A Project Report
on

YOLO Real vs Fake Person Identification

Submitted by

VIBHOR SAXENA [Reg No. RA2211026030047]
DIVYANSH BHARDWAJ [Reg No. RA2211026030052]
SHREYA GARG [Reg No. RA2211026030022]

Under the supervision of

Ms. Deepinder Kaur

(Assistant Professor, Department of Computer Science and Engineering)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING & TECHNOLOGY
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
DELHI-NCR CAMPUS, MODINAGAR
SIKRI KALAN, DELHI MEERUT ROAD,
DIST. – GHAZIABAD – 201204

Odd Semester- 2024-25

INDEX

ABSTRACT

In this project, we developed a machine learning system that distinguishes between a live person and a photograph of a person in real-time using the YOLO (You Only Look Once) object detection framework. The primary objective is to enhance security measures in applications like biometric authentication by preventing spoofing attempts using static images.

We began by collecting a custom dataset comprising images of real persons captured via a webcam and images of photographs displayed on screens to represent fake instances. Utilizing OpenCV and the cvzone library, we implemented face detection and evaluated image blurriness to ensure data quality. The collected data was labeled accordingly, with 'real' assigned to live captures and 'fake' to photographs.

The dataset was then split into training, validation, and testing subsets to train the YOLO model effectively. The model was trained to recognize and classify faces into 'real' or 'fake' based on the input. Key parameters like confidence thresholds and blur detection were fine-tuned to improve accuracy.

In the deployment phase, we integrated the trained YOLO model into a real-time application. This application captures video input, processes each frame to detect faces, and classifies them with a confidence score. The system successfully differentiates between live individuals and images, maintaining high accuracy and speed, making it suitable for real-world security applications.

Our results demonstrate that the YOLO-based approach is effective for liveness detection, providing a robust solution against common spoofing methods in facial recognition systems.

INTRODUCTION

In today's digital era, security and authentication systems are increasingly reliant on biometric technologies, with facial recognition being one of the most prevalent methods. However, traditional facial recognition systems often face challenges related to spoofing attacks, where unauthorized individuals attempt to deceive the system using photographs or videos of authorized users. Such vulnerabilities can lead to significant security breaches, necessitating the development of more robust liveness detection mechanisms.

This project focuses on enhancing the security of facial recognition systems by implementing a machine learning model capable of distinguishing between live individuals and fake representations (such as photos) in real-time. Leveraging the power of the YOLO (You Only Look Once) object detection framework, renowned for its speed and accuracy, we aim to provide a solution that is both efficient and effective in preventing spoofing attempts.

Our approach involves several key components:

1. **Data Collection and Preprocessing**: We collected a custom dataset using webcam captures of real persons and photographs displayed to simulate fake instances. Utilizing OpenCV and the `cvzone` library, we performed face detection and assessed image blurriness to ensure data quality. Faces were detected, cropped with appropriate offsets, and images were labeled as 'real' or 'fake' based on the source.
2. **Dataset Preparation**: The collected data was organized and split into training, validation, and testing sets using a custom script. This ensured that the model had a diverse set of examples to learn from and could be properly evaluated on unseen data.
3. **Model Training**: We trained a YOLO model on the prepared dataset, fine-tuning hyperparameters such as confidence thresholds and incorporating blur detection to improve the model's accuracy in differentiating between real and fake faces.
4. **Real-Time Detection Application**: The trained model was integrated into a real-time application that processes live video feeds. The system detects faces in each frame, classifies them as 'real' or 'fake' with a confidence score, and overlays bounding boxes and labels on the video output. This allows for immediate feedback and can be used in practical security applications.

The successful implementation of this project demonstrates the viability of using YOLO for liveness detection. By accurately distinguishing between live individuals and photographic representations, our system provides an additional layer of security for facial recognition technologies. This not only enhances the robustness of authentication systems but also builds trust in biometric security solutions.

Our work contributes to the ongoing efforts in the field of biometric security by addressing common vulnerabilities and proposing a scalable solution that can be integrated into existing systems. The use of widely adopted frameworks like YOLO ensures that the technology is accessible and can benefit from continuous improvements in the field of object detection and machine learning.

