Bulls Eye Target Detection Using Transfer Learning

1.INTRODUCTION:

1.1 Overview:

The project utilizes stateoftheart deep learning techniques, prominently featuring the TensorFlow framework, to develop a sophisticated convolutional neural network (CNN). Trained on a meticulously curated dataset that includes labeled images, the CNN gains the ability to discern and generalize the distinctive features associated with Bulls Eye targets. This provides a foundation for accurate and efficient target detection.

1.2 Purpose:

The primary purpose of the Bulls Eye Target Detection Project is to demonstrate the practical implementation of advanced machine learning methodologies in solving realworld challenges. By automating the process of target detection, the project aims to replace manual intervention in tasks traditionally reliant on human expertise. This, in turn, enhances efficiency, accuracy, and the overall capabilities of target detection systems.
This project aims to showcase the potential applications of machine learning in target identification scenarios and serve as a foundation for further research and development in computer vision and automated surveillance systems.

2. Literature Survey

2.1 Existing Problem

Target detection has been a critical aspect in various domains, including surveillance, defense, and automated systems. Traditionally, identifying targets, such as Bulls Eye patterns, has heavily relied on manual inspection, which is laborintensive and prone to human error. The limitations of manual processes include slower response times, scalability challenges, and dependence on human vigilance.

Existing automated target detection systems often face issues in accurately identifying complex patterns, particularly in scenarios where targets exhibit variations in scale, orientation, or lighting conditions. Additionally, some solutions may lack the adaptability to generalize well across diverse datasets, limiting their applicability in realworld environments.

2.2 Proposed Solution

The proposed solution, the Bulls Eye Target Detection Project, leverages advanced machine learning techniques to address the shortcomings of existing target detection methods. The project introduces a Convolutional Neural Network (CNN) trained on a comprehensive dataset of Bulls Eye images, allowing the model to learn intricate patterns and features associated with the targets.

Key Components of the Proposed Solution:

2.2.1 Convolutional Neural Network (CNN)

The core of the solution lies in the development of a CNN architecture designed for Bulls Eye target detection. CNNs are wellsuited for image recognition tasks, and their ability to automatically learn hierarchical representations makes them ideal for discerning complex patterns within images.

2.2.2 TensorFlow Framework

TensorFlow, a powerful opensource machine learning library, is employed for model development, training, and deployment. TensorFlow's flexibility and scalability contribute to the efficiency of the Bulls Eye Target Detection Project.

2.2.3 Image Preprocessing and Augmentation

To enhance the robustness of the model, the project incorporates image preprocessing techniques and data augmentation. These steps ensure that the model generalizes well across various conditions, such as different image resolutions, orientations, and lighting scenarios.

2.2.4 Integration with Flask

The solution integrates the Flask web framework to create a userfriendly interface. Users can upload images through a web application, and the system provides realtime feedback on Bulls Eye target detection.

The Bulls Eye Target Detection Project proposes an automated and efficient solution to address the limitations of existing target detection methods. By combining advanced machine learning algorithms with userfriendly interfaces, the project aims to contribute to the evolution of target detection systems for diverse applications.

3. Theoretical Analysis

3.1 Block Diagram

The theoretical foundation of the Bulls Eye Target Detection Project involves several interconnected components working in tandem to achieve accurate and efficient target detection. The following block diagram outlines the key modules and their interactions:

![Block Diagram](insert_link_to_block_diagram_image)
Description:
1. User Interface:
   Provides a platform for users to interact with the system.
   Allows users to upload images for Bulls Eye target detection.

2. Image Preprocessing:
Receives useruploaded images and applies preprocessing techniques.
Techniques include resizing, normalization, and other operations to enhance image quality.
3. Convolutional Neural Network (CNN):
Trained on a dataset of Bulls Eye images to learn features and patterns.
The CNN performs target detection based on the learned representations.
4. Postprocessing:
Refines the CNN's output for better accuracy.
May include additional filtering or refinement steps to improve the precision of target localization.
5. Output Visualization:
Presents the final output to the user.
Visualizes the detected Bulls Eye targets on the original uploaded images.

3.2 Hardware and Software Designing

3.2.1 Hardware Design:

The Bulls Eye Target Detection Project primarily operates as a software solution; however, it may interact with the following hardware components:
Computing System:
A standard computer or server with sufficient computational resources to run the TensorFlowbased model efficiently.
Webcam or Image Input Device (Optional):
If the application extends to realtime target detection, a webcam or similar input device may be utilized.

