Face Recognition: Methods and Applications

Hasham Asad
Department of Computer Science & IT, University of Lahore, Lahore, Pakistan
Hasham Asad

Abstract

Faces have been the main focus when it comes to recognition for a human being. The machine on the other hand failed to perceive plains like a human mind does and thus it's harder for a machine to recognize a face then on top of that recognize whose face it is. Image processing has been a hardware demanding job for machines and a lengthy and tiring process for humans, but with the recent improvements in hardware and software image recognition is no longer a thing of the past.

The goal of this research is to present a simple and efficient face recognition model that is easier to develop and improve upon.

The most important part of an image recognition system is obtaining the dataset, for this research in question a custom data set of a class containing almost 22 university students is used, to show the ability of the model to be extended into a face recognition based attendance system. Good high definition camera is the basic requirement. The photos must be different in different lighting conditions, different angles of class and different backgrounds.

The model used face recognition is the pre trained Yolov8m object detection model. Which is well documented and easy to train. Altho the model is based on object detection but ignoring the facial features and considering faces as 2 dimensional objects it can perform just as well. The model is trained on 100 epochs and a dataset of 859 png with 22 different faces. The number is achieved by rotating the images 3 sides 90, 180 1nd 270.

A removing duplication algorithm is used to remove duplication in images and furthermore median filter is applied for the removal of noise and histogram is used for contrast equalization.

Keyword: face detection, yolo, image processing

Introduction

Facial recognition technology has become a cornerstone of modern security systems, social media platforms, and user authentication processes. The ability to accurately and efficiently identify individuals based on their facial features holds significant promise for applications ranging from surveillance and access control to personalized user experiences. Recent advancements in deep learning and computer vision have paved the way for more robust and precise facial recognition models. Among these advancements, the YOLO (You Only Look Once) family of models has emerged as a powerful framework for real-time object detection and recognition.

In this research paper, we explore the development and implementation of a facial recognition model based on the YOLOv8 architecture. YOLOv8, the latest iteration in the YOLO series, introduces several enhancements over its predecessors, including improved accuracy, faster processing speeds, and more efficient handling of small and densely packed objects. These characteristics make YOLOv8 particularly well-suited for the task of facial recognition, where precision and speed are paramount.

Our study aims to harness the capabilities of YOLOv8 to create a facial recognition system that can operate effectively in diverse and dynamic environments. We address key challenges such as varying lighting conditions, occlusions, and differences in facial expressions. Additionally, we evaluate the model's performance against existing facial recognition benchmarks to determine its viability for real-world applications.

Through this research, we contribute to the growing body of knowledge in the field of facial recognition by demonstrating how state-of-the-art object detection models like YOLOv8 can be adapted and optimized for identifying human faces with high accuracy and efficiency. This paper presents a comprehensive overview of our methodology, experimental results, and potential future directions for enhancing the model's capabilities.

Literature Review

The literature found for face detection provides a variety of different ideas for the performance, optimization and more accurate predictions, making the model both fast and accurate.
Firstly to improve the dataset, the quality of the images it is suggested to use a median filter in order to reduce noise in an image and use histogram for the contrast balacement of different images.
Viola P and Jones M rectangular suggested a feature detection method which is widely used for its efficiency and effectiveness. According to them there are three kinds of rectangle features;

- Two rectangle features is the difference between the sums of the pixels within two rectangular regions.
- A three rectangle feature computes the sum within two outside rectangles subtracted from the sum in a center rectangle.
- Four rectangles compute the difference between diagonal pairs of rectangles.

