ProgressSoft Assignment

Problem 1 – Face Blurring:

Approach

This project aims to blur faces in images while keeping the rest of the image unchanged. To achieve this, we utilized two approaches to detect faces in an image:

The solution consists of two models: A pre-trained model using **Haar Cascade Classifier**, and a trained model from scratch with **YOLO v8**.

The Solution includes multiple files:

- ***FaceBlurring.ipynb***: which is a Jupyter notebook that has the models testing and the evaluation.
- ***YOLO_Training.ipynb***: which is a Jupyter notebook for training the **YOLO** model on a custom dataset, the training process happened on **Google Colab**.
- ***FB_YOLO.py***: which is a Python script to blur the faces of an image and save the blurred image using the trained **YOLO** weights.
- ***FB_CascadeClassifier.py***: a Python script to blur the faces of an image and save the blurred image using the **Haar Cascade Classifier**.
- **Pretrained_models file**: a file that contains the **Haar Cascade** pre-trained model, and the weights for the trained **YOLO model**.
- **Roboflow_data file**: a file that contains custom data for face-to-train the **YOLO** model.
- **Face file**: a file that contains images for faces to test the models.

Haar Cascades (pre-trained model):

I used a pre-trained Haar Cascade model to detect faces in an image. **haarcascade_frontalface_default.xml** classifier is available in the cv2 library for face detection. I implemented this approach in a Python script **FB_CascadeClassifier.py**.

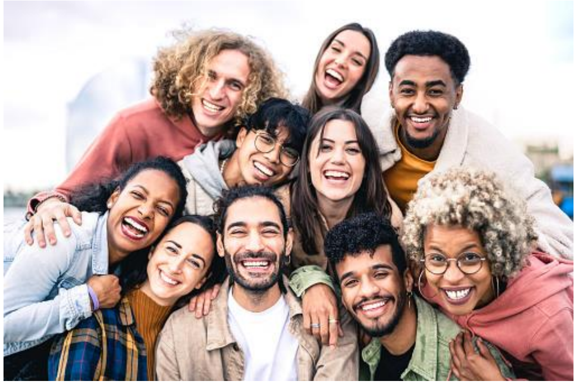

Figure 1 - Before Blurring

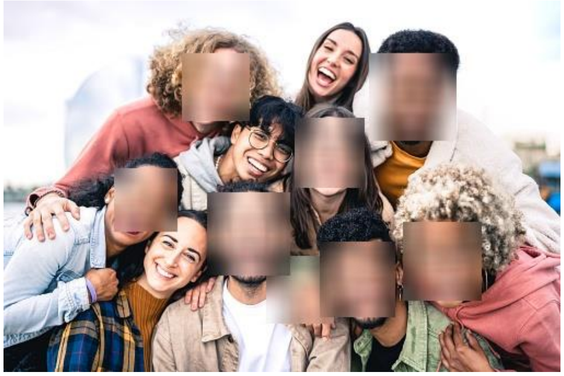

Figure 2 - After Blurring (Cascade Classifier)