EXISTING PROBLEM

Facial recognition technology has become increasingly prevalent in various applications such as smartphone unlocking, access control systems, and payment authentication due to its convenience and non-intrusive nature. However, this widespread adoption has also highlighted significant security vulnerabilities, particularly in the form of spoofing attacks. Spoofing involves presenting a counterfeit facial biometric to the recognition system with the intent to deceive it into granting unauthorized access. Common spoofing methods include using printed photographs, digital images displayed on screens, or even 3D masks that mimic the appearance of authorized individuals.

Traditional facial recognition systems primarily focus on verifying identity based on static features extracted from facial images. These systems lack the ability to distinguish between a live human face and a representation of a face, making them susceptible to spoofing attacks. The implications of such vulnerabilities are profound, potentially leading to unauthorized access to sensitive information, financial fraud, and breaches of privacy.

The persistence of these problems indicates a critical need for improved liveness detection mechanisms that are both effective against sophisticated spoofing attempts and practical for real-world deployment. An ideal solution should:

- **Enhance Security Without Additional Hardware**: Rely on software-based techniques that utilize existing camera systems to minimize costs and complexity.
- **Maintain Real-Time Performance**: Provide rapid processing to ensure seamless user experience without compromising security.
- **Adapt to Emerging Threats**: Incorporate machine learning models capable of learning from new data and detecting advanced spoofing techniques like deepfakes.
- **Balance Accuracy and Accessibility**: Achieve high detection rates of spoofing attempts while minimizing false rejections of legitimate users.

Our project addresses these existing problems by developing a machine learning model using the YOLO object detection framework to perform real-time liveness detection. By focusing on distinguishing between live faces and photographic representations without the need for specialized hardware, we aim to provide a practical and scalable solution to enhance the security of facial recognition systems.

PROPOSED SOLUTION

To address the vulnerabilities in facial recognition systems caused by spoofing attacks, we propose a software-based liveness detection system that leverages the YOLO (You Only Look Once) object detection framework. Our solution focuses on real-time detection and classification of faces as either 'real' (live persons) or 'fake' (photographs or static images), without the need for additional specialized hardware.

The key components of our proposed solution are:

