Faculty of Informatics and Computer Science

Artificial Intelligence

On AI-Driven Privacy Protection: A Study on AI-based Solutions using Face Blurring Techniques

By: **Sara Amjad Sabih Haidar Anabtawi**

Supervised By

Doctor Amr Ghoneim

June 2024

Abstract

Nowadays, in the era of digitization and images, privacy has become a highly concerning issue. The enormous amount of visible content over the internet has further raised privacy issues; one is individual privacy. This project contributes to that problem by relating to this issue. It uses artificial intelligence but is mainly based on deep learning for the automatic detection and recognition of faces found in multimedia content for blurring. This project will blur people's face in multimedia content to protect their identity.

While I experimented with several models in this project—for example, YOLOv5 and YOLOv8 for the face detection task, and YOLOv5, YOLOv8, and MobileNet for facial recognition—I also experimented with several individual blurring methodologies, which I then combined in pairs and quartets to iterate each manner of obscuring the face five times, after which this great deal of experimentation brought the development of a system whereby no model could recognize any of the blurred faces.

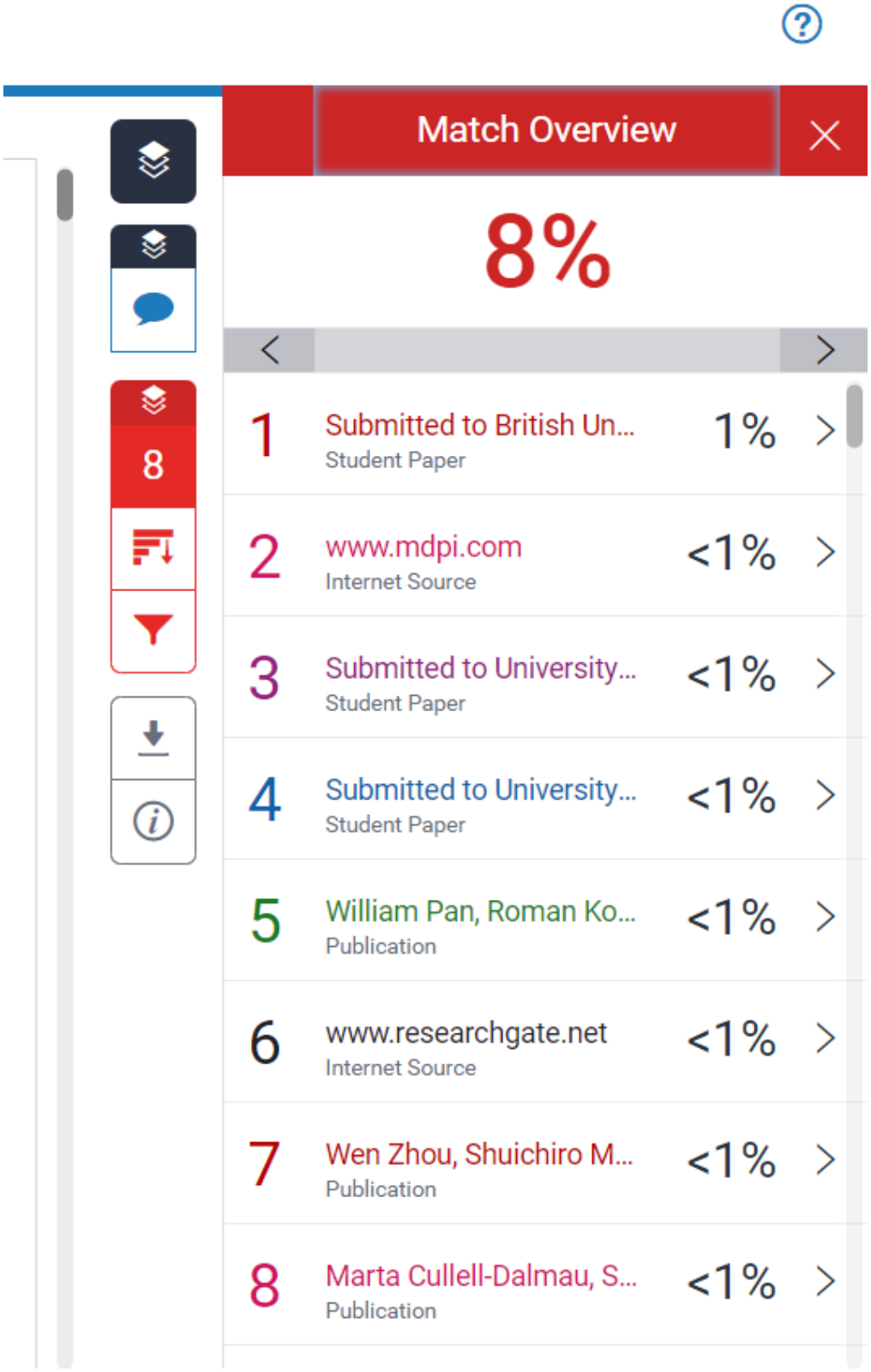
On the flip side, the datasets limited me. I had to work with the LFW and Avengers datasets, which were both full of challenges because of imbalances and a lack of labelled data per class. In particular, the face-recognition datasets generally have imbalanced courses with too few images per class; it will be necessary to use the LFW dataset for face detection while adding additional photos from the Avengers dataset from celebrity datasets. From there, these extra images had no bounding box coordinates, so I had to retrain the YOLO model for face detection on the LFW dataset and then predict and save bounding box coordinates for the Avengers dataset. Despite that effort, my attempts were futile because the Avengers dataset only had cropped face images, which would not allow the model to recognize faces from a distance.

In another result, YOLOv5 attained a precision of 0.982, recall of 0.965, mAP50 of 0.991, and mAP50-95 of 0.849, whereas YOLOv8 has also shown almost the same performance by giving the value of precision of 0.983, recall of 0.961, mAP50 of 0.99, and mAP50-95 of 0.839 for the same parameters in this category. For face recognition, the results were different across the various characters, with MobileNet generally outperforming the.

I would very much like to collect a more varied datasets for future work, including many classes. Such an extended dataset will boost performance in face detection and recognition but, more importantly, allow us to detect sensitive types of objects that may harm individual privacy, such as ATM details, plate numbers, street numbers, name tags, etc. The improvement will integrate the model with further abilities for privacy protection under various conditions. Increasing the diversity and volume of data will considerably raise the robustness and accuracy of the model across varying contexts. The expansion of detectable objects will further make the model versatile and thus very practical in real-world applications to ensure comprehensive privacy, which is reliable to persons who rightfully deserve their privacy.

