAIN)
    cvzone.putTextRect(img, "Product of Aydie's Avenue", (22, 95), scale=0.8, thickness=1,
                       colorT=(31, 18, 71), colorR=(201, 233, 107), font=cv2.FONT_HERSHEY_PLAIN, offset=7)
    cvzone.putTextRect(img, 'visit: aydie.in/projects', (600, 700), scale=0.8, thickness=1,
                       colorT=(31, 18, 71), colorR=(201, 233, 107), font=cv2.FONT_HERSHEY_PLAIN, offset=6)



    for r in resluts:
        boxes = r.boxes
        for box in boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            w, h = x2 - x1, y2 - y1
            cvzone.cornerRect(img, bbox=(x1, y1, w, h))

            conf = math.ceil(box.conf[0] * 100) / 100
            cls = int(box.cls[0])

            cvzone.putTextRect(img, f'{className[cls]} {conf}', (max(0, x1), max(35, y1)), scale=1.2,
thickness=2, colorT=(31, 18, 71), colorR=(201, 233, 117), font=cv2.FONT_HERSHEY_PLAIN)
```

```
            if conf > 0.8:
                hand.append(classNames[cls])

    hand = list(set(hand))
    print("Card Detected ⇒ ", hand)

    if len(hand) == 5:
        resluts = PokerHandFunction.findPokerHand(hand)
        print("Combination Detected ⇒ ", resluts)


        time_ = datetime.now().strftime('%H:%M')
        dbresult_ = MongoDBTest.mongoDataBaseUpdater(
            str(resluts), str(hand), reversedPokerHandRanks[str(resluts)], time_
        )
        print("MongoDB Feedback ⇒ ", dbresult_, "\n\n")

        cvzone.putTextRect(img, f'Your Hand is {resluts}, Rank score is
{reversedPokerHandRanks[str(resluts)]}', (475, 60), scale=1.5, thickness=2, colorT=(31, 18, 71),
colorR=(201, 233, 107), font=cv2.FONT_HERSHEY_PLAIN)

        cvzone.putTextRect(img, f'Combination → {str(hand)}', (475, 100), scale=1, thickness=2,
                           colorT=(31, 18, 71), colorR=(201, 233, 107), font=cv2.FONT_HERSHEY_PLAIN)

    cv2.imshow("Image", img)
    cv2.waitKey(1)
```

11.3.1.1 Code Summary

1. Import Libraries:
    - The code starts by importing several libraries necessary for its functionality, including YOLO for object detection, OpenCV for image processing, and others for handling poker hands and database operations.
2. Setup Camera:
    - The camera is set up using OpenCV to capture video. The resolution is set to 1280x720 pixels.
3. Load YOLO Model:
    - The YOLO model, which is pre-trained to detect playing cards, is loaded.
4. Class Names:
    - A list of class names is created, representing all the playing cards (e.g., '10C' for 10 of Clubs, 'AH' for Ace of Hearts).
5. Poker Hand Ranks:
    - Two dictionaries are defined to map poker hand ranks to their names (e.g., "Royal Flush") and vice versa.
6. Main Loop:
    - The main loop continuously captures frames from the camera and processes them.
7. Process Frame:
    - Each captured frame is processed by the YOLO model to detect playing cards.
    - Detected cards are stored in a list called hand.
8. Display Branding:
    - Some text is displayed on the frame for branding purposes (e.g., "Poker Eye", "Product of Aydie's Avenue").

9. Draw Bounding Boxes:
    - For each detected card, a bounding box is drawn around it on the frame.
    - The confidence level and class name of the detected card are also displayed.
10. Check for Poker Hand:
    - If exactly five cards are detected, the poker hand is determined using a function from the PokerHandFunction module.
    - The result is printed and displayed on the frame.
11. Update Database:
    - The detected poker hand and its details are updated in a MongoDB database using the MongoDBTestmodule.
12. Display Results:
    - The result of the poker hand detection and its rank are displayed on the frame.
13. Show Frame:
    - The processed frame is displayed in a window.
    - The loop continues, capturing and processing new frames until the program is stopped.

In summary, this code captures video from a webcam, detects playing cards in real-time, identifies poker hands, updates a database with the results, and displays the processed video with annotations on the screen.

11.3.2 PokerHandFunction.py

```
def findPokerHand(hand):

    ranks = []
    suits = []
    possibleRanks = []

    for card in hand:
        if len(card) == 2:
            rank = card[0]
            suit = card[1]
        else:
            rank = card[0:2]
            suit = card[2]


        if rank == "A":
            rank = 14

        if rank == "K":
            rank = 13

        if rank == "Q":
            rank = 12

        if rank == "J":
            rank = 11

        ranks.append(int(rank))
        suits.append(suit)

    sortedRanks = sorted(ranks)
```

```
    # Royal Flush && Straight Flush && Flush
    if suits.count(suits[0]) == 5:
        # Royal FLush
        if 14 in sortedRanks and 13 in sortedRanks and 12 in sortedRanks and 11 in sortedRanks and 10 in
sortedRanks:
            possibleRanks.append(10)