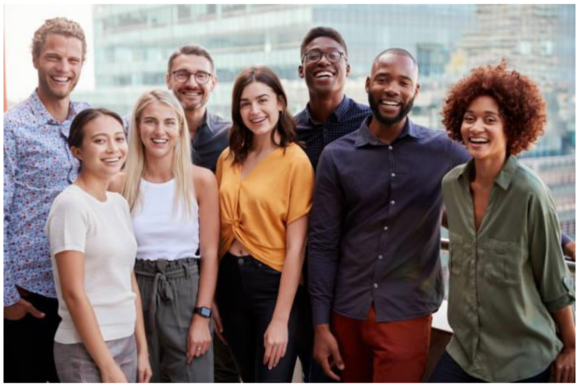

Figure 3 - Before Blurring

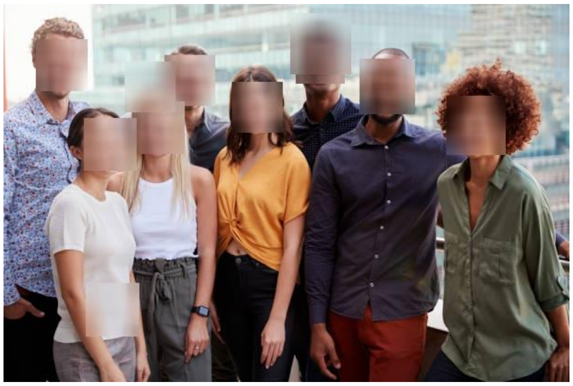

Figure 4 - After Blurring (Cascade Classifier)

As we can see, the **haarcascade_frontalface_defualt** is not accurate for the faces, as you can see when the face has some kind of rotation the model is not performing very well, that's why I build a model from scratch using **YOLOv8** as the second approach.

The link for the **Haar Cascade** pre-trained model is here.

In the **FB_CascadeClassifier.py** file, I have written a function called **blur_face.**
The function **blur_face()** takes an image path and applies Gaussian blur to the faces detected in the image using a pre-trained **Haar cascade classifier**.

YOLO v8 (train the model from scratch):

I build a **YOLO** model and train it from scratch on an annotated dataset from Roboflow the link for it from here. I made the training process on Colab because they have better GPUs. The link for the training notebook is here. (Please note that I connected my Google Drive account the Colab so I can give the model the path for the images directly without the need to upload all the files into Google Colab)

After training the model on Google Colab, I downloaded the trained weights and used them to make detection locally, I applied this on the ***FB_YOLO.py*** file. In this script, I made a function called ***yolo_face_blur***.

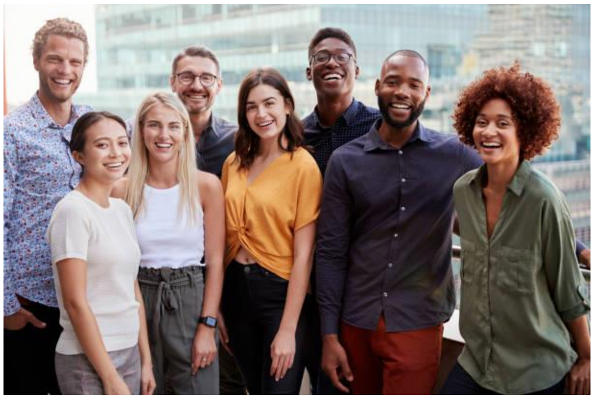

Figure 1 - Before Blurring

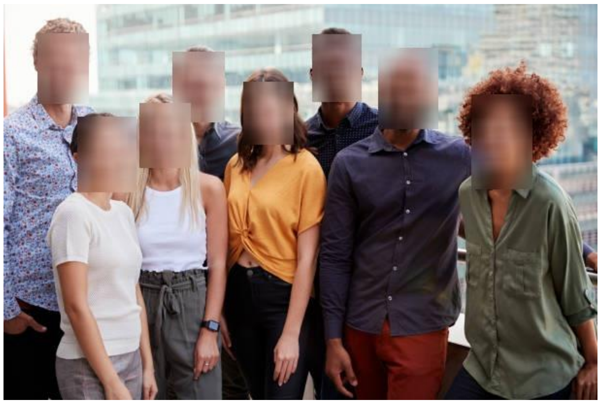

Figure 2 - After Blurring (YOLO)