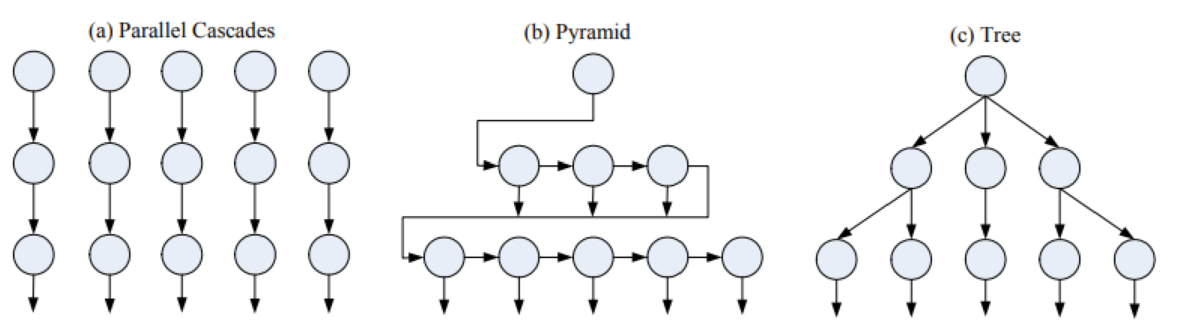
A simple edge detection can be used to differentiate the background d from the face and then the background can be removed forming a smaller image and processing time.

Using a face feature picking algorithm then using that to identify the mood of the person can also be performed.

This research is also based on Viola and Jones framework. They propose to extend the framework by training different cascades individually for each view and then use them as a whole like figure. They propose the following structure for better performance.

WFS Tree-Structure Detector

Pyramid structure adopts coarse-to-fine strategy to handle pose variance



And following algorithm is used

0. (Input) Given a sample $\mathbf{x}$ and the constructed tree detector T.
1. (Initialization) Set the node list L empty; push the root node of T into L; empty the output list O.
2. (WFS procedure)

   While L is not empty, do

   - Pop the first node d from L.
   - Calculate the determinative vector $\mathbf{G}^{(d)}(\mathbf{x})$, where $\mathbf{G}^{(d)}(\mathbf{x}) = \left[g_1^{(d)}(\mathbf{x}), \ldots, g_n^{(d)}(\mathbf{x}) \right]$
   - For t=1,…,n:

     If $g_t^{(d)}(\mathbf{x}) = 1$

       Get the t-th child node s_i of d

       If s_i is a leaf node

         Push l_i, the label of s_i, into the list O.

       Else

         Push s_i into the list L.

       End if

     End if

     End for

   End do
3. (Output) Output all labels in the list O for sample $\mathbf{x}$.

Methodology

The Data:

The pictures were obtained from a wide variety of tours and classes with different lighting conditions, backgrounds and locations. The total number was summed up to be about 214.

Rotating:

These images where rotating to 3 angles for a larger dataset and variety of pictures, using following python code,

```
from PIL import Image
import os

def rotate_and_save(img, save_path_prefix):
    for angle in [0, 90, 180, 270]:
        rotated_img = img.rotate(angle, expand=True)
        save_path = f"{save_path_prefix}_rotated_{angle}_Image.png"
        rotated_img.save(save_path)

save_directory = ''
images_directory = ''

def open_images_in_directory(directory):
    files = os.listdir(directory)
    image_files = [file for file in files if
file.lower().endswith(('.jpg', 'jpeg', '.png'))]

    for image_file in image_files:
        image_path = os.path.join(directory, image_file)
        save_path = os.path.join(save_directory,
os.path.splitext(image_file)[0])
        img = Image.open(image_path)
        rotate_and_save(img, save_path)

open_images_in_directory(images_directory)
```

Removing Duplicates:

```python
import os
from PIL import Image
import imagehash

def find_duplicates(directory):
    hashes = {}
    duplicates = []

    for root, dirs, files in os.walk(directory):
        for file in files:
            if file.endswith(".jpg") or file.endswith(".png"):
                file_path = os.path.join(root, file)
                with open(file_path, 'rb') as f:
                    try:
                        image = Image.open(f)
                        hash_value = str(imagehash.average_hash(image))
                        if hash_value in hashes:
                            duplicates.append(file_path)
                        else:
                            hashes[hash_value] = file_path
                    except Exception as e:
                        print(f"Error processing {file_path}: {e}")

    return duplicates

def delete_duplicates(duplicates):
    for duplicate in duplicates:
        os.remove(duplicate)
        print(f"{duplicate} Deleted")

if __name__ == '__main__':
    directory = ''
    duplicates = find_duplicates(directory)
    if duplicates:
        delete_duplicates(duplicates)
    else:
        print("No Duplicates")
```