        # Straight Flush
        elif all(sortedRanks[i] == sortedRanks[i - 1] + 1 for i in range(1, len(sortedRanks))):
            possibleRanks.append(9)

        # Flush
        else:
            possibleRanks.append(6)


    # Straight
    # 10 11 12 13 14
    # 11 == 10+1 True
    elif all(sortedRanks[i] == sortedRanks[i - 1] + 1 for i in range(1, len(sortedRanks))):
        possibleRanks.append(5)

    handUniqueVals = list(set(sortedRanks))


    #Four of a kind and Full House
    #3 3 3 3 5 --> set --> 3 5 --> Four of a kind --> if 3 == 4
    #3 3 3 5 5 --> set --> 3 5 --> Full house --> if 3 == 3
    if len(handUniqueVals) == 2:
        for val in handUniqueVals:
            if sortedRanks.count(val) == 4: #Four of a kind
                possibleRanks.append(8)

            elif sortedRanks.count(val) == 3: #Full house
                possibleRanks.append(7)


    #Three of a kind and two pairs
    # 5 5 5 6 7 --> set --> 5 6 7 --> Three of a kind
    # 5 5 6 6 7 --> set --> 5 6 7 --> Two pair
    if len(handUniqueVals) == 3:
        for val in handUniqueVals:
            if sortedRanks.count(val) == 3: #three of a kind
                possibleRanks.append(4)
            elif sortedRanks.count(val) == 2: #two pair
                possibleRanks.append(3)


    #Pair
    #4 4 5 6 7 --> set --> 4 5 6 7 --> 4 Unique --> Pair
    if len(handUniqueVals) == 4:
        possibleRanks.append(2)

    if not possibleRanks:
        possibleRanks.append(1)

    #print(max(possibleRanks))
    pokerHandRanks = {10:"Royal Flush", 9:"Straight Flush", 8:"Four of a Kind", 7:"Full House", 6:"Flush",
                      5:"Straight", 4:"Three of a Kind", 3:"Two Pair", 2:"Pair", 1:"High"}

    output = pokerHandRanks[max(possibleRanks)]

    print(hand,output,"\n")

    return output
```

```
#Demo Data
if __name__ == '__main__':
    findPokerHand(["AH", "KH", "QH", "JH", "10H"]) # Royal Flush
    findPokerHand(["QC", "JC", "10C", "9C", "8C"])  # Straight Flush
    findPokerHand(["5H", "5S", "5D", "5C", "QH"]) # Four of a Kind
    findPokerHand(["2H", "2S", "2D", "10C", "10H"])  # Full House
    findPokerHand(["KH", "7H", "6H", "10H", "2H"])  # Flush
    findPokerHand(["10H", "9C", "8D", "7S", "6H"])  # Straight
    findPokerHand(["7H", "7D", "7S", "QS", "3H"])  # Three of a kind
    findPokerHand(["JH", "JS", "5D", "5S", "7H"])  # Two Pair
    findPokerHand(["AH", "AC", "KD", "JH", "7S"])  # Pair
    findPokerHand(["KH", "8S", "3D", "10C", "2H"])  # High Card
```

11.3.2.1 Code Summary

1. Initialization:
    - Three empty lists ranks, suits, and possibleRanks are created to store the ranks, suits, and possible ranks of the poker hand, respectively.

2. Processing Cards:
    - The function loops through each card in the hand.
    - It determines the rank and suit of each card and converts them into numerical values if necessary.
    - The ranks and suits of the cards are stored in the respective lists.
3. Sorting Ranks:
    - The ranks are sorted in ascending order.
4. Check for Flushes:
    - If all the cards have the same suit, it checks for a Royal Flush, Straight Flush, or simply a Flush.
5. Check for Straights:
    - If the ranks form a sequence, it checks for a Straight.
6. Check for Four of a Kind and Full House:
    - If there are only two unique ranks, it checks for Four of a Kind or a Full House.
7. Check for Three of a Kind and Two Pairs:
    - If there are three unique ranks, it checks for Three of a Kind or Two Pairs.
8. Check for a Pair:
    - If there are four unique ranks, it checks for a Pair.
9. Determining the Poker Hand:
    - If no specific hand is detected, it defaults to a "High Card".
10. Output:
    - The detected poker hand is printed and returned as output.
11. Demo Data:
    - The function is called with demo data representing various poker hand combinations, and the results are printed.

This function essentially analyzes the input hand of playing cards and determines the best poker hand combination that can be formed from them.

11.3.3 MongoDBTest.py