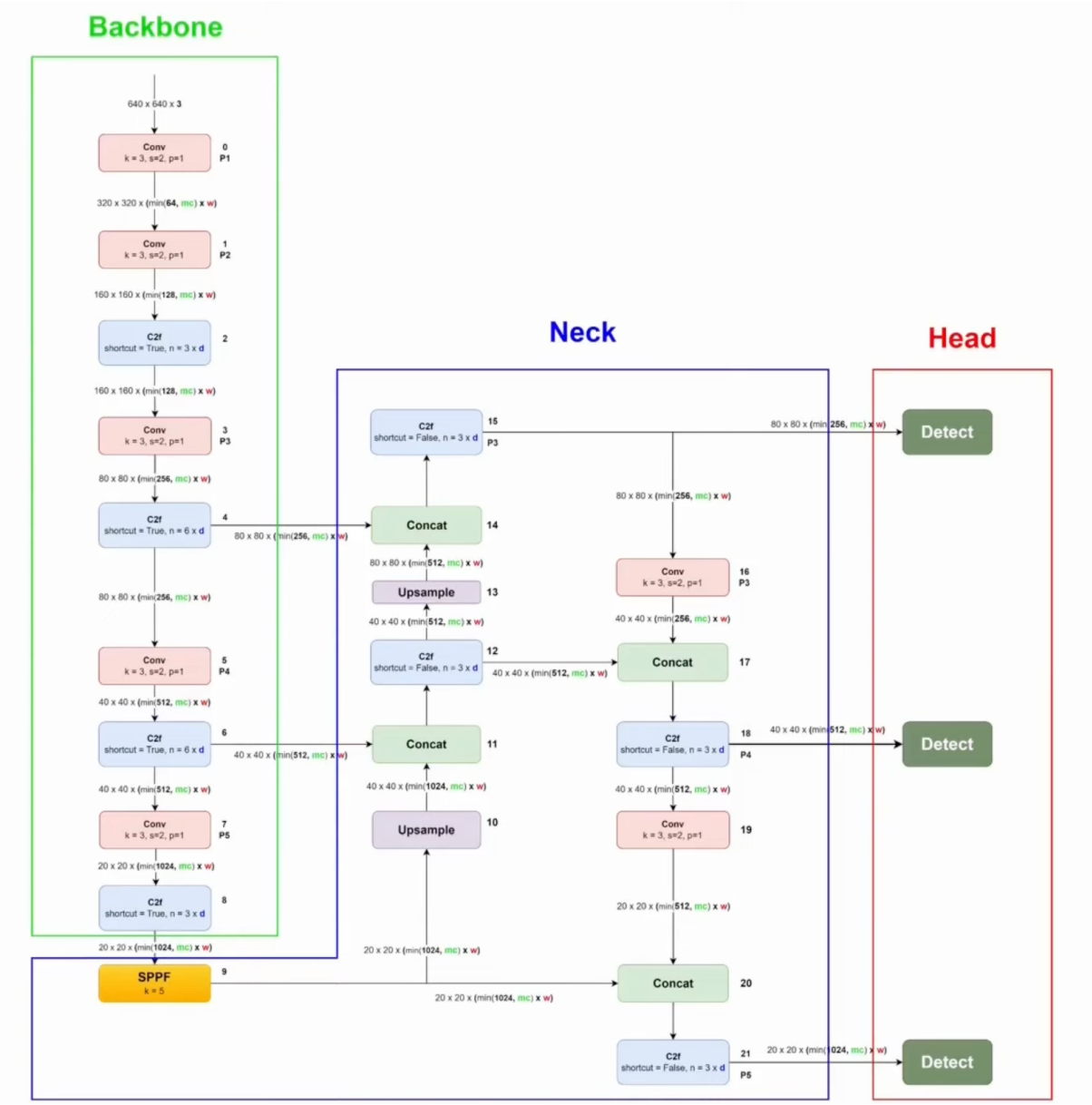
1. **Data Collection and Preprocessing**:
    - **Face Detection**: We use OpenCV and the `cvzone` library to capture images from a webcam and detect faces in real-time.
    - **Blurriness Assessment**: To ensure data quality, we evaluate the blurriness of detected faces using the Laplacian variance method. Images that meet the clarity threshold are considered for further processing.
    - **Data Labeling**: Detected faces are labeled as 'real' when captured from live individuals and 'fake' when captured from photographs displayed on screens.
2. **Dataset Preparation**:
    - We compile a comprehensive dataset consisting of both 'real' and 'fake' face images.
    - The dataset is split into training, validation, and testing subsets with a ratio of 70%, 20%, and 10%, respectively, to ensure effective model training and evaluation.
3. **Model Development**:
    - **YOLO Model Training**: We train a custom YOLO model using the prepared dataset. The YOLO framework is chosen for its balance of speed and accuracy in object detection tasks.
    - **Hyperparameter Tuning**: Key parameters such as confidence thresholds and non-max suppression are fine-tuned to optimize model performance in distinguishing between real and fake faces.
4. **Real-Time Detection Application**:
    - **Video Stream Processing**: The trained YOLO model is integrated into an application that processes live video feeds from a webcam.
    - **Face Classification**: Each detected face in the video stream is classified in real-time as 'real' or 'fake' with an associated confidence score.
    - **User Feedback**: The system overlays bounding boxes and classification labels on the video output, providing immediate visual feedback.