File Conversion:

The pictures shot from an iphone are transferred with the extension of heic while other pictures include jpg and jpeg so to uniform the extension they are al converted into png.

```
from PIL import Image
import os
from pillow_heif import register_heif_opener

def convert_heic_to_png(heic_path, png_path):
    register_heif_opener()
    try:
        with Image.open(heic_path) as heic_img:
            heic_img.convert('RGB').save(png_path, formate='PNG')
    except Exception as e:
        print(e)

heic_directory = ""

png_directory = ""

img_path = ""

for file in os.listdir(img_path):
    if file.endswith('.HEIC') or file.endswith(('.heic', 'jpg', 'jpeg')):
        heic_path = os.path.join(heic_directory, file)
        png_path = os.path.join(png_directory, os.path.splitext(file)[0] +
".png")
        convert_heic_to_png(heic_path, png_path)
```

Numbering

All the images are numbered for the ease of png file aligning with txt files.

```
import os
import glob

def rename_img_files(directory):
    os.chdir(directory)
    png_files = glob.glob('*.png')
    png_files.sort()
    for index, filename in enumerate(png_files):
        new_name = f"{index + 1}.png"
        os.rename(filename, new_name)
        print(f"Renamed {filename} to {new_name}")

img_directory_path = ''
rename_img_files(img_directory_path)
```

Dependencies:

Following dependencies are required when training the yolo model

- Pip
- Python
- Conda (env management)
- ultralytics

Training:

Yolov8 can be directly trained from the command line by passing following parameters

yolo task=detect mode=train model=yolov8m.pt data=data_custom.yaml epoch=100 imgsz=640

Or you can right the following python code

from ultralytics import YOLO

mode = YOLO("yolov8m.pt")

results = mode.train(data="/kaggle/input/yaml-file-for-yolo-training/data_custom.yaml", epochs=100, imgsz=640)

Yolov8m is the name of pretrained model, where m stands for medium and other types include yolov8n and yolov8s

Yaml file is used, with the following structure:

```
train: <path_of_train_folder>
val: <path_of_val_folder>

nc: <number_of_classes>

names: [
“Array of classes”
]
```

In my case it looked as as shown below

```yaml
train: /kaggle/input/850-label-png-yolo-22-classesfacesdetection/train
val: /kaggle/input/val-data-for-yolo-4-png-with-labels/val

nc: 27

names: [
    "Muhammad Abdullah",
    "Raheeb Gill",
    "Hasham Asad",
    "Muaz Asim",
    "Hamza Khalid",
    "Bilal Munir",
    "Ali Ahmed",
    "Mustafa Raja",
    "Bilal Ch",
    "Hasham Mukhtar",
    "Abdullah Arshad",
    "Ghulam Mujtaba",
    "Zaid Atif",
    "Sahar Arif",
    "Anmol Nisar",
    "Sahil Kumar",
    "Umar Niazi",
    "Jannat Sameer",
    "Faizan Rasul",
    "Bakhtawar Asad",
    "Rumaiha",
    "Rizwan Ghuri",
    "Rizwan Ghuriw",
    "G",
    "Mustafa RajaW",
    "Hasham AsadW",
    "Hamza KhalidW"
]
```

Performance:

Start

Loads all the images.

```
Transferred 469/475 items from pretrained weights
TensorBoard: Start with 'tensorboard --logdir runs/detect/train2', view at http://localhost:6006/
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
  ········································
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

wandb version 0.17.0 is available! To upgrade, please run: $ pip install wandb --upgrade

Tracking run with wandb version 0.16.6

Run data is saved locally in /kaggle/working/wandb/run-20240521_235930-5tapmdi2

Syncing run **train2** to Weights & Biases (docs)

View project at https://wandb.ai/karbonion/YOLOv8

View run at https://wandb.ai/karbonion/YOLOv8/runs/5tapmdi2