```
from pymongo import MongoClient, errors
from datetime import datetime

client = MongoClient('mongodb://localhost:27017/')
db = client['PokerEyeTest']
collection = db['PokerEyeTestData']

collection.create_index([("Hand", 1), ("Combinations", 1)], unique=True)

def mongoDataBaseUpdater(Hand, Combinations, Score, Time):

    latest = collection.find_one(sort=[('_id', -1)])

    if latest is None:
        latest_id = 1
    else:
        latest_id = latest['_id'] + 1



    data = {
        '_id': latest_id,
        'Hand': Hand,
        'Combinations': Combinations,
        'Score': Score,
        'Time': Time
    }

    try:
        collection.insert_one(data)
        return f"New Inserted record......"
    except errors.DuplicateKeyError:
        return f"Duplicate Record Found......"
        result = collection.update_one({'Hand': Hand, 'Combinations': Combinations},
                                       {'$set': {'Score': Score, 'Time': Time}})
        if result.matched_count:
            return f"Old Record Updated......\n"
        else:
            return f"Update Failed\n\n"
    except Exception as e:
        return f"An error occurred: {e}"
```

11.3.3.1 Code Summary

1. Import Libraries:
    - The pymongo library is imported to interact with MongoDB, and datetime is imported to handle timestamps.
2. Initialize MongoDB Client and Collection:
    - A MongoDB client is initialized to connect to the local MongoDB server (localhost:27017).
    - A database named 'PokerEyeTest' and a collection named 'PokerEyeTestData' are created.
    - A unique index is created on the fields 'Hand' and 'Combinations'. This ensures that each document in the collection is unique based on these fields.

3. Define mongoDataBaseUpdater Function:
    - This function takes four parameters: Hand, Combinations, Score, and Time.
    - It retrieves the latest document from the collection to determine the _id for the new document.
    - It creates a new document with the provided data.
    - It attempts to insert the new document into the collection.
        - If a duplicate key error occurs, it means a document with the same 'Hand' and 'Combinations‘ exists. In this case, it updates the existing document with the new Score and Time.
        - If any other error occurs during insertion, it catches the exception and returns an error message.
    - It returns a success or error message indicating the outcome of the operation.

4. Demo Data:
    - This section demonstrates how to use the mongoDataBaseUpdater function by calling it with sample data representing various poker hand combinations.
5. Close MongoDB Client:
    - The MongoDB client is not closed in this script. If you want to close the client connection after use, you can uncomment the client.close() line.

Overall, this script sets up a MongoDB database and provides a function to update it with poker hand data, ensuring uniqueness based on the hand and combination.

11.4 Running the Model

11.4.1 Step 1: Run the MongoDB Server using Terminal