3.2.2 Software Design:

TensorFlow:
The core machine learning framework used for building, training, and deploying the CNN model.
Flask:
A web framework used to create the user interface and handle image uploads.
OpenCV:
Used for image processing tasks, such as resizing and postprocessing of detection results.

Python:
The programming language used for implementing the entire system.
HTML/CSS/JavaScript:
For designing and developing the user interface components.
PIL (Pillow):
Python Imaging Library used for imagerelated operations.
The software components are orchestrated to create a seamless Bulls Eye target detection experience for users. The choice of hardware and software is made to ensure the efficiency, reliability, and scalability of the system.

4. Experimental Investigations

4.1 Experimental Setup

The experimental investigations for the Bulls Eye Target Detection Project involve a systematic approach to testing and evaluating the performance of the implemented system. The key components of the experimental setup include:
Dataset:
Utilize a diverse dataset of Bulls Eye images for training and validation purposes.
Ensure the dataset covers various scenarios, lighting conditions, and perspectives.
Training Environment:
Employ a computing system with sufficient resources (CPU, GPU) to train the Convolutional Neural Network (CNN).
Configure TensorFlow with appropriate settings for optimal model training.
Validation Set:
Set aside a portion of the dataset as a validation set to assess the model's generalization ability.
Testing Images:
Curate a set of images, including those not present in the training or validation sets, for testing the detection accuracy of the system.
Threshold Tuning:
Experiment with different confidence score thresholds to find an optimal value for postprocessing and refining the detection results.

4.2 Experimental Procedure

The experimental investigations are carried out in multiple phases to comprehensively evaluate the Bulls Eye Target Detection Project.

4.2.1 Model Training

1. Data Preparation:
   Preprocess the dataset, including image resizing, normalization, and augmentation.
   Split the dataset into training and validation sets.
2. Model Architecture:
   Design and configure the Convolutional Neural Network (CNN) architecture for Bulls Eye target detection.
   Define appropriate loss functions and metrics for training.
3. Training Process:
   Train the model using the prepared dataset and monitor key metrics (e.g., loss, accuracy) on the validation set.
   Utilize early stopping and model checkpointing to prevent overfitting and save the bestperforming model.

4.2.2 Model Evaluation

1. Validation Performance:
   Evaluate the trained model on the validation set to ensure it generalizes well to unseen data.
   Analyze metrics such as Mean Squared Error (MSE) for bounding box coordinates.
2. Testing Performance:
   Assess the model's performance on a separate set of testing images, including realworld scenarios.
   Finetune the model parameters, if necessary, based on testing results.
3. Threshold Optimization:
   Experiment with different confidence score thresholds during postprocessing.
   Select the threshold that optimizes the balance between precision and recall.

