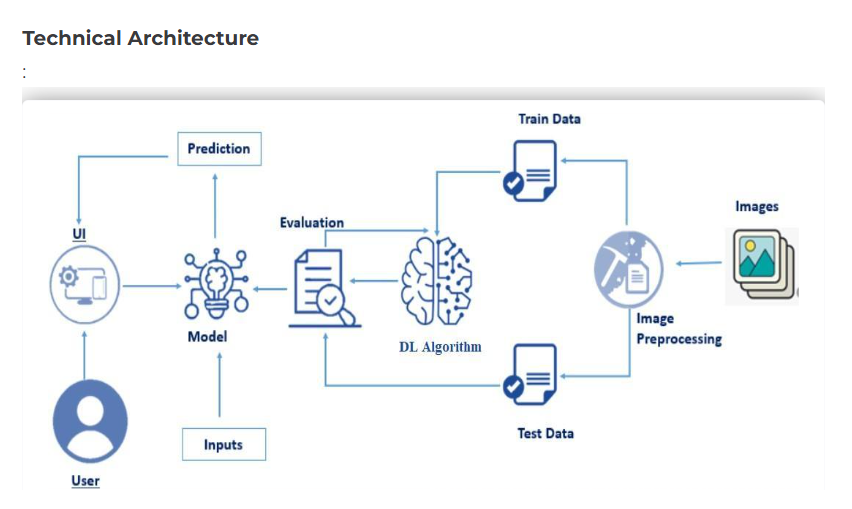
Bullseye Detection - Final Project Report

# Introduction

The Bullseye Detection project aims to automate the scoring of target-based activities such as archery, darts, and shooting sports. Utilizing a deep learning approach with MobileNetV2, this system classifies and evaluates bullseye targets from visual input, providing high-confidence feedback and visual analytics. The project follows milestones like problem understanding, data preparation, model training, performance evaluation, and deployment.



# Milestone 1: Problem Understanding and Planning

## Activity 1: Define Problem Statement

Manual scoring in target-based sports and defense training is error-prone and inefficient. This project addresses the need for an automated, reliable system to detect and score bullseye targets from image input. It enhances decision-making in sports, military surveillance, and manufacturing quality control by reducing human error.

## Activity 2: Project Proposal

We propose a vision-based system using MobileNetV2 for accurate bullseye classification. The model is trained on curated image datasets labeled as 'bulls eye' or 'not bulls eye'. Predictions are validated visually using circle detection techniques (HoughCircles) and confidence metrics.

## Activity 3: Initial Project Planning

The project was planned with milestones including data sourcing, preprocessing, model development, evaluation using metrics (accuracy, precision, F1), and visualization with seaborn and matplotlib.

# Milestone 2: Data Collection & Preparation

Data was collected from Kaggle and prepared using preprocessing techniques such as image resizing, normalization, class balancing, and directory structuring. Dummy images were used for the 'not bulls eye' class to simulate a two-class problem.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset. Link: <https://www.kaggle.com/datasets/ashwinsangareddypeta/bulls-eye-target-images-scraped-from-google>

Activity 1.1 import libraries

Import the necessary libraries as shown in the imageA screen shot of a computer program

AI-generated content may be incorrect.

Activity 1.2 Data Preparation

This section of the code focuses on data preparation for the bullseye classification model by organizing and simulating a balanced dataset. It first defines the source and target directories, then systematically creates the necessary folder structure for training and testing images by clearing any existing data and generating fresh folders for the two classes: "bulls eye" and "not bulls eye." Bullseye images are read from the Kaggle dataset, with all images copied into the training set and every fourth image also copied into the test set to ensure a proper evaluation split. Since actual images for the "not bulls eye" class are not available, the code handles this missing data scenario by generating blank white images using NumPy to simulate non-bullseye examples. These dummy images are saved into both the train and test directories to ensure the dataset remains balanced and the model can learn to distinguish between bullseye and non-bullseye patterns. This approach is a practical example of handling missing data through synthetic generation, helping maintain the integrity of the binary classification task.A screenshot of a phone

AI-generated content may be incorrect.

# Milestone 3: Exploratory Data Analysis & Preprocessing

Activity 1: Visual Analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

1. **Univariable analysis**:

This univariate plot helps us understand the class balance in our dataset. As seen in the image, there are significantly more "bulls eye" images than "not bulls eye" images. This kind of class imbalance is crucial to recognize before training a model, as it can bias the learning process toward the majority class. Knowing this early allows us to consider solutions like adding more negative samples, using synthetic data, or applying class weights.

A graph with blue rectangles

AI-generated content may be incorrect.

1. **Bivariable analysis:**

A box plot showing the confidence scores (model prediction probabilities) for each true class — “bulls eye” and “not bulls eye”. This plot helps us compare how confidently the model predicts for each class. The model appears to be slightly more confident on average when predicting "bulls eye" samples, as the box plot for "bulls eye" is taller and closer to the top of the y-axis (near 1.0). The presence of outliers in the "not bulls eye" class suggests inconsistent confidence, possibly due to fewer examples or less representative data. This insight helps us evaluate how reliable our model is across both classes.