In summary, this graduation project will aim to use the advancement in artificial intelligence for automatic face blurring, which happens to be a unique solution and innovative approach toward protecting personal privacy. In other words, the idea behind this project is that AI-based technologies should be implemented for enhanced privacy to share multimedia content ethically. It has the potential to establish the basis for an online environment that is safer and more secure.

Turnitin Report



Acknowledge

I would also like to offer my gratitude to my supervisor, Doctor Amr Ghoneim, for his priceless supervision, moral support, and constant encouragement during the period of this graduation project. His insight and comments have been critical in steering challenges and sharpening the approach to this research. I'm genuinely grateful for the time and effort he committed to mentoring and providing me with constructive advice at every project stage.

Also, I will express my respect to Doctor Nahla Barakat, the leader of my AI module. Her teaching has led me to know what AI is, and she was the key person to help me get the essential core of AI-related concepts and techniques. Her time devoted to teaching was one of the factors that significantly led to making this work a success. Also, I appreciate these lovely TAs: Shehab, Amgad, and Ahmed Tamaa, for their great help.

Finally, I thank my family and friends for their continual support and encouragement throughout my academic journey. The belief they had in me and the support given encouraged me to complete this project successfully.

Table of Contents

Abstract 2
Turnitin Report 3
Acknowledge 4
Table of Contents 5
List of Figures 8
List of Tables 12
1 Introduction 13
1.1 Overview 13
1.2 Problem Statement 13
1.3 Scope and Objectives 14
1.3.1 The three main components of the AI tool include the following: 14
1.3.2 System Workflow 14
1.3.3 Project Objective 15
1.3.4 In Scope 15
1.3.5 Out of Scope 15
1.3.6 Targeted Users 15
1.4 Report Organization (Structure) 15
1.5 Work Methodology 17
1.6 Work Plan (Gantt chart) 18
2 Related Work (State-of-The-Art) 19
2.1 Background 19
General Approach 20
2.2 Literature Survey 20
2.3 Analysis of the Related Work 40
3 Proposed solution 42
3.1 Solution Methodology 42
3.1.1 Capabilities of YOLO in this project are summarized as follows: 42
3.1.2 Functional/ Non-functional Requirements 43
3.2 Design / Simulation set up 44
3.2.1 The Full Generic Project Pipeline: 44
3.2.2 The YOLOV8 For Face Detection and Recognition Project Pipeline (Pipeline 1): 46
3.2.3 The YOLOV5 For Face Detection and Recognition Project Pipeline (Pipeline 2): 48
3.2.4 The YOLOV5 For Face Detection and MobileNet for Face Recognition Project Pipeline (Pipeline 3): ..50

3.2.5 The YOLOV8 For Face Detection and MobileNet for Face Recognition Project Pipeline (Pipeline 4): ..52
4 Implementation 54
4.1 Pre-Processing: 54
4.1.1 YoloV5/YOLOV8 as a Face Detection Model 54
4.1.2 YoloV5 as a Face Recognition Model: 55
6.1.1 YoloV8 as a Face Recognition Model: 63
6.1.2 Mobile Net for Face Recognition: 71
7 Blurring Techniques: 73
7.1 GrabCut/Gaussian 73
7.2 Gaussian Blurring 73
7.3 Median Blurring 74
7.4 Box Filter Blurring 74
7.5 Mean Shift Blurring 75
7.6 Distance-Based Blurring 76
7.7 Radial Blur 76
Motion Blur 77
Testing and evaluation 78
7.8 Testing 78
7.8.1 YoloV8 for Face Detection 78
7.8.2 Yolov5 for Face Detection 79
7.8.3 MobileNet for Face Recognition 80
7.8.4 YOLOv5 for Face Recognition 83
7.8.5 YOLOV8 for Face Recognition 86
7.8.6 Blurring Techniques 89
....92
....93
7.9 Evaluation 94
7.9.1 YOLOV8 as a Face Detection Model: 94
7.9.2 YOLOV5 as a Face Detection Model: 105
7.9.3 MobileNet as a Face Recognition Model: 113
7.9.4 YOLOV5 as a Face Recognition Model: 119
7.9.5 YOLOV8 as a Face Recognition Model: 127
8 Results 136
8.1 Functions Used by Yolov8/v5 Models: 136
8.2 Functions Used by Yolov8/v5 Models Specifically for Images with Multiple Faces: 138

8.3 Functions Used when the Yolov5/v8 is used as face detection model and MobileNet Model as a face recognition model: 139
8.4 The YOLOV8 For Face Detection and Recognition Project Pipeline (Pipeline 1): 141
8.4.1 Results of an image with a Single Face: 141
8.4.2 Multiple Faces Per Image 146
8.5 The YOLOV5 For Face Detection and Recognition Project Pipeline (Pipeline 2): 150
8.5.1 Results of an image with a Single Face: 150
8.5.2 Multiple Faces Per Image 154
8.6 The YOLOV8 For Face Detection and MobileNet for Face Recognition Project Pipeline (Pipeline 3): 158
8.6.1 Results of an image with a Single Face: 158
8.6.2 Multiple Faces Per Image 163
8.7 The YOLOV5 For Face Detection and MobileNet for Face Recognition Project Pipeline (Pipeline 4): 168
8.7.1 Results of an image with a Single Face: 168
8.7.2 Multiple Faces Per Image 173
9 Discussion 178
9.1 Face Detection Models Results: 178
9.2 Face Recognition Models Results: 179
9.3 Blurring Techniques Results: 181
10 Challenges and Limitations 182
11 Trials 182
12 Conclusions and Future Work 184
12.1 Summary 184
12.2 Future Work 185
References 186