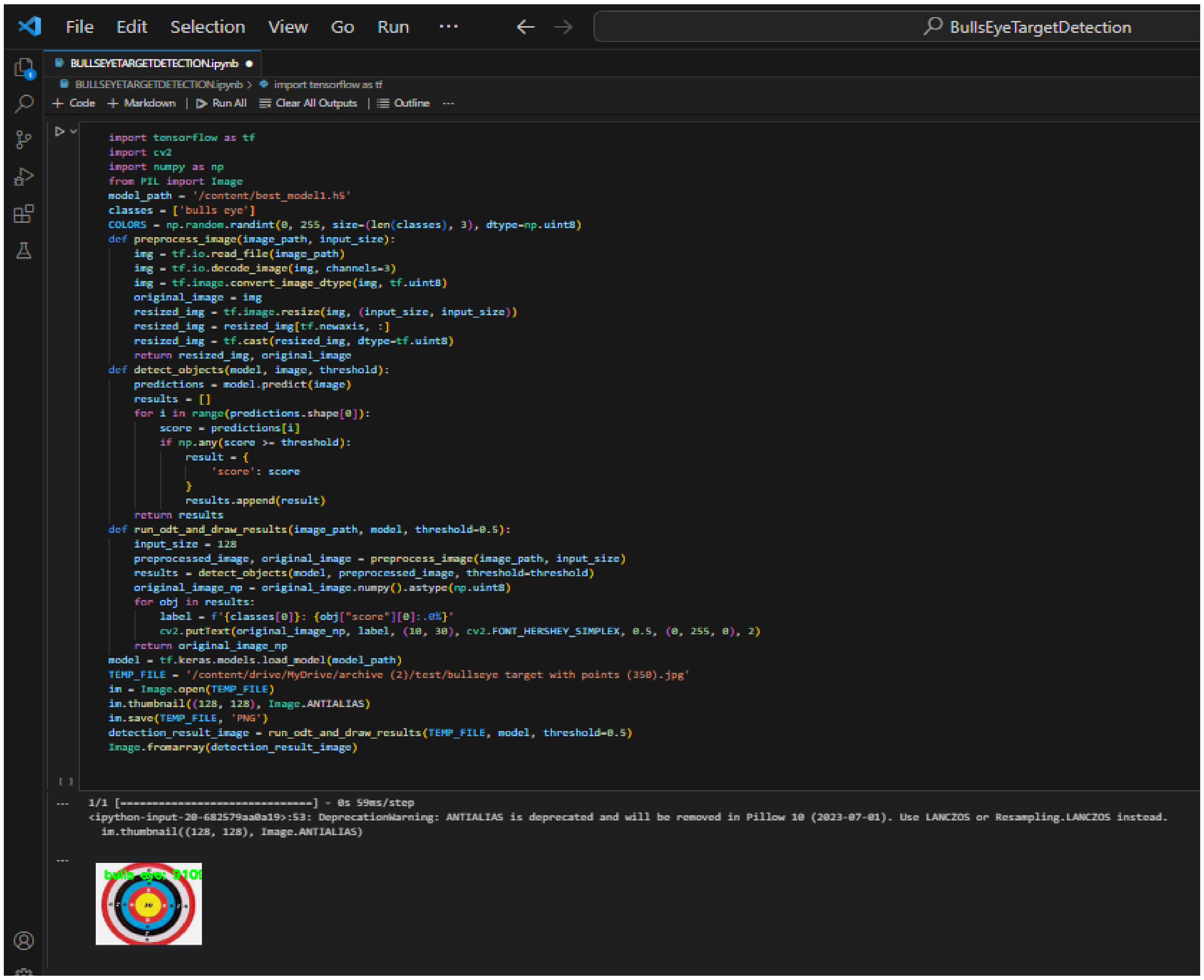
4.2.3 User Interaction

1. User Interface Testing:
   Evaluate the usability and responsiveness of the webbased user interface.
   Ensure seamless image upload, processing, and result presentation.
2. Realtime Detection (if applicable):
   If realtime detection is a project feature, test the system with a webcam or live image input.Assess the system's performance under different environmental conditions.