A screenshot of a computer screen

AI-generated content may be incorrect.

1. **Multivariable analysis:**

This multivariable plot explores the relationship between confidence level and true class label. From the heatmap, we observe that all predictions for both "bulls eye" and "not bulls eye" fall into the “Very High” confidence bucket, indicating the model is highly confident — but this could also signal overconfidence. For instance, if predictions are very confident but inaccurate, the model might be misled by biased or insufficient data. This type of analysis allows us to monitor how prediction strength (confidence) aligns with the actual correctness of the predictions.

A blue and yellow box

AI-generated content may be incorrect.

Activity 2: Image Data Loading and Preprocessing

This section of the code is responsible for loading and preprocessing the image data for both training and testing. It begins by defining the paths to the train and test directories, which contain categorized folders for the "bulls eye" and "not bulls eye" images. The ImageDataGenerator is then initialized with a rescaling factor of 1./255, which normalizes pixel values to fall within the [0, 1] range — a standard preprocessing step that improves neural network training stability. Using the flow\_from\_directory() method, the images are loaded from their respective folders into train\_generator and test\_generator, with the images resized to 224x224 pixels and grouped into batches of 32. The class\_mode='categorical' setting ensures that the labels are one-hot encoded, which is necessary for multi-class classification. The final output confirms that the training dataset contains 34 images across 2 classes, and the test dataset includes 12 images across the same 2 classes — ensuring the model has sufficient data structure to begin training and evaluation.

A screenshot of a computer program

AI-generated content may be incorrect.

# Milestone 4: Model Building

The MobileNetV2 model was used with transfer learning. Only one model architecture was tested, with a GlobalAveragePooling layer followed by a dense classifier layer. The model was compiled with categorical crossentropy loss and Adam optimizer.

A screenshot of a computer code

AI-generated content may be incorrect.

# Milestone 5: Performance & Optimization

Evaluation included confusion matrix, classification report, and visual accuracy plots. Metrics like F1-score were reported using sklearn.

The performance of the bullseye detection model was evaluated using standard classification metrics such as precision, recall, F1-score, and accuracy. After training for 20 epochs using MobileNetV2, the model achieved an overall accuracy of 75%. Class-wise performance shows that it performs significantly better on the "bulls eye" class, with a precision, recall, and F1-score of 0.83, indicating strong reliability when detecting actual bullseyes. However, for the "not bulls eye" class, all three metrics dropped to 0.45, highlighting a need for improved representation or handling of that class. The macro average across classes stands at 0.64, and the weighted average F1-score is approximately 0.70, reflecting moderate overall model performance. These results suggest that while the model is well-optimized for detecting bullseyes, further tuning or data balancing is required to improve generalization, particularly for negative cases.

**A screenshot of a graph

AI-generated content may be incorrect.**

# Milestone 6: Model Evaluation and saving the model as Tflite

After training the MobileNetV2 model for 20 epochs, this section evaluates its performance on the test dataset. The evaluation is done using a separate ImageDataGenerator that rescales the test images and loads them from the test\_img\_dir. The model is then evaluated using the evaluate() method, which returns the loss and accuracy. If the test generator contains any images, the test accuracy is printed. This step helps validate how well the model generalizes to unseen data after training and is crucial for identifying underfitting, overfitting, or potential improvements needed in model tuning, The trained MobileNetV2 model was converted to TensorFlow Lite format for deployment on lightweight environments. A screenshot of a computer code

AI-generated content may be incorrect.

# Milestone 7: Model Interference and Bullseye Accuracy Calculation

This code block performs inference on a test image and calculates how close the detected circle is to the center of the image — referred to as bullseye accuracy. The preprocess\_image() function loads and resizes the image, normalizing it for model input. The detect\_inner\_circle() function then uses Hough Circle Transform (cv2.HoughCircles) to identify circular shapes in the image. Among all detected circles, the one closest to the image center is selected as the most likely bullseye. A green circle is drawn on it for visualization, and the distance to the center is converted into an accuracy percentage, where a smaller distance gives a higher bullseye score. For example, a prediction might result in a bullseye accuracy of 98.92%, indicating near-perfect alignment. This function combines computer vision and geometric distance to quantify how accurate the aim is — making it an essential evaluation metric beyond simple classification.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a target

AI-generated content may be incorrect.

# Final Submission and Deliverables

The final submission includes all essential components that reflect the complete development lifecycle of the project. This comprises a comprehensive explanation video demonstrating the solution end to end, along with detailed project documentation that outlines each stage—from data preparation and model training to evaluation and optimization. The documentation is structured according to the provided template to ensure clarity and consistency. These deliverables collectively demonstrate the technical implementation, decision-making process, and outcomes of the bullseye detection system.