```
% sudo mongod --dbpath=/Volumes/AydieT7/PokerEye_V24.2.0/Database/pkdb1
```

This command is executed in a Unix-like environment, such as Linux or macOS, and it starts the MongoDB server (**mongod**). Let's break down the command:

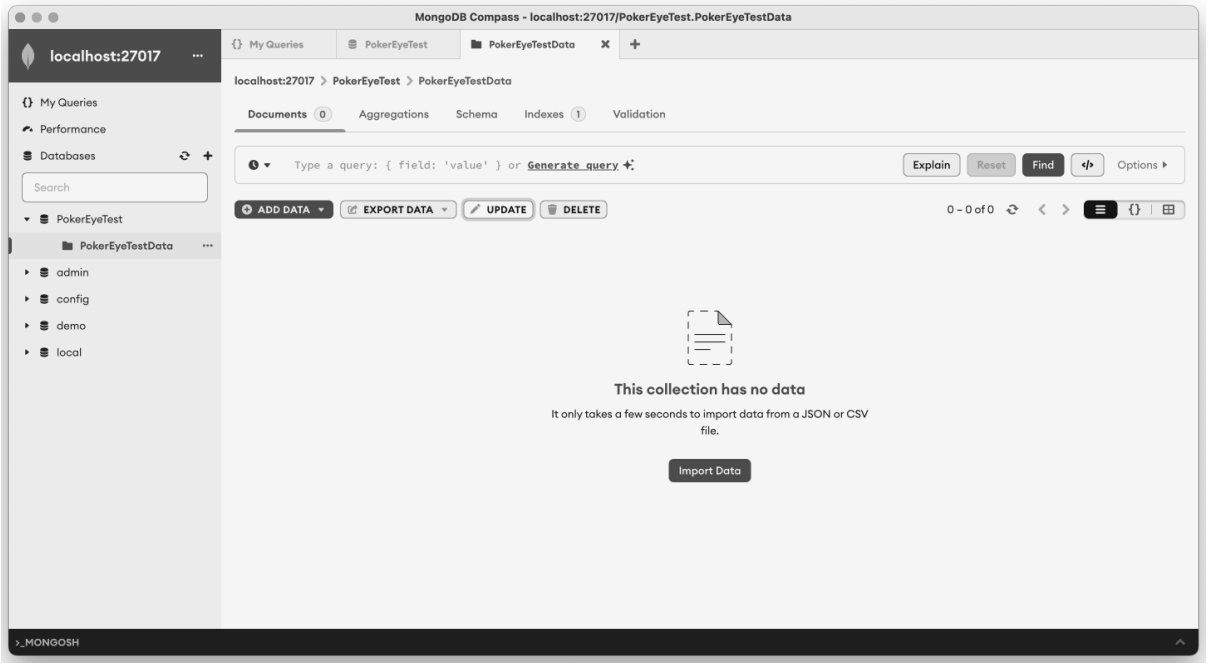
- % sudo: The % sign at the beginning might indicate that this command is executed in a Jupyter Notebook or IPython environment. sudo is a

command used in Unix-like operating systems to run programs with elevated privileges, typically as the superuser or root.

- **mongod**: This is the MongoDB server daemon. It's the primary process for the MongoDB database system. Running this command starts the MongoDB server.
- **--dbpath=/Volumes/AydieT7/PokerEye_V24.2.0/Database/pkdb1**: This option specifies the path to the MongoDB database files. In this case, it's set to **/Volumes/AydieT7/PokerEye_V24.2.0/Database/pkdb1.** This directory contains the database files for the MongoDB server. It's where MongoDB will store its data.

To sum up, this command starts the MongoDB server (**mongod**) with elevated privileges (**sudo**) and specifies the location of the database files using the --dbpath option.

11.4.2 Step 2: Open MongoDB Compass



11.4.3 Step 3: Run The main function using PyCharm IDE

11.4.4 Step 4: Update the database by showing the card combination to the camera

MogogDB Data Base Table

_id	Hand	Combinations	Score	Time
1	High Card	[‘KH’,‘8C’,‘QD’,‘2S’,‘7H’]	1	15:35
2	Pair	[‘AH’,‘AC’,’KD’,’JS’,’7H’]	2	15:40
3	Two Pair	[‘JH’,’JC’,’5D’,’5S’,’7H’]	3	15:42
4	Three of a Kind	[‘7H’,’7D’,’7C’,’QS’,’3H’]	4	15:55
5	Straight	[‘10’,’9C’,’8D’,’7S’,’6H’]	5	15:57
6	Flush	[‘KC’,’10C’,’9C’,’6C’,’5C’]	6	16:01
7	Full House	[‘AH’,’AC’,’AD’,’3S’,’3D’]	7	16:05
8	Four of a Kind	[‘2S’,’2D’,’2H’,’2C’,’AH’]	8	16:07
9	Straight Flush	[‘JC’,’10C’,’9C’,’8C’,’7C’]	9	16:11
10	Royal Flush	[‘AH’,’KH’,’QH’,’JH’,’10H’]	10	16:15

12. Scope of the Solution

Scope of the Solution The project's primary objective lies in the meticulous recognition of standard five-card poker hand combinations, a task requiring intricate algorithmic processing and pattern recognition. Moreover, it encompasses a spectrum of functionalities, delving into the granular details of card identification, including the precise recognition of individual card ranks and meticulous suit discernment. This comprehensive approach ensures a thorough understanding and interpretation of each hand's composition, facilitating accurate assessment and strategic decision-making in poker gameplay.

Furthermore, it is imperative to underscore the ethical framework within which this system operates. Designed with a dual purpose, it serves the realms of personal exploration and educational enlightenment. As a tool crafted for personal use, it empowers individuals to delve into the nuances of poker hand analysis, enhancing their understanding and proficiency in the game. Simultaneously, it functions as an educational resource, fostering learning and skill development in the domain of poker hand recognition.

However, it is paramount to emphasize the ethical imperative ingrained within the system's design. It unequivocally prohibits exploitation for undue advantage in online poker settings, echoing the principles of fair play and integrity upheld within the gaming community. Any attempt to deploy this system for such purposes contravenes not only ethical standards but also gaming regulations, underscoring the importance of responsible utilization and adherence to established guidelines.

In essence, the scope of this solution transcends mere technical prowess; it embodies a commitment to ethical integrity and educational empowerment, serving as a beacon for responsible innovation in the realm of gaming technology.

13. Work Plan and Timeline

The meticulous planning and execution of the project's development are paramount to its success. Therefore, the work plan and timeline are structured into discrete phases, each meticulously crafted to ensure efficiency and efficacy.

- **Phase 1: Data Collection and Preparation (1-2 weeks):** This foundational phase sets the stage for subsequent activities by acquiring and preparing the requisite data. It involves a multifaceted approach:

  Gathering poker card images from diverse sources: Scouring various repositories, databases, and online platforms to amass a comprehensive collection of high-quality poker card images.

  Labeling images with their corresponding hand combinations: Engaging in manual or automated labelling processes to annotate each image with the specific poker hand it represents, facilitating supervised learning during model training.

  Pre-processing and organizing the dataset for training: Employing data preprocessing techniques such as normalization, resizing, and augmentation to enhance the quality and diversity of the dataset. Additionally, organizing the dataset into appropriate training, validation, and testing subsets to ensure robust model training and evaluation.

- **Phase 2: Model Training (2-3 Days):**

  With the dataset meticulously curated and prepared, the focus shifts to the development and refinement of the AI model architecture:

  Defining and building the AI model architecture using Yolo and OpenCV: Leveraging state-of-the-art deep learning frameworks and computer vision libraries to construct a robust and versatile model capable of accurately recognizing poker hand combinations.

  Training the model on the prepared dataset using Google Colab for faster processing: Harnessing the computational resources and collaborative capabilities of Google Colab to expedite the model training process, thereby accelerating iterations and optimizations.