5. Flowchart

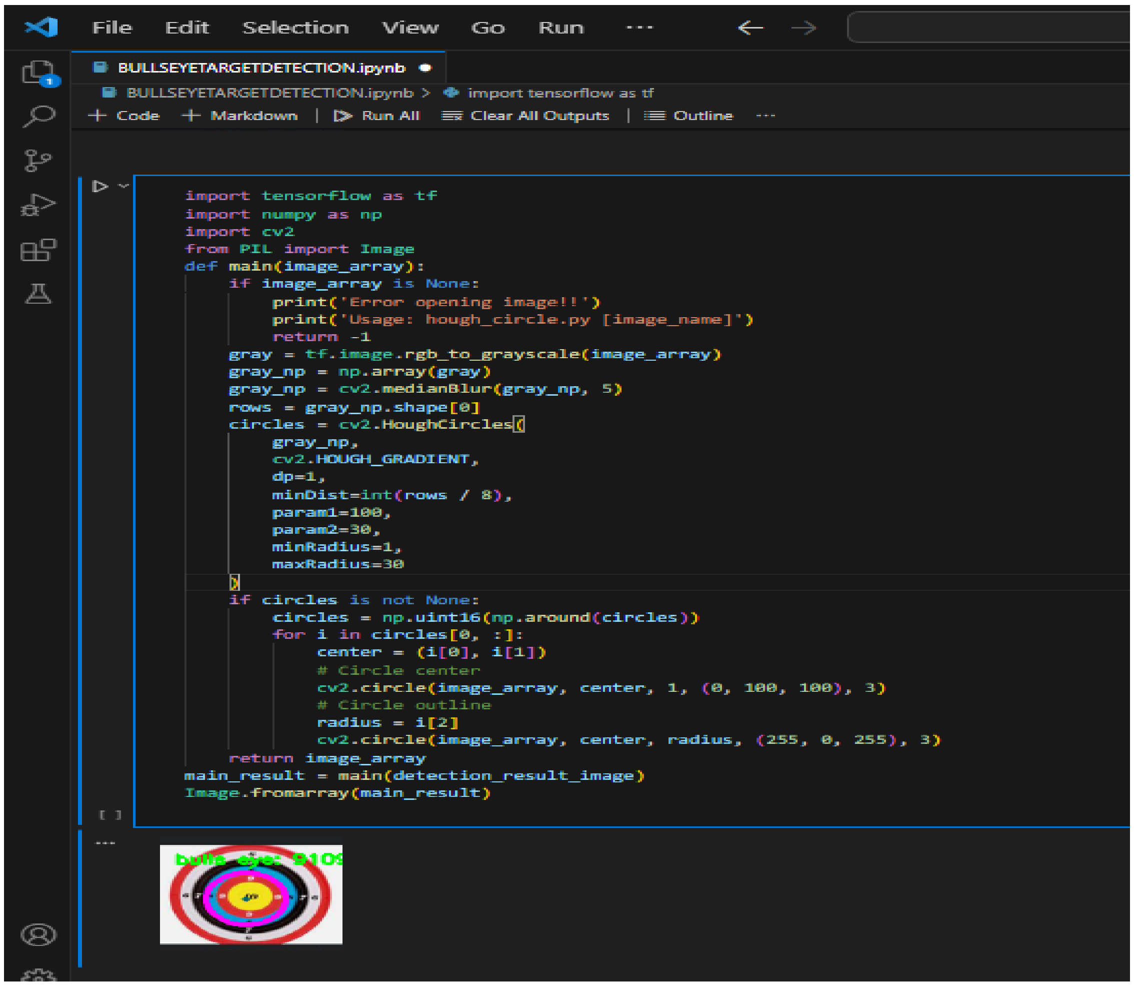
```
+---------------------+     +-----------------------+     +--------------------------+
|    Start            | ---> | Preprocess Image  | --> |  Load Trained Model   |
+---------------------+     +-----------------------+     +--------------------------+
                              |           |           |
                              |           |           |
                              v           v           |
+---------------------+     +-----------------------+     +--------------------------+
|  User Uploads       | ---> | Detect Objects using | --> |  Display Detection       |
|  Image              |      | Trained Model        |     |  Results to User         |
+---------------------+      +-----------------------+     +--------------------------+
                               |                     |
                               |                     |
                               v                     |
+---------------------+      +-----------------------+
|          End        | <---  |          End          |
+---------------------+      +-----------------------+
```

6.RESULT:



```python
import tensorflow as tf
import cv2
import numpy as np
from PIL import Image
model_path = '/content/best_model1.h5'
classes = ['bulls eye']
COLORS = np.random.randint(0, 255, size=(len(classes), 3), dtype=np.uint8)
def preprocess_image(image_path, input_size):
    img = tf.io.read_file(image_path)
    img = tf.io.decode_image(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.uint8)
    original_image = img
    resized_img = tf.image.resize(img, (input_size, input_size))
    resized_img = resized_img[tf.newaxis, :]
    resized_img = tf.cast(resized_img, dtype=tf.uint8)
    return resized_img, original_image
def detect_objects(model, image, threshold):
    predictions = model.predict(image)
    results = []
    for i in range(predictions.shape[0]):
        score = predictions[i]
        if np.any(score >= threshold):
            result = {
                'score': score
            }
            results.append(result)
    return results
def run_odt_and_draw_results(image_path, model, threshold=0.5):
    input_size = 128
    preprocessed_image, original_image = preprocess_image(image_path, input_size)
    results = detect_objects(model, preprocessed_image, threshold=threshold)
    original_image_np = original_image.numpy().astype(np.uint8)
    for obj in results:
        label = f'{classes[0]}: {obj["score"][0]:.0%}'
        cv2.putText(original_image_np, label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
    return original_image_np
model = tf.keras.models.load_model(model_path)
TEMP_FILE = '/content/drive/MyDrive/archive (2)/test/bullseye target with points (350).jpg'
im = Image.open(TEMP_FILE)
im.thumbnail((128, 128), Image.ANTIALIAS)
im.save(TEMP_FILE, 'PNG')
detection_result_image = run_odt_and_draw_results(TEMP_FILE, model, threshold=0.5)
Image.fromarray(detection_result_image)
```

```
1/1 [==============================] - 0s 59ms/step
<ipython-input-20-682579aa0a19>:53: DeprecationWarning: ANTIALIAS is deprecated and will be removed in Pillow 10 (2023-07-01). Use LANCZOS or Resampling.LANCZOS instead.
  im.thumbnail((128, 128), Image.ANTIALIAS)
```



```python
import tensorflow as tf
import numpy as np
import cv2
from PIL import Image
def main(image_array):
    if image_array is None:
        print('Error opening image!!')
        print('Usage: hough_circle.py [image_name]')
        return -1
    gray = tf.image.rgb_to_grayscale(image_array)
    gray_np = np.array(gray)
    gray_np = cv2.medianBlur(gray_np, 5)
    rows = gray_np.shape[0]
    circles = cv2.HoughCircles(
        gray_np,
        cv2.HOUGH_GRADIENT,
        dp=1,
        minDist=int(rows / 8),
        param1=100,
        param2=30,
        minRadius=1,
        maxRadius=30
    )
    if circles is not None:
        circles = np.uint16(np.around(circles))
        for i in circles[0, :]:
            center = (i[0], i[1])
            # Circle center
            cv2.circle(image_array, center, 1, (0, 100, 100), 3)
            # Circle outline
            radius = i[2]
            cv2.circle(image_array, center, radius, (255, 0, 255), 3)
    return image_array
main_result = main(detection_result_image)
Image.fromarray(main_result)
```