```
Freezing layer 'model.22.dfl.conv.weight'
AMP: running Automatic Mixed Precision (AMP) checks with YOLOv8n...
Downloading https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov8n.pt to 'yolov8n.pt'...
100%|██████████| 6.23M/6.23M [00:00<00:00, 72.6MB/s]
AMP: checks passed ✅
train: Scanning /kaggle/input/850-label-png-yolo-22-classesfacesdetection/train/labels... 469 images, 0 backgrounds, 0 corrup
t:  55%|█████     | 469/859 [00:12<00:17, 22.51it/s]
```

Starts with the cls loss of 3.517 box loss of 1.646 and dfl loss 1.32

```
train: Scanning /kaggle/input/850-label-png-yolo-22-classesfacesdetection/train/labels... 859 images, 0 backgrounds, 0 corrup
t: 100%|██████████| 859/859 [00:24<00:00, 35.16it/s]
train: WARNING ⚠ Cache directory /kaggle/input/850-label-png-yolo-22-classesfacesdetection/train is not writeable, cache not s
aved.
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01), CLAHE(p=0.01, clip_lim
it=(1, 4.0), tile_grid_size=(8, 8))
val: Scanning /kaggle/input/val-data-for-yolo-4-png-with-labels/val/labels... 4 images, 0 backgrounds, 0 corrupt: 100%|███████
███| 4/4 [00:00<00:00, 12.66it/s]
val: WARNING ⚠ Cache directory /kaggle/input/val-data-for-yolo-4-png-with-labels/val is not writeable, cache not saved.

Plotting labels to runs/detect/train2/labels.jpg...
optimizer: 'optimizer=auto' found, ignoring 'lr0=0.01' and 'momentum=0.937' and determining best 'optimizer', 'lr0' and 'momen
tum' automatically...
optimizer: AdamW(lr=0.000323, momentum=0.9) with parameter groups 77 weight(decay=0.0), 84 weight(decay=0.0005), 83 bias(decay
=0.0)
TensorBoard: model graph visualization added ✅
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/detect/train2
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      1/100      7.32G      1.646      3.517       1.32        152        640: 100%|██████████| 54/54 [01:54<00:00,  2.11s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:02<00:00,
2.40s/it]
                   all          4         62      0.117      0.391      0.124     0.0842

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      2/100      7.25G      1.276      2.212      1.096        174        640: 100%|██████████| 54/54 [01:51<00:00,  2.07s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 1
1.31it/s]
                   all          4         62      0.453      0.404      0.497      0.376

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      3/100      7.11G       1.24      1.877      1.102        222        640:  56%|█████     | 30/54 [00:56<00:48,  2.00s/it]
```

On 27 epochs trained the loss drops to 0.509 for cls, 0.9817 for box and 1.009 for dfl

```
      21/100      7.2G      1.033     0.5678      1.032        157        640: 100%|██████████| 54/54 [01:50<00:00,  2.04s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 11.67it/s]
                   all          4         62      0.942      0.974      0.991      0.765

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      22/100     7.21G      1.024     0.5489      1.021        142        640: 100%|██████████| 54/54 [01:48<00:00,  2.01s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 13.35it/s]
                   all          4         62      0.909      0.991      0.995      0.786

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      23/100     7.27G      1.001     0.5379      1.021        147        640: 100%|██████████| 54/54 [01:51<00:00,  2.07s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 13.25it/s]
                   all          4         62      0.949       0.94      0.995      0.767

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      24/100      7.2G      1.008     0.5353      1.021        242        640: 100%|██████████| 54/54 [01:46<00:00,  1.97s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 12.95it/s]
                   all          4         62      0.929      0.973      0.995      0.763

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      25/100     7.18G     0.9912     0.5313      1.017        227        640: 100%|██████████| 54/54 [01:52<00:00,  2.09s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 12.84it/s]
                   all          4         62      0.934      0.955      0.988      0.751

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      26/100     7.24G     0.9927     0.5318      1.021         96        640: 100%|██████████| 54/54 [01:53<00:00,  2.10s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 11.26it/s]
                   all          4         62      0.896      0.977      0.984      0.761