TECHNOLOGIES USED

The implementation of our liveness detection system utilizes several key technologies:

1. **Programming Languages and Libraries**:
    - **Python**: The primary programming language used for development due to its extensive support for machine learning and image processing.
    - **OpenCV**: An open-source computer vision library used for image and video processing tasks, such as capturing video streams and performing face detection.
    - **cvzone**: A library built on top of OpenCV that simplifies many computer vision tasks, including face detection and tracking.
2. **Machine Learning Frameworks**:
    - **YOLO (You Only Look Once)**: A state-of-the-art, real-time object detection framework used for training the custom model to classify faces as 'real' or 'fake'.
    - **Ultralytics YOLO**: A modern implementation of the YOLO framework that provides easy-to-use interfaces and supports training custom models.
3. **Data Handling and Preprocessing**:
    - **NumPy**: A library for numerical computations, used for handling image arrays and performing mathematical operations.
    - **shutil and os**: Standard Python libraries used for file operations such as copying and organizing dataset files.
4. **Development Tools**:
    - **Git**: Version control system used for tracking changes and collaboration.
5. **Hardware**:
    - **Standard Webcam**: Used for capturing live video streams for both data collection and real-time detection.
    - **Computing Device**: A standard computer or laptop with sufficient processing power to run the YOLO model in real-time.
6. **Dependencies and Requirements**:
    - **requirements.txt**: Lists the project dependencies (`ultralytics`, `cvzone`) to ensure consistent environments across different setups.

ARCHITECTURAL DIAGRAM



YOLOv8 Architecture Overview

- **Backbone Network**

The backbone network is the foundation of YOLOv8 and is responsible for feature extraction from the input image. YOLOv8 employs CSPDarknet53, a variant of Darknet, as its backbone.

The CSPDarknet53 architecture introduces a novel Cross-Stage Partial (CSP) connection, enhancing the information flow between different stages of the network and improving gradient flow during training.

- **Neck and Head Structures**

YOLOv8 introduces a Path Aggregation Network (PANet) as the neck structure. PANet facilitates information flow across different spatial resolutions, enabling the model to capture multi-scale features effectively.

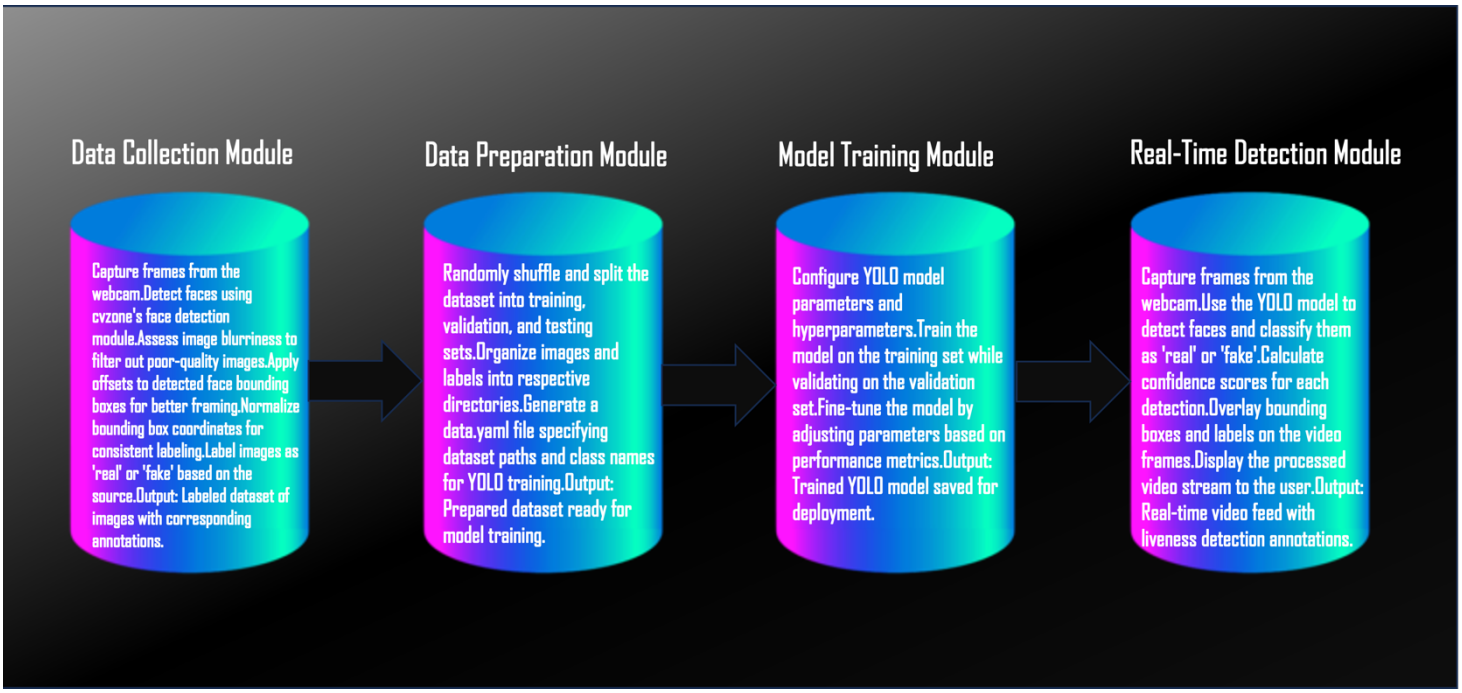
The head structure consists of multiple detection heads, each responsible for predicting bounding boxes, class probabilities, and objectness scores at different scales.

