Face Detection and Enhancement Pipeline for CCTV Footage

Naman

October 2, 2024

1 Introduction

In this part of problem statement, we developed a system for detecting and enhancing faces from CCTV footage. The pipeline captures frames from the video, detects faces, performs image enhancements such as denoising and lighting improvements, and displays the final enhanced face images(given to next part of the pipeline for further processing). Further models highlighted below this part can be used to enhance the image quality even more!

2 Methodology

The methodology can be divided into several key steps: video processing, face detection, and image enhancement.

2.1 Video Processing

We used the `OpenCV` library to load and process frames from a video. In order to speed up the process and reduce computational load, we employed frame skipping, processing every 5th frame rather than every frame in the video.However in the original code we are not skipping and frame. Additionally, we resized the frames to 640x360 resolution before running the face detection model to further reduce the computational cost but there was not any significant difference in speed as can be seen in the colab file provided for the same.

2.2 Face Detection

For face detection, we employed the **RetinaFace** model. RetinaFace is known for its accuracy in detecting faces across a wide range of poses, including profiles, and its robustness in various environmental conditions.

3 Why This Model

Face Detection Model Selection for Facial Recognition

4 Introduction

Face detection is the first critical step in a facial recognition pipeline. A robust face detector ensures that the system accurately identifies faces in images, which directly impacts the downstream tasks. Selecting the right model, however, requires careful consideration of multiple factors like accuracy, speed, and the operational context.

5 Criteria for Choosing a Face Detector

When comparing face detection models, the main trade-off is between accuracy and speed. However, additional factors such as false positives, false negatives, and performance on various face poses or lighting conditions must also be considered.

Many existing articles and studies on face detection are either biased toward specific models or focus on particular implementations. Additionally, different model implementations, even with the same neural network architecture, may lead to variations in performance. The ideal model largely depends on the project context and specific requirements, such as the need for high precision or real-time processing.

5.1 CPU Model: YuNet

YuNet, designed for edge computing, impressed me with its speed and ability to detect larger faces effectively. However, it struggles with smaller faces and tends to produce some false positives. Despite these limitations, its speed makes it a good candidate for real-time applications.

Strengths: Fast, low memory usage, real-time capable on edge devices.
Weaknesses: Struggles with small faces, lower accuracy than GPU models.

5.2 GPU Models

5.2.1 Dlib

Dlib is known for balancing accuracy and performance. However, its accuracy improves only with upsampling, which slows down processing significantly and can crash if memory is limited. Dlib does minimize false positives, which can be valuable for reducing errors downstream.

Strengths: Minimizes false positives, adjustable via confidence thresholds.
Weaknesses: Slower with upsampling, lower accuracy at default settings.

5.2.2 OpenCV DNN

OpenCV's face detection model is optimized for speed but offers mediocre accuracy. It is incredibly fast but did not perform well enough to warrant use in my project.

Strengths: Fast, easy to implement.
Weaknesses: Low accuracy, not suitable for challenging datasets.

5.2.3 MTCNN

MTCNN, while more accurate than OpenCV DNN, falls short in comparison to the other models in both accuracy and speed. Given its slower processing time and lower accuracy, I didn't find it compelling for my project.

Strengths: More accurate than OpenCV, handles varying face poses.
Weaknesses: Slower and less accurate than YuNet and RetinaFace.

5.2.4 RetinaFace

RetinaFace emerged as the most accurate model in my tests. It excelled at identifying even challenging faces, such as those in reflections or partially occluded, but its false positive rate was relatively high. However, these false positives often included faces I hadn't annotated, indicating its sensitivity to faces.

Strengths: High accuracy, detects difficult faces, handles different poses well.
Weaknesses: Some false positives, slower than YuNet.

6 Finalizing

After testing multiple models, I concluded that YuNet is best suited for real-time applications where speed is critical, while RetinaFace is the go-to model for projects prioritizing accuracy, even in challenging scenarios. Dlib could be useful when minimizing false positives is the primary goal, but it did not offer significant enough improvements in my tests to justify its slower performance.

For my project, which involves large-scale facial recognition, RetinaFace strikes the right balance between accuracy and speed. **This is referenced from Medium Article**

6.1 Image Enhancement

After detecting the face, we perform image enhancement to make the face clearer and more recognizable. This includes denoising and lighting improvement to address common issues in CCTV footage, such as poor lighting, noise, and blurriness.

- `cv2.fastNlMeansDenoisingColored()`: This function reduces noise in the extracted face images. It was used to clean up the detected faces and improve their clarity.
- `cv2.filter2D()`: After denoising, we applied sharpening filters to enhance the edges and improve lighting in the face images. This upscaling method helps in enhancing the overall quality of the image.

7 Pipeline Overview

1. Load video using `cv2.VideoCapture()`.
2. Process every 5th frame(optional) to reduce computational load.
3. Detect faces using RetinaFace.
4. For each detected face:
   - Crop the face based on the bounding box.
   - Enhance the face using denoising and sharpening techniques.
   - Display and save the enhanced face.
5. Continue processing the next frame.

8 Conclusion

This part of the pipeline successfully detects faces from CCTV footage and enhances their quality for better recognition. The use of RetinaFace for detection and OpenCV for image enhancement ensures accurate face detection and improved image clarity, even in challenging scenarios such as side profiles and poor lighting conditions.