List of Figures

Figure 1 Gantt Chart 18
Figure 2 Confusion Matrix Results 29
Figure 3 Comparison of detection rate 29
Figure 4 Calculation of confusion matrix 29
Figure 5 sample of testing.... **Error! Bookmark not defined.**
Figure 6 Block diagram of proposed real-time face recognition system 32
Figure 7 Samples of AT&T database 33
Figure 8 Comparison of Face Recognition Results 34
Figure 9 Snapshots of results obtained using the real-time system 35
Figure 10 The Generic Project Pipeline 44
Figure 11 The Yolov8 Pipeline 46
Figure 12 The YoloV5 Pipeline 48
Figure 13 The Yolov5/MobileNet Pipeline 50
Figure 14 The Yolov8/MobileNet Pipeline 52
Figure 15 Loading and downloading the dataset code snippet 54
Figure 16 Yolo Configuration File 55
Figure 17 Detect_Faces Function 56
Figure 18 Save_bounding_boxes function 57
Figure 19 Process_images function snippet 58
Figure 20 Split_dataset Function 59
Figure 21 part of the splitting dataset function 59
Figure 22 part2 of splitting dataset function 60
Figure 23 move_files function snippet 60
Figure 24 moving files and converting labels snippet 61
Figure 25 define and create training and val directories snippet 61
Figure 26 running the split_dataset function snippet 62
Figure 27 create the yaml file snippet 62
Figure 28 verify_dataset_structure function snippet 63
Figure 29 yolov8 detect_faces function snippet 65
Figure 30 yolov8 save_bounding_boxes function snippet 66
Figure 31 yolov8 process_images function snippet 66
Figure 32 yolov8 split_dataset function snippet 67
Figure 33 yolov8 split_dataset part2 function snippet 68
Figure 34 yolov8 split images and val sets snippet p3 68
Figure 35 yolov8 move_files function snippet p4 69
Figure 36 yolov8 move files and convert labels snippet p5 69
Figure 37 yolov8 defining and creating training and val directories p6 70
Figure 38 running split dataset function snippet p7 70
Figure 39 create yaml file 71
Figure 40 verify_dataset_structure function snippet(yolov8) 71
Figure 41 label encoding snippet (mobileNet) 72
Figure 42 train-test split snippet 72
Figure 43 grabcut(gaussian) function 73
Figure 44 gaussian blur function 74
Figure 45 Median Blur Function 74

Figure 46 Box Filter Function 75
Figure 47 MeanShift Filtering Function 75
Figure 48 Distance-Based Function....76
Figure 49 Radial Blur function....77
Figure 50 Motion Blur Function 78
Figure 51 Yolov8 training snippet 78
Figure 52 yolov8 labelled testing batch 78
Figure 53 yolov8 predicted testing batch 79
Figure 54 Yolov5 training snippet 79
Figure 55 yolov5 Labeled testing batch 79
Figure 56 yolov5 predicted testing batch 80
Figure 57 mobilenet predict snippet 80
Figure 58 Mapping true and predicted Labels to Class Names 80
Figure 59 Code for counting correct and Incorrect Predictions 81
Figure 60 Output of Model Predictions: Comparison of predicted and actual Labels 82
Figure 61 Total number of correct and incorrect prediction....82
Figure 62 load and preprocess function 82
Figure 63 predict image class function 83
Figure 64 predicted class of a test image 83
Figure 65 Organizing and Running YOLOv5 predictions on testing Images....83
Figure 66 sample of predictions of testing images (YOLOv5) 84
Figure 67 sample of test predictions (YOLOv5) 85
Figure 68 another sample of test predictions (YOLOv5)....85
Figure 69 Organizing and Running YOLOv8 predictions on testing Images....86
Figure 70 sample of predictions of testing images (YOLOv8) 86
Figure 71 testing predictions samples(yolov8) 88
Figure 72 another testing predictions samples(yolov8) 89
Figure 73 Before Any Blurring Techniques: Correct and Incorrect Predictions of several classes 89
Figure 74 original image before any blurring 89
Figure 75 After Guassian Blur: Correct and Incorrect Predictions of several classes 89
Figure 76 image after gaussian blur....90
Figure 77 After median Blur: Correct and Incorrect Predictions of several classes 90
Figure 78 image after median blur 90
Figure 79 After motion Blur: Correct and Incorrect Predictions of several classes....90
Figure 80 image after motion blur....91
Figure 81 After Grabcut(gaussian) Blur: Correct and Incorrect Predictions of several classes 91
Figure 82 image after grabcut(gaussian) blur....91
Figure 83 image after meanshift filtering 91
Figure 84 After MeanShift Filtering Blur: Correct and Incorrect Predictions of several classes....92
Figure 85 After Box filter Blur: Correct and Incorrect Predictions of several classes....92
Figure 86 image after box filter 92
Figure 87 After Radial Blur: Correct and Incorrect Predictions of several classes....92
Figure 88 image after radial blur 93
Figure 89 After Motion and meanshift filtering Blur: Correct and Incorrect Predictions of several classes 93
Figure 90 After radial and gaussian Blur: Correct and Incorrect Predictions of several classes....93
Figure 91 After Motion, radial, gaussian and meanshift filtering Blur: Correct and Incorrect Predictions of several classes 94
Figure 92 After Motion, radial, gaussian and meanshift filtering Blur: Correct and Incorrect Predictions of several classes (5 iterations each)....94

Figure 93 yolov8 training code 95
Figure 94 yolov8 epochs results.....96
Figure 95 confusion matrix of yolov8 for face detection.....97
Figure 96 F1-confidence curve of yolov8 for face detection 98
Figure 97 precision-confidence curve of yolov8 for face detection 99
Figure 98 recall - confidence curve of yolov8 for face detection 100
Figure 99 results curves of yolov8 for face detection.....101
Figure 100 yolov8 validation sample batch 1 labels 102
Figure 101 yolov8 validation sample batch 1 predictions 103
Figure 102 yolov8 validation sample batch 2 labels 104
Figure 103 yolov8 validation sample batch 1 predictions 105
Figure 104 training of yolov5 for face detection 105
Figure 105 yolov5 epochs results.....108
Figure 106 yolov5 confusion matrix for face detection.....109
Figure 107 yolov5 F1-confidence curve for face detection 110
Figure 108 yolov5 precision-confidence curve for face detection 111
Figure 109 yolov5 recall-confidence curve for face detection 112
Figure 110 yolov5 Results curves for face detection 113
Figure 111 load the mobilenet snippet.....114
Figure 112 adding new top layers.....114
Figure 113 model compilation snippet 114
Figure 114 training execution snippet 115
Figure 115 training logs of epochs.....116
Figure 116 classification report of mobilenet.....117
Figure 117 confusion matrix of mobilenet 117
Figure 118 learning curve of mobileNet 118
Figure 119 yolov5 for face recognition training.....119
Figure 120 yolov5 epochs log 121
Figure 121 confusion matrix of yolov5 for face recognition.....123
Figure 122 yolov5 F1-confidence curve for face recognition 124
Figure 123 yolov5 precision-confidence curve for face recognition 125
Figure 124 yolov5 recall-confidence curve for face recognition 126
Figure 125 validation sample batch prediction (yolov5 for face recognition).....127
Figure 126 yolov8 for face recognition training.....127
Figure 127 Figure 120 yolov8 for face recognition epochs log.....130
Figure 128 confusion matrix of yolov8 for face recognition.....132
Figure 129 yolov8 F1-confidence curve for face recognition 132
Figure 130 yolov8 precision-confidence curve for face recognition 133
Figure 131 yolov5 recall-confidence curve for face recognition 134
Figure 132 validation sample batch prediction (yolov8 for face recognition).....135
Figure 133 load model function.....136
Figure 134 preprocess image function 136
Figure 135 predict function 136
Figure 136 draw bounding boxes function 136
Figure 137 display image function.....137
Figure 138 display cropped image function.....137
Figure 139 blur faces function 137
Figure 140 save image function.....138
Figure 141 crop multiple bounding boxes 138