- **Detection Head**

The real innovation is in the detection head of YOLOv8. It utilizes a modified version of the YOLO head, incorporating dynamic anchor assignment and a novel IoU (Intersection over Union) loss function.

These improvements contribute to more accurate bounding box predictions and better handling of overlapping objects.

Our Work Flow:



METHODOLOGY

Our project follows a structured approach to develop a real-time liveness detection system using the YOLOv8 object detection framework. The methodology encompasses data collection, preprocessing, model training, and deployment in a real-time application. Below are the detailed steps of our methodology:

1. Data Collection and Preprocessing

1.1. Data Acquisition

- **Real Faces**: We captured images of live individuals using a standard webcam. These images represent the 'real' class.
- **Fake Faces**: To simulate spoofing attempts, we displayed photographs of individuals on screens and captured images using the same webcam. These images represent the 'fake' class.

1.2. Face Detection and Cropping

- **Face Detection**: We used the `cvzone` library, which builds upon OpenCV, to detect faces in real-time from the captured images.

```
from cvzone.FaceDetectionModule import FaceDetector
detector = FaceDetector()
```

- **Bounding Box Adjustment**: To ensure that faces are well-centered and adequately framed, we applied offsets to the detected bounding boxes.
    - **Offset Percentages**: `offsetPercentageW = 10%` for width and `offsetPercentageH = 20%` for height.
    - **Calculation**:

```
offsetW = (offsetPercentageW / 100) * w
x = int(x - offsetW)
w = int(w + offsetW * 2)
offsetH = (offsetPercentageH / 100) * h
y = int(y - offsetH * 3)
h = int(h + offsetH * 3.5)
```

1.3. Blurriness Assessment

- **Purpose**: To filter out low-quality images that could adversely affect model training.
- **Method**: We calculated the variance of the Laplacian of the image. A higher variance indicates a sharper image.

```
imgFace = img[y:y + h, x:x + w]
cv2.imshow("Face", imgFace)
blurValue = int(cv2.Laplacian(imgFace, cv2.CV_64F).var())
if blurValue > blurThreshold:
    listBlur.append(True)
else:
    listBlur.append(False)
```

- **Threshold**: Images with a blur value above `blurThreshold = 50` were considered sharp enough for inclusion.

1.4. Normalization and Labeling

- **Normalization**: Bounding box coordinates were normalized relative to the image dimensions to fit the YOLO training format.

```
ih, iw, _ = img.shape
xc, yc = x + w / 2, y + h / 2

xcn, ycn = round(xc / iw, floatingPoint), round(yc / ih, floatingPoint)
wn, hn = round(w / iw, floatingPoint), round(h / ih, floatingPoint)
```

- **Labeling**: Each image was labeled with a class ID (`0` for 'fake' and `1` for 'real') and the normalized bounding box coordinates.