  Monitoring model performance and adjusting hyperparameters (learning rate, optimizer) for optimal results: Iteratively fine-tuning model hyperparameters and monitoring performance metrics to enhance accuracy, convergence speed, and generalization capabilities.

  Phase 3: System Integration and Testing (1-2 Days):

  With the trained model in hand, the focus shifts to integrating it into a cohesive system and rigorously testing its functionality:

  Integrating the trained model with the video capture and user interface modules: Seamlessly integrating the model into a unified software architecture, ensuring compatibility and interoperability with the video capture hardware and user interface components.

  Testing the system functionality with various hand combinations and camera setups: Conducting comprehensive testing scenarios to validate the system's ability to accurately recognize diverse poker hand combinations under varying environmental conditions, camera angles, and lighting conditions.

  Debugging and refining the system to ensure accuracy and performance: Iteratively identifying and resolving software bugs, performance bottlenecks, and edge cases to enhance the system's reliability, robustness, and user experience.

- **Phase 4: User Interface Development (2 Days):**

  The final phase encompasses the design and implementation of a user-friendly interface to facilitate intuitive interaction with the system:

  Designing a user-friendly interface for displaying the identified hand combination in real-time: Collaborating with UX/UI designers to craft an

aesthetically pleasing and intuitive user interface that conveys relevant information effectively and efficiently.

Implementing the interface using chosen libraries or frameworks: Leveraging frontend development technologies such as HTML, CSS, JavaScript, and relevant UI frameworks to translate the design mockups into functional user interfaces seamlessly.

Conducting user testing to gather feedback and improve the user experience: Soliciting feedback from end-users through usability testing sessions, surveys, and feedback forms to identify pain points, preferences, and opportunities for refinement, iteration, and enhancement.

In summary, the work plan and timeline delineate a structured and iterative approach to project development, encompassing data collection, model training, system integration, and user interface design. By adhering to this systematic methodology, the project endeavors to achieve its objectives effectively while ensuring the delivery of a robust, reliable, and user-friendly poker hand recognition system.

14. Expected Outcomes

The project aims to deliver a comprehensive and innovative AI-driven solution for recognizing poker hand combinations. The expected outcomes encompass the following key aspects:

1. **14.1 A Functional and User-Friendly AI System:**
    - **Real-Time Poker Hand Recognition:** The core functionality of the system is its ability to accurately recognize standard poker hand combinations from a live video stream. This involves the seamless integration of advanced computer vision techniques and deep learning models, ensuring the system can identify and classify hand combinations in real-time with high precision.
    - **High Accuracy:** Leveraging state-of-the-art models and extensive training data, the system is expected to achieve high accuracy rates in recognizing various poker hands. This entails correctly identifying hands such as pairs, flushes, straights, and full houses, even in diverse and challenging environmental conditions.
    - **User-Friendly Interface:** The system is designed with a focus on user experience. A clean, intuitive interface will display the identified poker hand combinations in real-time, providing users with immediate and clear feedback. The interface will be accessible and straightforward, ensuring ease of use for individuals with varying levels of technical expertise.

2. **14.2 Efficiency in Hand Identification:**
    - **Reduced Time and Effort:** By automating the process of hand identification, the system significantly reduces the time and effort required to determine poker hand combinations. Players no longer need to manually assess their hands, allowing for a more streamlined and efficient gameplay experience.
    - **Focus on Strategy:** With the burden of hand identification lifted, players can redirect their focus towards strategic decision-making. This enhanced focus can lead to improved gameplay, as players can concentrate on their overall strategy, opponent analysis, and in-game tactics without being distracted by the need to verify their hands manually.
3. **14.3 Educational Value:**
    - **Learning Tool:** The system serves as a valuable educational resource for individuals looking to learn and practice poker hand recognition techniques. By providing instant feedback on hand combinations, users can gain a deeper understanding of poker hands, improving their recognition skills over time.
    - **Interactive Practice:** Users can engage in interactive practice sessions, using the system to test their ability to identify poker hands in real-time scenarios. This hands-on learning approach can accelerate skill development and enhance users' confidence in their poker hand recognition abilities.
    - **Enhanced Comprehension:** The system's detailed feedback can help users understand the nuances of poker hand rankings and probabilities. Educational modules or tutorials integrated within the system can further elucidate concepts such as hand strength, odds calculation, and strategic play based on hand recognition.