Figure 142 blur multiple faces 138
Figure 143 replace blurred faces function 139
Figure 144 Mobilenet imports 139
Figure 145 Mobilenet predict image class and resize function 139
Figure 146 blur face function 139
Figure 147 crop bounding box (mobilenet) 140
Figure 148 predict image classes function 140
Figure 149 crop bounding boxes(mobilenet) 140
Figure 150 class to name function 141
Figure 151 load face detection and recognition models 141
Figure 152 define paths 141
Figure 153 load model 141
Figure 154 preprocess image 142
Figure 155 prediction 142
Figure 156 prediction results 142
Figure 157 draw bounding box 142
Figure 158 display image with bounding box 142
Figure 159 image 143
Figure 160 crop bounding box 143
Figure 161 cropped image 143
Figure 162 define paths 144
Figure 163 load path 144
Figure 164 preprocess image 144
Figure 165 predict 144
Figure 166 prediction result 144
Figure 167 blur if thor 144
Figure 168 blurred face 145
Figure 169 save blurred image 145
Figure 170 face detection prediction results 147
Figure 171 image with bounding boxes drawn 147
Figure 172 cropped images 148
Figure 173 final image after blurring thor 149
Figure 174 image with drawn bbox 151
Figure 175 cropped image 151
Figure 176 blurred cropped face 152
Figure 177 full image after blurring 152
Figure 178 blurred cropped face 154
Figure 179 precition of face recognition model after blurring 154
Figure 180 image with draw bboxes 155
Figure 181 cropped faces 156
Figure 182 full image after blurring 157
Figure 183 full image with bbox drawn 159
Figure 184 cropped image 159
Figure 185 blurring cropped face then replacing in original image 162
Figure 186 full image with drawn bboxes 164
Figure 187 cropped faces image 165
Figure 188 recognizing cropped faces and blur if thor 166
Figure 189 full image after blurring 167
Figure 190 image with drawn bbox 169

Figure 191 cropped face 169
Figure 192 recognizing cropped face and if thor blur and replace in original image 172
Figure 193 full image with bboxes drawn.....174
Figure 194 cropped faces.....175
Figure 195 recognize cropped faces and if thor blur 176
Figure 196 final image after blurring 177

List of Tables

Table 1 Face Detection Models Results 178
Table 2 Face Recognition Models Results.....179
Table 3 Blurring Techniques Results 181

1 Introduction

1.1 Overview

The number of images circulating on the internet in this digital age increases numberless concerns about privacy. Many publishing services make it very easy for people to publish images without permission, especially on social networking services, giving rise to risks like identity theft, cyberbullying, and damage to one's reputation. The primary aim of this project is to apply artificial intelligence in automatically blurring faces on images to address such risks that could potentially cause privacy infringement. This would allow the user to secretly hide information that can expose their identity to undesired eyes and take more control over their online persona.

The project would try to ensure that an efficient AI system is delivered for robust facial detection and blurring in images, making the faces unrecognizable by the advanced models of AI recognition systems, for example, YOLO and MobileNet. It may include applying multiple blur techniques to the recognized faces. The blurring will be applied to those who do not consent for their images to be used; therefore, ethical compliance is followed. Even with the blurring, the quality of the image remains the same, thereby maintaining the integrity of the photos in aesthetic quality but providing privacy given that the move is supposed to drastically boost safeguards on digital privacy.

This practical tool for automatic face detection and blurring will thus increase people's control over their online identities while reducing unauthorized face visibility and increasing the safety of communities. The learned results and techniques from this project will also further enhance AI-driven strategies for privacy protection and set innovations in that critical area.

1.2 Problem Statement

The publication of people's faces without their consent has evident severe consequences for their reputation, identity theft, cyberbullying, and physical harm. Visual identification makes people vulnerable to several different types of aggression.
The techniques for analysing facial data include face detection, face recognition, and face verification. Face recognition predicts the identity from a predefined database; face verification decides whether it is the claimed identity of the face with a registered photo or not; face detection finds faces in images.
Identity Theft: Better face recognition technologies have created the opportunity to spot and identify persons from digital photo collections quickly. This could be a scenario where a potential attacker mines personal information and carries out identity theft. With an identified face, an attacker can access someone's social media profile, know where someone lives, or even obtain a sensitive financial view. Such access eases itself into fraudulent account openings, unauthorized transactions, or outright impersonation [17].

Cyberbullying: This situation is usually much easier and more devastating because images of the people concerned are utilized. Victims feel as if the bullying is personally intended for them, and they are harassed often with terrible psychological effects, reputational damage, and emotional scarring. Such bullying can be magnified by its visual nature, making it more personal and intrusive than other forms of cyber harassment [15].

Reputational Damage: Exposure of one's facial image against one's wish can severely damage one's public image. The wide dissemination of videos or pictures containing hurtful content that projects someone in a bad light through editing, manipulation, or just unflattering angles can cause severe and long-term harmful reputational effects [16]. In conclusion, the effects of disclosing the face of an individual in images without their consent could be devastating, ranging from identity theft to cyberbullying, damage to one's reputation, and physical harm.