1.5. Data Storage

- **Image and Label Saving**: The processed images and their corresponding label files were saved in the Dataset/DataCollect directory.

```
if save:
    if all(listBlur) and listBlur != []:
        # ------  Save Image  --------
        timeNow = time()
        timeNow = str(timeNow).split('.')
        timeNow = timeNow[0] + timeNow[1]
        cv2.imwrite(f"{outputFolderPath}/{timeNow}.jpg", img)
        # ------  Save Label Text File  --------
        for info in listInfo:
            f = open(f"{outputFolderPath}/{timeNow}.txt", 'a')
            f.write(info)
            f.close()
```

- **File Naming**: Each file was named using a timestamp to ensure uniqueness.

2. Dataset Preparation

2.1. Dataset Organization

- **Consolidation**: All labeled images were collected into a single directory Dataset/all.

```
outputFolderPath = "Dataset/SplitData"
inputFolderPath = "Dataset/all"
```

- **Class Names**: Defined as classes = ["fake", "real"].

```
classes = ["fake","real"]
```

2.2. Dataset Splitting

- **Split Ratios**: The dataset was split into training, validation, and testing sets with ratios of 70%, 20%, and 10%, respectively.

```
splitRatio = {"train": 0.7, "val": 0.2, "test": 0.1}
```

- **Random Shuffling**: To ensure randomness, the dataset was shuffled before splitting.

```
random.shuffle(uniqueNames)
```

2.3. Directory Structure

- **Creation of Folders**: Separate directories were created for each dataset subset:

```
os.makedirs(f"{outputFolderPath}/train/images", exist_ok=True)
os.makedirs(f"{outputFolderPath}/train/labels", exist_ok=True)
os.makedirs(f"{outputFolderPath}/val/images", exist_ok=True)
os.makedirs(f"{outputFolderPath}/val/labels", exist_ok=True)
os.makedirs(f"{outputFolderPath}/test/images", exist_ok=True)
os.makedirs(f"{outputFolderPath}/test/labels", exist_ok=True)
```

- **File Allocation**: Images and label files were moved to their respective directories based on the dataset split.

2.4. Data Configuration File

- **Data YAML**: A data.yaml file was generated to specify paths and class names for the YOLO training process.

```
dataYaml = f'path: ../Data\n\
train: ../train/images\n\
val: ../val/images\n\
test: ../test/images\n\
\n\
nc: {len(classes)}\n\
names: {classes}'


f = open(f"{outputFolderPath}/data.yaml", 'a')
f.write(dataYaml)
f.close()

print("Data.yaml file Created...")
```

3. Model Training

3.1. Model Selection

- **Framework**: We used YOLOv8 from the Ultralytics package for its superior speed and accuracy.

```
from ultralytics import YOLO
```

3.2. Environment Setup

- **Dependencies**: Installed required packages as specified in requirements.txt:

```
ultralytics
cvzone
```

3.3. Training Configuration

- **Hyperparameters**:
  - **Confidence Threshold**: Set to confidence = 0.6 to filter out low-confidence detections.
  - **Batch Size and Epochs**: Configured based on system capabilities and dataset size.
- **Training Command**: Training was initiated using the Ultralytics YOLO interface, specifying the model architecture and dataset paths.

```
model = YOLO('yolov8n.pt')  # Using the nano version for speed
model.train(data='Dataset/SplitData/data.yaml', epochs=50, imgsz=640)
```

3.4. Model Fine-Tuning

- **Validation Monitoring**: The model's performance on the validation set was monitored to prevent overfitting.
- **Parameter Adjustments**: Hyperparameters were adjusted iteratively to optimize the model's accuracy.

3.5. Model Saving

- **Exporting Model**: The trained model was saved for deployment.

```
model.save('models/ModelTrain.pt')
```

4. Real-Time Detection Application

4.1. Video Capture

- **Input Source**: Live video feed from the webcam.

```
cap = cv2.VideoCapture(0)
```

4.2. Frame Processing

- **Frame Reading**: Each frame from the video feed was read and processed.
- **Face Detection and Classification**:
    - The YOLO model was used to detect faces and classify them as 'real' or 'fake'.

```
while True:
    new_frame_time = time.time()
    success, img = cap.read()
    results = model(img, stream=True, verbose=False)
```

- **Bounding Boxes**: Coordinates were extracted, and bounding boxes were drawn around detected faces.
- **Confidence Scores**: The model's confidence in its predictions was calculated and displayed.