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      27/100     7.26G     0.9817      0.509      1.009        337        640:  94%|█████████▍| 51/54 [01:44<00:08,  2.73s/it]
```

Half way through the cls loss is 0.4144, box loss is 0.8432 and dfl loss 0.9637

```
      44/100     7.22G     0.8773     0.4332     0.9738        188        640: 100%|██████████| 54/54 [01:46<00:00,  1.98s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 12.86it/s]
                   all          4         62       0.96      0.925      0.995      0.833

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      45/100     7.21G     0.8651     0.4285     0.9629        167        640: 100%|██████████| 54/54 [01:56<00:00,  2.15s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 12.86it/s]
                   all          4         62       0.96      0.901      0.993      0.832

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      46/100      7.2G     0.8741     0.4264     0.9663        202        640: 100%|██████████| 54/54 [01:54<00:00,  2.12s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 12.84it/s]
                   all          4         62      0.927      0.991      0.993      0.835

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      47/100     7.21G     0.8522     0.4187     0.9562        181        640: 100%|██████████| 54/54 [01:51<00:00,  2.06s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 13.21it/s]
                   all          4         62       0.93      0.959      0.988      0.826

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      48/100     7.25G     0.8643     0.4246     0.9628        172        640: 100%|██████████| 54/54 [01:50<00:00,  2.05s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  7.98it/s]
                   all          4         62      0.921      0.969      0.993      0.846
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      49/100     7.18G     0.8519     0.4158      0.954        165        640: 100%|██████████| 54/54 [01:51<00:00,  2.06s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 13.30it/s]
                   all          4         62      0.904       0.97      0.993      0.833

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      50/100     7.18G     0.8438     0.4144     0.9637        270        640:  13%|█▎        | 7/54 [00:09<00:59,  1.26s/it]
```

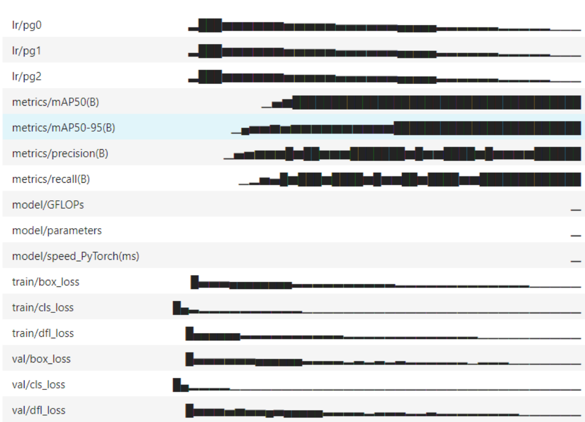
At the last run the losses are as fallows box 0.5968 , cls 0.2572 an dfl 0.8815

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     95/100      7.17G     0.6102     0.2665     0.8879         91        640: 100%|██████████| 54/54 [01:52<00:00,  2.09s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 12.99it/s]
                   all          4         62      0.957      0.959      0.993      0.874

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     96/100      7.18G     0.6058     0.2636     0.8902         95        640: 100%|██████████| 54/54 [01:56<00:00,  2.15s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 13.66it/s]
                   all          4         62      0.958      0.959      0.993      0.873

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     97/100      7.17G     0.6057     0.2664     0.8859        108        640: 100%|██████████| 54/54 [01:51<00:00,  2.07s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 10.21it/s]
                   all          4         62      0.943      0.969      0.993      0.872

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     98/100       7.2G     0.5936     0.2603     0.8798         86        640: 100%|██████████| 54/54 [01:52<00:00,  2.09s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 12.52it/s]
                   all          4         62      0.955      0.965      0.993      0.874

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100      7.17G     0.5976     0.2632     0.8802        134        640: 100%|██████████| 54/54 [01:55<00:00,  2.14s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 13.08it/s]
                   all          4         62      0.953      0.965      0.993      0.872

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100      7.18G     0.5908     0.2572     0.8815         91        640: 100%|██████████| 54/54 [01:51<00:00,  2.06s/it]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00, 11.87it/s]
                   all          4         62      0.952      0.964      0.993      0.867
```

The model was fully trained with up to 94% accuracy.