1.3 Scope and Objectives

This project is about creating and deploying a tool that will use artificial intelligence to automatically detect faces in uploaded images and blur them, hence resulting in an enhanced privacy provision for the users in the scenario of digital media. The AI system must deal with these tasks so that the general image quality does not degrade yet performs accurately.

1.3.1 The three main components of the AI tool include the following:

1. **Face Detection:** The functionality that detects human faces in loaded images using the state-of-the-art concept of face detection algorithms. It mainly depends on some deep learning models, like YOLOv5/YOLOv8, to guarantee that robust detections can be made under difficult conditions, including the variability of illumination, poses, and image quality.
2. **Face Recognition:** The module, after face detection, independently applies YOLOv5, YOLOv8, and MobileNet for recognition of the identities from the set of pre-defined faces. It makes the decision on the specification of blurring based on the consent parameters agreed by the user.
3. **Face Blurring:** When blurring needs to be done, this module uses advanced functions for blurring the detected faces. Multiple blurring methods are applied one after the other, making faces not recognizable anymore, even by AI models like YOLO and MobileNet, maintaining the utility of the image but ensuring privacy.

1.3.2 System Workflow

These advanced-level algorithms are applied in systems for the detection of human faces and the processing of images; detected and recognized faces are then evaluated based on consent parameters for face blurring, so only those faces without permission are blurred. It gives excellent privacy protection with maintained image integrity because the surrounding content of the image is retained in the final result.

1.3.3 Project Objective

The main objective of this project is to serve as a powerful tool for helping the individual and, at the same time, allowing the organization to maintain and use their privacy. Also being utilized is an image that is used in confidential or privacy-respectful scenarios.

1.3.4 In Scope

Image Input: The system can accept different image files as input.

Multiple Face Detection and Recognition: In one image, the system can detect and recognize many faces accurately.

Selective Face Blurring: Faces are blurred based on non-consent, preserving the rest of the image quality.

1.3.5 Out of Scope

- **Video Processing:** The current tool is only developed for static image processing and does not handle video files.
- **Manual User Interaction:** The tool is fully automated and does not cater to manual interventions like face selection or detailed changes.
- **Full Privacy Enhancements:** Its primary purpose is face detection, recognition, and blurring; other features of privacy enhancement are not considered.

1.3.6 Targeted Users

- **Individuals:** For use personally, such as posting photos on social media or anywhere else, where they might need some identities to be blurred at their discretion.
- **Businesses:** For businesses that are marketing, advertising, and conducting internal communications to meet compliance with requirements for privacy enhancement and keep individual identifiers protected in public or internal company materials.

This AI tool is versatile and effective in maintaining privacy during this digital era, making its application among a wide range of users attractive for controlling their digital representations, whether personal or professional.

1.4 Report Organization (Structure)

Section 1: Introduction

This section gives the first overview of the AI-based tool developed for automatically detecting and blurring faces in images. Its purpose is to protect privacy in the digital world. It enables the reader to understand what problem this project is designed to solve clearly, and leads out the scope and objectives, work methodology, and work plan through a Gantt chart by the day.

Section 2: Related Work (State-of-The-Art)

- This section provides comprehensive information on related research, detailing works similar to my proposed project. It covers the methodologies used to develop these projects, offers a thorough literature survey, and includes an analysis of related works, making it clear how my project fits within the broader research landscape.

Section 3: Proposed Solution

- This section introduces the proposed solution aimed at mitigating the problem of unauthorized exposure of individuals' faces in online images, thereby reducing risks such as identity theft, cyberbullying, and reputational damage. It explains the solution methodology and elaborates on the tools and techniques used to implement and develop the project.

Section 4: Implementation

This is the section where the practical part of implementing the project is explained. It describes very well the technical steps taken, challenges encountered, and the solutions employed. It describes the trials with different models like YOLOv5, YOLOv8, and MobileNet for face detection and recognition, along with several blurring methods. The challenges faced and how it was addressed in terms of the limitations of datasets are further discussed in the following.

Section 5: Testing and Evaluation

- This section describes the testing procedures and evaluation metrics used to assess the performance of the implemented models. It covers the results of testing the face detection, recognition, and blurring models on both the LFW and Avengers datasets, highlighting the precision, recall, and F1-scores achieved.

Section 6: Results and Discussions

This section elaborates the discussion of results obtained after the testing and evaluation phase. It analyzed the performance of the various models in comparison: strengths and weaknesses, and the implications of such findings for the objectives of the project.

Section 7: Challenges and Limitations

- This section discusses the main challenges faced during the project, such as finding suitable datasets for the entire pipeline, the imbalanced nature of face recognition datasets, and the constraints of the Avengers dataset. It outlines the steps taken to address these issues, such as using the LFW dataset for face detection and supplementing the Avengers dataset with additional images.

Section 8: Trials

- This section provides a summary of the various models and approaches tried throughout the project, including MTCNN, CNN, YOLOv7, Haar Cascades, Facenet, and data augmentation techniques. It highlights why some models did not perform as expected and the limitations encountered during these trials.

Section 9: Conclusions and Future Work

- This section provides a summary of the project's achievements, including the successful implementation of AI-driven face detection, recognition, and blurring techniques. It outlines the limitations encountered, such as dataset constraints and challenges with highly cropped images. Additionally, it proposes future work, including the collection of a larger and more diverse dataset, and the expansion of the model to detect other sensitive objects to ensure comprehensive privacy protection.

1.5 Work Methodology

- **Submit the proposal**.
- **Research existing models and algorithms**: explore face detection, recognition models, and blurring algorithms.
- **Evaluate the existing models and algorithms**: check the strengths, weaknesses, and applicability of existing algorithms and models.
- **Explore AI-driven techniques for privacy protection**
- **Review state-of-the-art technologies**: Review state-of-the-art technologies in face detection, recognition, and blurring.
- **Write the interim report**
- **Gather diverse datasets** of high-quality images.
- **Preprocess the datasets**.
- **Choose Models**: choose flexible models that would be suitable for future work as well.
- **Implement the AI algorithm.**
- **Divide the datasets** into training, validation, and testing sets.
- **Train each detection and recognition model separately**.
- **Blurring Techniques and Pipelines**: Try various blurring techniques and create the pipelines using the pre-trained models that were adjusted and trained on my datasets.
- **Evaluate the models' performance on the validation and testing sets**.
- **Implement test cases.**
- **Document the implementation code.**
- **Finalize the thesis.**
- **Ensure individual privacy is preserved**: through using face recognition model on blurred images.