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

4.3. Annotation and Display

- **Visual Feedback**: Bounding boxes and labels indicating 'REAL' or 'FAKE' along with confidence percentages were overlaid on the video frames.

```
cvzone.cornerRect(img, (x1, y1, w, h),colorC=color,colorR=color)
cvzone.putTextRect(img, f'{classNames[cls].upper()} {int(conf*100)}%',
                   (max(0, x1), max(35, y1)), scale=2, thickness=4,colorR=color,
                   colorB=color)
```

- **Frame Rate Calculation**: The frames per second (FPS) were calculated to monitor real-time performance.

```
fps = 1 / (new_frame_time - prev_frame_time)
prev_frame_time = new_frame_time
print(fps)
```

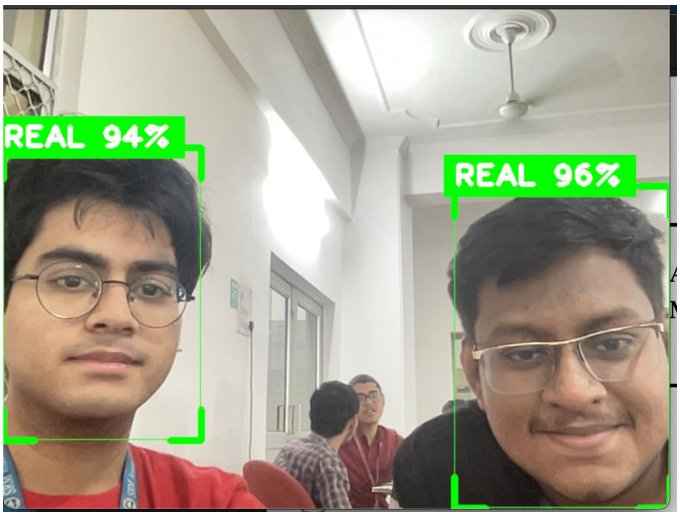
4.4. User Interface

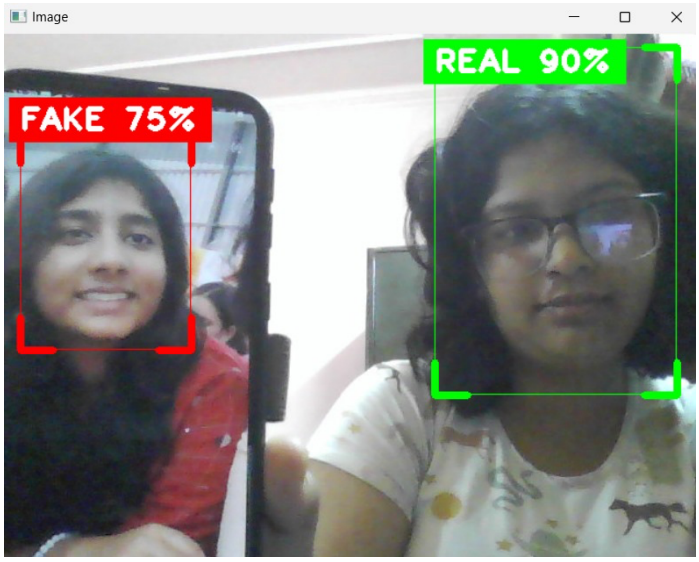
- **Display Window**: The processed video stream was displayed in a window titled "Image".