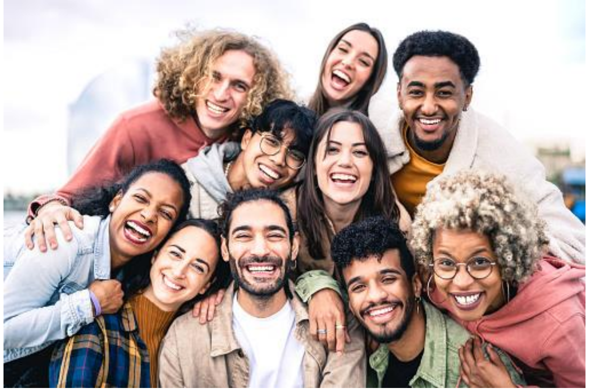

Figure 2 - Before Blurring

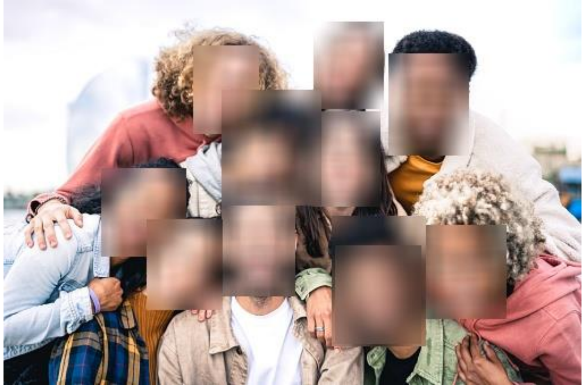

Figure 1 - After Blurring (YOLO)

As we can see, the **YOLO** model performed better than the cascade classifier. This model is version **8** of the YOLO (You Only Look Once) model. It is a powerful algorithm that makes many data augmentation while training to make better performance, and in this task, we use the **nano** model which is the smallest model of the other YOLO v8 models.

Dataset:

The dataset used for training the **YOLO** model is the Face Detection dataset from roboflow. This dataset contains 3 folders for training. It has a train, validation, and test folders for the sets. The training folder contains **2,871** images, the validation folder contains **267** images, and the testing folder contains **145** images. All the images are labeled from the roboflow website. Which gives a total of **3,283** images.

Evaluation:

For the evaluation, I chose the **IoU** (Intersection over Union) metric with precision, and recall. Precision and Recall can be calculated from the IoU.

I used a function called ***calculate_iou*** which has an argument for **bounding box 1** and **bounding box 2**, then calculate the IoU between these two boxes, I tested the IoU between the two models to how similar they are two each other for a specific image, and I got I **0.67 IoU**, which means that the detections for the models are close to each other.

Then, I build an evaluation function to test the model on a test dataset, this function is called ***evaluate_model***, which takes the **ground_truth_path**, the **bb_model_func**, and the **iou_threshold**. This function calculates the IoU between the bounding box detection from the model and the ground truth bounding box for a test dataset, then calculate the precision and the recall from the IoU, and returns the precision, recall, and IoU.

I applied this function for both models, but there is something wrong, I got a very bad IoU even though I saw the detections and they were very well. I went a little deeper and I found the problem. The problem is that the range for the detection bounding box values is different from the ground truth. The ground truth bounding box has values between 0 and 1, but the detection has values much higher than that. I tried to rescale both bounding boxes in many ways but I couldn't solve the problem.

The detections for both models are very accurate, and you can test them yourself, I will tell you how to run the scripts to see for yourself.

Python Version:

The script was developed using Python version **3.9.13**

How to Run the Scripts:

1. Download the Face Detection dataset from roboflow and extract it.

2. Install the required packages by running ***pip install -r requirements.txt***.

3. To blur a specific image using the **Haar cascade classifier** write the following command
   python FB_CascadeClassifier.py < image path >
   The blurred image will be saved as ***cascade_output.jpg***

4. To blur a specific image using the **YOLO** model write the following command
   python FB_YOLO.py < image path >
   The blurred image will be saved as ***Yolo_output.jpg***

Note: I only tested it on Windows.

Future Enhancements:

1. We could explore other face detection models and techniques to improve the accuracy of the module. For example, we could use the MTCNN model, which is known to be robust and accurate for face detection.

2. We could use an adversarial approach to blur the faces, which would improve the privacy of the individuals in the image while preserving the quality of the image.

3. We could explore using deep learning-based methods to detect and blur other sensitive information in the image, such as license plates, personal identification numbers, and so on.