4. **14.4 Additional Anticipated Benefits:**
    - **Accessibility:** The system's design ensures it is accessible to a broad audience, including novice players, enthusiasts, and educators. The intuitive interface and comprehensive educational resources make it an invaluable tool for anyone interested in mastering poker hand recognition.
    - **Versatility:** While primarily focused on standard five-card poker hand combinations, the system's underlying architecture can be adapted for other poker variations, such as Texas Hold'em or Omaha, broadening its applicability and utility.
    - **Community Engagement:** By fostering a deeper understanding and appreciation of poker through advanced technology, the system can engage and inspire the poker community. It can serve as a catalyst for discussions, workshops, and collaborative learning opportunities centered around poker strategy and AI applications in gaming.

In conclusion, the expected outcomes of this project extend beyond the creation of a highly accurate and efficient AI system for poker hand recognition. It aims to transform the way players interact with and learn about poker, providing a robust, user-friendly, and educational tool that enhances both gameplay and strategic understanding.

To ensure the effectiveness and reliability of the developed poker hand recognition system, a rigorous and comprehensive evaluation and testing process will be implemented. This multifaceted approach is designed to assess the system's performance, usability, and robustness across a variety of conditions and user interactions.

15. Accuracy Testing

1. **15.1 Testing Dataset Evaluation:**
    - **Objective Assessment:** The model's performance will be evaluated using a separate testing dataset that was not included in the training process. This ensures an unbiased and objective assessment of the model's ability to generalize to unseen data.
    - **Metrics Utilized:** Key performance metrics such as accuracy, precision, recall, and F1-score will be used to evaluate the model's proficiency in correctly identifying different poker hand combinations.
        - **Accuracy:** Measures the proportion of correctly identified hands out of the total hands.
        - **Precision:** Evaluates the ratio of correctly identified positive observations (e.g., a specific hand type) to the total identified positive observations.
        - **Recall:** Assesses the ratio of correctly identified positive observations to all actual positive observations in the dataset.
        - **F1-Score:** Provides a balance between precision and recall, offering a single metric that considers both false positives and false negatives.
    - **Confusion Matrix:** A confusion matrix will be generated to provide detailed insights into the model's performance across different hand combinations, highlighting areas where misclassifications are more likely to occur.

2. **15.2 User Interaction and Feedback:**
    - **Diverse User Group:** The system will be tested with a group of potential users, including novice players, experienced poker enthusiasts, and educators. This diverse group will provide comprehensive feedback on the system's usability and effectiveness.

- **Usability Testing:** Participants will use the system in realistic scenarios to identify any challenges or obstacles they face when interacting with the system.
    - **Task Completion Time:** Measuring the time users take to complete specific tasks, such as identifying a poker hand.
    - **Ease of Use:** Collecting subjective feedback on the ease of use and intuitiveness of the interface.
    - **User Satisfaction:** Gathering overall satisfaction scores and comments regarding the user experience.
- **Feedback Collection:** Detailed surveys, interviews, and observation sessions will be conducted to gather qualitative and quantitative feedback.
    - **User Challenges:** Identifying any difficulties or pain points users encounter.
    - **Suggestions for Improvement:** Soliciting user suggestions for enhancing the user interface and overall experience.
- **Iterative Refinement:** Based on user feedback, iterative improvements will be made to the system to enhance usability and user satisfaction.

3. **15.3 Stress Testing (Robustness and Reliability Assessment):**
    - **Simulated Environmental Conditions:** The system will be subjected to a variety of simulated environmental conditions to evaluate its robustness and reliability.
        - **Different Lighting Scenarios:** Testing the system's performance under varying lighting conditions, including low light, bright light, and fluctuating lighting environments.
        - **Varied Camera Angles:** Assessing the system's accuracy when the camera is positioned at different angles relative to the poker table and cards.
        - **Card Positioning Variations:** Evaluating the system's ability to correctly identify hand combinations when cards are placed in unconventional or skewed positions.
    - **Stress Testing Scenarios:**
        - **Rapid Movements:** Introducing rapid movements of cards or players' hands to test the system's responsiveness and accuracy in dynamic environments.
        - **Background Noise:** Adding visual noise or clutter in the background to assess the system's capability to maintain accuracy in less controlled settings.
    - **Performance Monitoring:** Continuously monitoring the system's performance under these conditions to identify any degradation in accuracy or responsiveness.