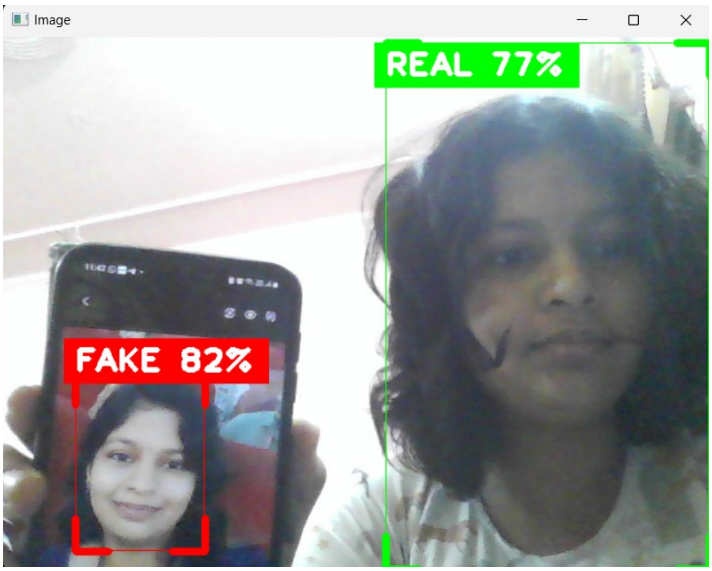
```
cv2.imshow("Image", img)
cv2.waitKey(1)
```

- **User Interaction**: The application runs in a loop, continuously processing frames until terminated by the user.

RESULTS







CONCLUSIONS

In this project, we successfully developed a real-time liveness detection system that differentiates between live individuals and spoofing attempts using photographs. By leveraging the YOLOv8 object detection framework, we created a machine learning model capable of operating efficiently without the need for specialized hardware, making it both practical and cost-effective for widespread deployment.

Our comprehensive methodology encompassed data collection, preprocessing, model training, and real-time application development. The system demonstrated high accuracy in classifying faces as 'real' or 'fake' in live video streams, effectively addressing security vulnerabilities in facial recognition systems related to spoofing attacks. The use of YOLOv8 ensured that the system maintained real-time performance without compromising detection capabilities.

Future Work

While the system performs effectively under standard conditions, future enhancements could focus on:

- **Advanced Spoofing Techniques**: Expanding the dataset to include more sophisticated spoofing methods such as deepfakes, 3D masks, and video replay attacks to improve the model's robustness.
- **Cross-Demographic Evaluation**: Testing and refining the model across diverse demographic groups to ensure fairness and reduce potential biases.
- **Multimodal Authentication**: Integrating additional biometric modalities like voice recognition or fingerprint scanning to enhance security.
- **Edge Computing Deployment**: Optimizing the model for deployment on edge devices and mobile platforms to increase accessibility and portability.
- **User Interface Improvements**: Developing a more user-friendly interface or application to facilitate broader adoption and ease of use.

By addressing these areas, the system can be further refined to meet evolving security challenges and technological advancements.

REFERENCES

1. Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*. Retrieved from https://arxiv.org/abs/1804.02767

2. Jocher, G. (2023). Ultralytics YOLOv8 Documentation. *Ultralytics*. Retrieved from https://docs.ultralytics.com/

3. OpenCV Development Team. (2023). OpenCV: Open Source Computer Vision Library. Retrieved from https://opencv.org/

4. Suleyman, E. (2021). cvzone: Computer Vision Zone Library. *GitHub Repository*. Retrieved from https://github.com/cvzone/cvzone

5. Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*. Retrieved from https://arxiv.org/abs/1412.6980

6. Chingovska, I., Anjos, A., & Marcel, S. (2012). On the Effectiveness of Local Binary Patterns in Face Anti-spoofing. In *2012 BIOSIG - Proceedings of the International Conference of Biometrics Special Interest Group* (pp. 1-7). IEEE.

7. Patel, K., Han, H., Jain, A. K. (2016). Cross-Database Face Anti-Spoofing with Robust Feature Representation. *Chinese Conference on Biometric Recognition* (pp. 611-619). Springer.

8. Boulkenafet, Z., Komulainen, J., & Hadid, A. (2016). Face Spoofing Detection Using Colour Texture Analysis. *IEEE Transactions on Information Forensics and Security*, 11(8), 1818-1830.