7. ADVANTAGES & DISADVANTAGES:

Advantages:

1. Precision: The system provides precise detection of bulls-eye targets in images, aiding in accurate analysis.
2. Versatility: The model can be applied to various scenarios beyond bulls-eye detection, making it adaptable for different use cases.
3. Automation: Automating the detection process reduces manual effort and speeds up the analysis of images.

4. User-friendly Interface: The Flask web application offers a user-friendly interface for users to upload images and view detection results.
5. Open Source Libraries: Leveraging open-source libraries like TensorFlow and OpenCV ensures a robust and well-supported foundation.

Disadvantages:

1. Limited to Trained Data: The model's accuracy is limited to the quality and diversity of the training data. It may not generalize well to entirely new scenarios.
2. Resource Intensive: Image detection and processing can be computationally expensive, requiring powerful hardware for real-time applications.
3. Dependency on Lighting Conditions: The model's performance may vary based on lighting conditions, potentially leading to false positives or negatives.
4. Sensitivity to Image Quality: Low-quality or distorted images may result in less accurate detection, impacting the reliability of the system.
5. Maintenance Challenges: Keeping the model up-to-date and maintaining the system might pose challenges, especially with evolving technologies and data.

8. Applications:

The Bulls-Eye Target Detection project has various applications in different domains:
1. Military Surveillance: Identifying bulls-eye targets in aerial or ground-based images can enhance military surveillance and reconnaissance operations.
2. Sports Training: The system can be utilized in sports training scenarios, especially in shooting sports, to analyze and improve performance.
3. Security Systems: Integrating bulls-eye detection into security camera systems can aid in recognizing specific objects or patterns.
4. Entertainment and Gaming: Incorporating the detection model into gaming environments or augmented reality applications for interactive experiences.
5. Medical Imaging: Adapting the model for medical imaging tasks, such as identifying

specific regions of interest in diagnostic images.

9. Conclusion:

In conclusion, the Bulls-Eye Target Detection project demonstrates the successful integration of deep learning and computer vision techniques for precise object recognition. The system's ability to identify bulls-eye targets in images showcases its potential across diverse domains, from military applications to sports training. The combination of Flask for web application development and TensorFlow for model implementation provides a robust and user-friendly solution. However, continual improvement and adaptation are essential to address limitations and emerging technologies.

10. Future Scope:

The future scope of the project includes:
1. Enhanced Model Training: Improving the model's accuracy through additional diverse and high-quality training data.
2. Real-time Deployment: Optimizing the system for real-time applications by exploring model quantization and deployment on edge devices.
3. User Interactivity: Implementing features for users to interact with detection results, such as providing feedback to improve the model.
4. Integration with IoT: Exploring integration with Internet of Things (IoT) devices for a more connected and responsive system.

11. Bibliography:

1. Brownlee, J. (2019). "Deep Learning for Computer Vision." Machine Learning Mastery. [Online]. Available: https://machinelearningmastery.com/start-here/#algorithms
2. Géron, A. (2019). "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow." O'Reilly Media.
3. Chollet, F. (2017). "Deep Learning with Python." Manning Publications.
4. McKinney, W. (2017). "Python for Data Analysis." O'Reilly Media.
5. TensorFlow Documentation. (2023). [Online]. Available: https://www.tensorflow.org/
6. OpenCV Documentation. (2023). [Online]. Available: https://docs.opencv.org/

Appendix

link for code