        - **Error Rates:** Tracking the frequency and nature of errors under different stress conditions.

- **System Stability:** Ensuring the system remains stable and does not crash or produce significant delays during testing.

In conclusion, the evaluation and testing process is designed to thoroughly assess the poker hand recognition system's accuracy, usability, and robustness. By employing a combination of accuracy testing, user testing, and stress testing, the project aims to deliver a reliable, user-friendly, and high-performance system that meets the needs of its diverse user base and performs consistently in real-world conditions.

16. Limitations

While this project provides a significant advancement in poker hand recognition technology, several limitations must be acknowledged. Understanding these constraints is essential for setting realistic expectations and guiding future development efforts.

16.1 Limited Hand Recognition

1. **Focus on Standard Combinations:**
   - The current system is specifically designed to recognize standard seven-card poker hand combinations. While this encompasses the majority of commonly encountered hands, it does not account for the unique variations and complexities found in other popular forms of poker.
2. **Exclusion of Other Poker Variants:**
   - **Texas Hold'em:** This widely played variation involves a different card distribution and community cards that the system is not currently equipped to handle. Recognizing hands in Texas Hold'em requires the system to interpret combinations from two personal cards and five community cards.
   - **Omaha:** In Omaha, players receive four personal cards and use two of them in conjunction with three of the five community cards to form a hand. This additional complexity is not addressed by the current model.
3. **Future Expansion Potential:**
   - Future iterations of the system could be expanded to include these and other poker variations. This would involve additional training on specific datasets representing the unique rules and hand combinations of each poker variant.

16.2 Individual Card Recognition

1. **Current Scope Limitations:**
   - The project does not currently include functionalities for recognizing individual card ranks (e.g., Ace, King, Queen) or suits (e.g., hearts,

clubs). The focus has been on recognizing hand combinations rather than individual cards.

2. **Implications for Analysis:**
    - A more granular level of recognition, such as identifying each card's rank and suit, could enhance the system's utility, providing detailed analysis and broader applicability. For instance, it could be used for more comprehensive game analysis, including strategies based on specific card distributions.
3. **Future Development:**
    - Future development could incorporate these functionalities, allowing the system to perform detailed card recognition and thereby support a wider range of analytical and educational purposes.

16.3 Ethical Considerations

1. **Intended Use:**
    - As emphasized throughout the project, the system is intended strictly for personal use and educational purposes. It is designed to aid in learning and practicing poker hand recognition, rather than to be used in competitive or real-money gaming environments.
2. **Potential for Misuse:**
    - There is an inherent risk that such a system could be misused to gain unfair advantages in online poker games, which is explicitly prohibited by most gaming platforms and can result in significant penalties for users.
3. **Responsible Development:**
    - Future work should prioritize ethical considerations, ensuring that the system adheres to all relevant regulations and guidelines. This includes implementing safeguards to prevent misuse and actively discouraging any attempts to use the system in ways that violate online poker regulations.

16.4 Additional Limitations

1. **Dependence on Hardware:**
    - The accuracy and performance of the system may be influenced by the quality of the video capture hardware and the environmental conditions during use. Variations in camera resolution, lighting, and card visibility can affect the system's ability to accurately recognize hand combinations.
2. **Real-World Variability:**
    - The system's effectiveness can be impacted by real-world variability such as inconsistent card layouts, player movements, and background distractions. These factors can introduce challenges

that were not fully accounted for during the development and testing phases.

3. **Computational Requirements:**
    - High-performance model training and real-time recognition may require significant computational resources, potentially limiting the system's accessibility for users with less powerful hardware.

In summary, while the poker hand recognition system represents a significant technological advancement, it is important to acknowledge its current limitations. These include a focus on standard hand combinations, the absence of individual card recognition, and critical ethical considerations. Recognizing and addressing these limitations will be key to refining and expanding the system's capabilities in future development efforts.

17. Future Work

Building upon the foundation established in this project, future work can explore several avenues to enhance the system's capabilities, broaden its applications, and ensure it remains at the cutting edge of poker hand recognition technology.

17.1 Enhanced Model Training

1. **Exploring Advanced Architectures:**