1.6 Work Plan (Gantt chart)

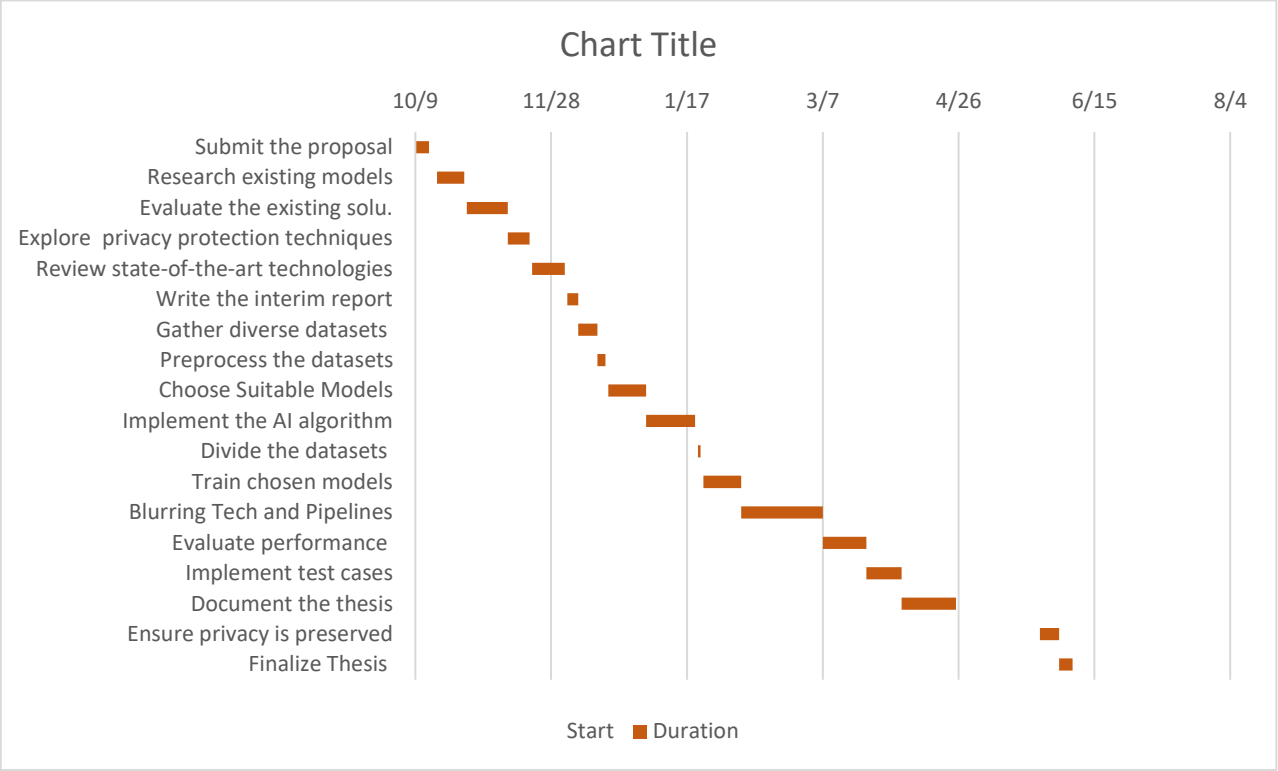


Figure 1 Gantt Chart

2 Related Work (State-of-The-Art)

2.1 Background

As personal data increasingly integrates with digital content, security and privacy concerns are escalating. The advancement of facial recognition technologies facilitates easy identification and tracking of individuals without their consent, necessitating effective methods to blur faces in images. This is crucial for maintaining privacy and adhering to legal and ethical standards while ensuring the utility of visual content.

AI-enabled solutions offer sophisticated facial blurring techniques that obscure identifiable features through pixelation, box blur, and Gaussian blur, without masking the entire face. These methods balance privacy with the usability of images, proving beneficial across various domains:

- **Individuals**: Share personal photos online without revealing bystanders' identities.
- **Journalists**: Protect the identity of sensitive sources in reports.
- **Businesses**: Enhance customer privacy in security footage and marketing materials.

The project uses advanced AI techniques, such as deep learning, for efficient face blurring while minimizing image distortion. This approach adequately balances privacy protection with content usability so that visual material becomes responsible and ethical.

Research Goals & Questions

The primary goal is to balance multimedia content utility and privacy using AI-based facial-blurring techniques. Key research questions include:

- How can AI effectively obfuscate faces in images while maintaining content utility?
- How can AI-based blurring strategies be made resilient against various attacks?
- What are effective evaluation practices for AI-based obfuscation methods regarding both privacy and content informativeness?

Components of the AI Solution The following are key components of the project:

Face Detection and Recognition: This phase involves identifying human faces in images, adapting the detection mechanisms to achieve high accuracy under diverse conditions.

Blurring Techniques: Implementing various methods such as pixelation, box blur, Gaussian blur, and more advanced techniques to effectively obscure faces while maintaining the context of the image.

Selective Blurring: Blurring faces based on explicit permission, ensuring privacy rights are respected. This step is crucial for the ethical deployment of AI.

Testing and Evaluation: Conducting extensive testing on predefined datasets and Internet-provided images to ensure the system's robustness and applicability in real-life scenarios.

General Approach

- **Privacy Preservation**: Using state-of-the-art algorithms to make faces unrecognizable, even with determined recognition attempts.
- **Utility Preservation**: Balancing optimal blurring practices with minimal image quality distortion to maintain valuable digital content for multiple uses.

2.2 Literature Survey

Paper [1] "Deep Learning Approach for Face Recognition"

Samar S. Mohamed, Wael A. Mohamed, A. T. Khalil, and A. S. Mohra proposed a face recognition system utilizing a deep learning approach with a Convolutional Neural Network (CNN) architecture. The developed model consisted of 15 layers: convolutional, batch normalization, ReLU, max-pooling, fully connected, and softmax layers. The CNN was trained by stochastic gradient descent to automatically extract and classify facial features from differently aligned face images.

This study is based on the Face96 dataset: it has 1000 images of 50 individuals, with each individual having 20 images. Images have different facial expressions, lighting conditions, backgrounds, poses, and camera distances. Preprocessing, in this case, included resizing images to 112 × 92 pixels, graying them, and normalizing them by histogram equalization.