Results saved to runs/detect/train2

Run history:



Run summary:

lr/pg0	1e-05
lr/pg1	1e-05
lr/pg2	1e-05
metrics/mAP50(B)	0.995
metrics/mAP50-95(B)	0.8832
metrics/precision(B)	0.94073
metrics/recall(B)	0.98763
model/GFLOPs	79.149
model/parameters	25871953
model/speed_PyTorch(ms)	106.903
train/box_loss	0.59084
train/cls_loss	0.25718
train/dfl_loss	0.8815
val/box_loss	0.58056
val/cls_loss	0.2774
val/dfl_loss	0.87203

```
100 epochs completed in 3.191 hours.
Optimizer stripped from runs/detect/train2/weights/last.pt, 52.1MB
Optimizer stripped from runs/detect/train2/weights/best.pt, 52.1MB

Validating runs/detect/train2/weights/best.pt...
Ultralytics YOLOv8.2.19 🚀 Python-3.10.13 torch-2.1.2 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 218 layers, 25855393 parameters, 0 gradients, 78.8 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  8.73it/s]
                   all          4         62      0.941      0.988      0.995      0.883
     Muhammad Abdullah          4          4      0.954          1      0.995      0.894
           Raheeb Gill          4          4      0.954          1      0.995      0.908
           Hasham Asad          4          3      0.958          1      0.995      0.846
             Muaz Asim          4          4      0.975          1      0.995      0.911
          Hamza Khalid          4          2      0.952          1      0.995      0.808
           Bilal Munir          4          4      0.904          1      0.995       0.87
             Ali Ahmed          4          4          1      0.916      0.995      0.874
          Mustafa Raja          4          4      0.954          1      0.995      0.925
              Bilal Ch          4          1      0.903          1      0.995      0.995
        Hasham Mukhtar          4          1      0.868          1      0.995      0.796
       Abdullah Arshad          4          1      0.965          1      0.995      0.995
        Ghulam Mujtaba          4          4          1      0.852      0.995      0.846
             Zaid Atif          4          4          1      0.985      0.995      0.834
            Sahar Arif          4          4      0.903          1      0.995      0.869
           Anmol Nisar          4          3      0.935          1      0.995      0.844
           Sahil Kumar          4          4      0.943          1      0.995       0.85
         Jannat Sameer          4          3      0.915          1      0.995      0.895
          Faizan Rasul          4          4      0.917          1      0.995      0.865
               Rumaiha          4          1      0.866          1      0.995      0.895
          Rizwan Ghuri          4          3      0.949          1      0.995      0.942
Speed: 0.3ms preprocess, 15.5ms inference, 0.0ms loss, 1.2ms postprocess per image
Results saved to runs/detect/train2
```

After the training the folder contains following files

data_custom.yaml
runs
train
val
yolov8m.pt

Weights contains the best.pt model which act as a custom trained model and can use to predict faces on the given img.

Prediction:

yolo task=detect mode=predict model=path_to_best.pt show=True conf=0.5 source=filepath.png

Or write a python script passing the same parameters.

The predicted image will be stored inside run/detect/predict
Some of the examples are

One with a little less people

And 1 in a class setting and inverted

References:

- https://www.researchgate.net/profile/Anila-Satish/publication/225292501_Simple_and_Fast_Face_Detection_System_Based_on_Edges/links/09e414fd75a23d2c1b000000/Simple-and-Fast-Face-Detection-System-Based-on-Edges.pdf
- Awais-Jumani/publication/Face_Detection_and_Recognition_System_for_Enhancing_Security_Measures_Using_Artificial_Intelligence_System
- ieeexplore.ieee.org/stamp/stamp.jsp
- Ibrahim-Ali-Mohammed/An-Exploratory-Study-Into-The-Face-Detection-And-Recognition-System-To-Strengthen-Security-Precautions-Using-An-Artificial-Intelligence-System
- https://rogerioferis.com/ClassMarch10/HomeworkVectorBoosting