    - Deep Learning Models: Experimenting with advanced deep learning architectures such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based models can significantly improve the system's ability to accurately recognize and differentiate poker hand combinations.
    - Transfer Learning: Leveraging pre-trained models from related domains and fine-tuning them on poker-specific data can enhance performance by utilizing existing knowledge.
2. **Optimized Training Techniques:**
    - Data Augmentation: Implementing sophisticated data augmentation techniques to create a more diverse and comprehensive training dataset can help the model generalize better to varied real-world scenarios.
    - Hyperparameter Tuning: Systematically adjusting hyperparameters such as learning rate, batch size, and optimizer types through methods like grid search or random search to identify optimal settings that maximize model performance.
    - Cross-Validation: Using cross-validation techniques to ensure the model's robustness and reliability by training and validating on different subsets of the dataset.

17.2 Advanced Hand Analysis

1. **Pot-Odds Calculation:**
    - Mathematical Integration: Developing modules to calculate pot-odds and equity based on the recognized hand and the current game state can provide players with critical insights into their betting strategies.
2. **Strategic Recommendations:**
    - Optimal Strategy Suggestions: Implementing algorithms that suggest optimal strategies based on hand strength, table position, and game dynamics can enhance decision-making for players.
    - Opponent Analysis: Incorporating features to analyze opponents' play styles and tendencies, helping users adapt their strategies accordingly. This could include tracking betting patterns, win rates, and common hand combinations.

17.3 Mobile Application Development

1. **Cross-Platform Compatibility:**
    - Android and iOS Support: Developing a mobile application that is compatible with both Android and iOS platforms, allowing users to access the system's capabilities on their smartphones or tablets for convenience.
2. **User-Friendly Design:**
    - Intuitive Interface: Creating a mobile interface that is intuitive and easy to use, ensuring a seamless user experience with responsive design and interactive elements.
3. **Real-Time Processing:**
    - On-Device Optimization: Ensuring the system runs efficiently on mobile devices by optimizing the model for lower computational power and leveraging on-device processing capabilities.
    - Cloud Integration: Utilizing cloud-based services for heavy computational tasks, allowing the mobile app to offload processing to remote servers and deliver results in real-time.

17.4 Multi-Player Integration

1. **Extending Recognition to Multi-Player Settings:**
    - Multi-Player Hand Recognition: Developing the system to recognize and analyze hands in a multi-player poker setting.
    - Identifying and tracking hands for several players simultaneously, ensuring recognition and analysis for each player in real-time.
2. **Comprehensive Game Analysis:**
    - Real-Time Multi-Player Analysis: Providing tools for real-time analysis of a multi-player game, including the ability to track community cards, player actions, and overall game state.

3. **Collaborative Features:**
    - Shared Insights: Implementing features that allow multiple users to collaborate and share insights during a game, enhancing the overall strategic depth and engagement.
    - Live Streaming Support: Integrating with live streaming platforms to provide real-time analysis and commentary for online poker games, offering viewers an enriched viewing experience.

17.5 Additional Future Directions

1. **Individual Card Recognition:**
    - Rank and Suit Identification: Enhancing the system to recognize individual card ranks and suits, providing more granular data and expanding its analytical capabilities.
2. **Ethical Safeguards:**
    - Responsible Use Enforcement: Developing mechanisms to ensure the system is used ethically and in compliance with online poker regulations, such as monitoring usage patterns and implementing restrictions for competitive play.
3. **User Customization:**
    - Personalized Settings: Allowing users to customize the system's interface and functionality according to their preferences, including adjustable display options, notification settings, and analysis depth.

In conclusion, future work on this project holds great potential to significantly advance its capabilities and applications. By focusing on enhanced model training, advanced hand analysis, mobile application development, and multi-player integration, the system can evolve into a comprehensive and versatile tool for both casual and serious poker players. These enhancements will not only improve the accuracy and utility of the system but also ensure it remains a cutting-edge resource in the realm of poker hand recognition and strategy analysis.

18. Conclusion

This project presents the development of an AI-powered poker hand recognition system. By utilizing computer vision and machine learning techniques, the system automates and streamlines hand identification, offering benefits for both casual and serious poker players. This project not only enhances gameplay efficiency but also serves as a valuable educational tool. Furthermore, it fosters the exploration of AI applications in real-world scenarios, paving the way for further innovation in the field.

PokerEye

AI-Powered Poker Hand Recognition System

Product of Aydie's Avenue

Aydie's Avenue

aydie.in

business@aydie.in