A high degree of accuracy of 99.67% has been achieved by the CNN model on the Face96 database for the proposed model, with a 70% training and 30% validation split. The comparative performance of some state-of-the-art results is shown:

- **PCA + Euclidian Distance:** 93.33%
- **LBP + LTP:** 98.75%
- **DeepFace:** 97.53%
- **DeepID:** 97.45%
- **DeepID2:** 99.15%
- **DeepID3:** 99.53%

Additionally, the paper compared the proposed CNN method with a conventional method (Fuzzy Hidden Markov Models - FHMM) using the ORL database. CNN achieved 97.89% accuracy compared to the FHMM's 95% accuracy, demonstrating its superiority over traditional methods.

This study highlights the effectiveness of deep learning approaches in face recognition tasks, particularly the robustness of CNNs in handling various facial recognition challenges such as variations in expression, lighting, and pose. The findings support the trend of adopting deep learning techniques to enhance the accuracy and reliability of biometric systems.

Paper [2] “A Deep Learning Approach for Face Detection using YOLO”

Approach: This paper discusses an approach to face detection using a deep learning system based on the YOLO framework. YOLO is a real-time, state-of-the-art object detection approach in which a single neural network is applied to one complete image. It partitions this image into a grid and predicts bounding boxes and probabilities for each grid cell. It can be used to predict multiple bounding boxes and class probabilities concurrently, very fast, and accurately, using this method.

Dataset Applied: The experiments in this study are carried out by training and testing the proposed model on the FDDB dataset. The FDDB is made up of 5171 faces in a set of 2845 images intended for addressing the problem of face detection in unconstrained settings. This is split into 70% for training data and 30% for testing data, which translates to 2667 images, totalling 52.2 MB.

Model and Experimental Setup: The proposed network architecture takes an input image of size 448x448 and consists of:

- 7 convolutional layers
- Max-pooling layers of size 2x2
- fully connected layers
- Output layer followed by the last fully connected layer

The model was trained using the gradient descent optimizer algorithm for 25 epochs with a learning rate of 0.0001. The experiments were done on two different GPU setups:
• 8GB RAM, 2GB GeForce 820M GPU, Core i5
• Core i7 processor, 16GB RAM, 4GB NVIDIA GTX 1050 Ti GPU

YOLO-faced results: The proposed face detection model based on YOLO garnered 92.2% correctness from the test data after 20 epochs. The performance of the model was compared with other face detection algorithms, which gave better accuracy concerning the Haar-cascade algorithm and models of face detection based on R-CNN.

Comparison Against Other Models:
• Proposed YOLO Model: 92.2% accuracy
• Haar Cascade Algorithm: 83.8 which is a Low accuracy compared to YOLO
• R-CNN Based Face Detection: 89.6, that is an Accuracy lower than YOLO

Conclusion: The associated deep learning methods for large amounts of data require the networks to be used on NVIDIA GPUs with very high configurations. This has been shown by the research done. It was demonstrated that the learning rate changes along with the network size and the object's size where we estimate the best learning rate of 0.0001 for the network suggested of 18 layers. The number of optimal epochs for training was found at 20 epochs to avoid overfitting. For example, future work includes optimization of this model for microscopic face detection, different viewpoint variations, and partial face detection.

Paper [3] “Deep Learning Face Representation from Predicting 10,000 Classes”

Approach: This paper presents a method based on deep learning for learning representations of faces for performing verification developed under the name of Deep hidden IDentity features, or DeepID. It is further suggested that these challenging multi-class identification tasks result in features of general use, such as the task of face verification, working very well for new identities—ones not seen during training. Their essential idea is to train ConvNets so that they have the property of reducing the number of neurons along the feature extraction hierarchy, making a compact set of identity-related features at the top of the hierarchy.

Dataset Used: The model was trained on the CelebFaces dataset and tested on the Labeled Faces in the Wild (LFW) dataset. The CelebFaces dataset contains 87,628 face images of 5,436 celebrities from the Internet, while the LFW dataset contains 13,000 images of faces collected from the web, with each face labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the dataset.

Examples and Experimental Setup:

The architecture of the proposed ConvNets includes:

• Four convolutional layers with max-pooling

• A fully connected layer of DeepID

• A softmax output layer for classification of identity

The DeepID features are extracted from the last hidden layer neuron activations. The networks are trained to classify about 10,000 face identities simultaneously; this is more challenging than classifying binary face verification or identification tasks. We designed the task like that to make full use of the high learning capacity of neural networks in extracting compelling features for face recognition.

Their method, which they termed DeepID, achieved an accuracy of 97.45% on the LFW dataset when tested for face verification with only weakly aligned faces. This is slightly below human performance, which is at around 97.53%.

Comparison with Other Models

- **DeepID on CelebFaces+:** 97.20% accuracy
- **DeepID on CelebFaces+ with Transfer Learning (TL):** 97.45% accuracy
- **DeepFace (Facebook):** 97.25% accuracy

It was observed in the current paper that with an increase in the number of training identities, the verification performance increased monotonously, meaning that more training data would contribute further to better feature representations learned by the model.

Paper [4] “DeepFace: Closing the Gap to Human-Level Performance in Face Verification”

The Approach:

- The DeepFace system is based on a DNN with over 120 million parameters in total, several locally connected layers without weight sharing.
- It uses explicit 3D face modeling for alignment, and extracts face representations from a nine-layer deep neural network.

- The alignment step employs fiducial points and a piecewise affine transformation to arrive at a frontal view of the face before pushing it through the network.

The Dataset Used:

• The training data set involved 4 million facial images from over 4000 numbers.
• Was tested on the LFW (Labeled Faces in the Wild) dataset containing 13323 web photos of 5749 celebrities.
• Also tested on the YouTube Faces (YTF) Database, which includes 3,425 YouTube videos of 1,595 subjects.

Models and Accuracies

Model Details:

- **Input**: RGB face images 152x152 pixels in size and 3D aligned.
- **Layers**:
  - **Convolutional Layer 1 (C1)**: 32 filters of size 11x11x3
  - **Max-Pooling Layer (M2)**: 3x3 spatial neighborhoods with a stride 2
  - **Convolutional Layer 2 (C3)**: 16 filters of size 9x9x16
  - **Locally Connected Layers (L4, L5, L6)**: These layers apply filters to different regions of the image without weight sharing.
  - **Fully Connected Layers (F7, F8)**: Capture correlations between features in different parts of the face image.
- **Activation Function**: ReLU (Rectified Linear Units)
- **Output**: K-way softmax, where K is the number of classes (face identities).

Training

- This network was trained in the stochastic gradient descent (SGD) with a momentum of 0.9.
- The learning rate was manually decreased because the validation error was not decreasing anymore, from 0.01 to 0.0001.
- The network has been trained for 15 epochs over the entire dataset.

The Proposed Model:

- on the LFW dataset they achieved an accuracy of 97.35%.
- on the YTF dataset they achieved an accuracy of 91.4%.

Comparison Models:

- **Joint Bayesian**: 92.42%
- **High-dimensional LBP**: 95.17%
- **TL Joint Bayesian**: 96.33%

Conclusion

- The DeepFace system significantly advances the state-of-the-art in face verification, closely approaching human-level performance.
- The use of 3D alignment and a large-scale dataset for training contributes to its high accuracy.
- The results show a marked improvement over previous methods, reducing the error rate by more than 27% on the LFW dataset.

Paper [5] "Feature Extraction Technique of PCA for Face Recognition with Accuracy Enhancement"

Approach

• the paper intends to design a Face Recognition System based on Principal Component Analysis for the feature extraction method and Euclidean Distance for classification.
• Using PCA to reduce the dimensionality of the face images by extracting out the principal features and forming a feature space with lower dimensionality.
• Further, an Euclidean distance classifier matches the features extracted to a training set from which faces can be recognized.

The Dataset Used:

• The Yale Face Database: 165 images of 15 individuals.
• The images include variations in illumination, facial expressions, and the presence or absence of glasses.

The Models and Experimental Setup

1. **Face Detection**:
   - Images have gone through preprocessing, such as conversion to grayscale and histogram equalization.
   - The Haar classifier is used to detect faces stored in the training database
2. **Face Recognition:**
   - PCA is used for feature extraction of the detected faces.
   - The Euclidean distance classifier compares these features with the one in the training database resembling the same individual face.

Results:

- **Face Detection**: Achieved a detection ratio of 98.18% on the Yale database.
- **Face Recognition**: The highest accuracy was achieved for the "Happy" facial expression, with a recognition rate of 93.33%.

Conclusion

- For feature extraction, the PCA approach is working very well in identifying faces even under highly varying conditions, including facial expressions and with glasses on or off.

- The method showed lower performance under varying illumination conditions.
- Future work would consist of the use of Linear Discriminant Analysis for feature extraction instead of PCA, which indeed is going to result in improvements in recognition.

Paper [6] "Deep Face Recognition"

Approach

1. **Dataset Creation**:

- A large-scale dataset with 2.6 million images of 2,622 identities was assembled using a combination of automation and human annotation.
- The dataset creation involved scraping images from the web and using a human-in-the-loop approach to ensure quality.

2. **CNN Architectures**:

- The paper explores various CNN architectures for face identification and verification.
- The network architecture includes multiple convolutional layers followed by fully connected layers.
- Specific architectures used include VGG-16 and VGG-19, known for their depth and performance.

3. **Face Alignment and Metric Learning**:

- The paper emphasizes the importance of face alignment in improving recognition accuracy.
- Metric learning techniques, such as learning embeddings that minimize the distance between images of the same person while maximizing the distance between images of different people, are used.

The Dataset Used

- **Training Data**: A newly created dataset with 2.6 million images and 2,622 identities.
- **Benchmark Data**: Evaluations were conducted on standard benchmarks including the Labeled Faces in the Wild (LFW) and YouTube Faces (YTF) datasets.

Models and Accuracies

- **Proposed Model**:
  - They achieved on the Lfw dataset an accuracy of 98.95% .
  - They achieved on the YTF dataset an accuracy of 97.3%.
- **Comparison Models**:
  - **DeepFace**: On LFW they achieved 97.35% accuracy.
  - **FaceNet**: They achieved an 98.87% accuracy on LFW without alignment and 99.63% with alignment.
  - **DeepID-2,3**: They achieved a 99.47% accuracy on LFW.

Face Recognition Accuracy with Emotions

- The highest face recognition accuracy reported in the paper for the proposed CNN model is 98.95% on the LFW dataset.

Paper [7] "Facial Expression Preserving Privacy Protection Using Image Melding"

The Approach:

• The present paper proposes an image welding-based approach for privacy protection of the images, keeping the facial expression.
• The technique does not tamper with the facial regions in an obvious way, though the non disclosable subjects do keep their expression.
• This contrasts directly with previous approaches, for instance, peeling and covering up faces, which will often entail the cost of the quality and context of the image.

The Dataset Used:

• The dataset used for the experiments includes images of 8 subjects with frontal faces and profile faces in happiness, disgust, and neutral expression.

The Models and Experimental Setup:

1. **Face Detection**:

- The paper does not talk of a specific model for face detection. Still, the idea of an existing face detection method to annotate faces, whether it is a disclosable face or non-disclosable face.

2. **Face Recognition:**

- Local Binary Pattern Histograms (LBPH) are applied for face recognition to evaluate how well the proposed method of privacy protection works.
- LBPH is a widely used face recognition method where local binary patterns are extracted from an image, and histograms of such patterns are constructed.

3. **Privacy Protection Method**:

- The proposed method uses a source image (another person's face) and a target image (the image to be protected).
- Corresponding points on the target and source images are specified to retain facial expressions.
- The image melding algorithm adjusts the source image to match the target image's corresponding points and mixes them using a weighted mask.

Original Recognition Rate:

- The recognition rate of the original face images (when no privacy-protecting method is applied) is 88% (28/32 correct recognitions).

Recognition Rates After Applying Privacy Protection Methods:

1. **Blurring:**
    - Parameter (i): 53% (17/32 correct recognitions)
    - Parameter (ii): 56% (18/32 correct recognitions)
    - Parameter (iii): 38% (12/32 correct recognitions)
2. **Morphing:**
    - **Recognition Rate After Morphing:**
        - Parameter (i): 53% (17/32 correct recognitions)
        - Parameter (ii): 41% (13/32 correct recognitions)
        - Parameter (iii): 38% (12/32 correct recognitions)
3. **Proposed Method:**
    - **Recognition Rate After Using Proposed Method:**
        - Parameter (i): 44% (14/32 correct recognitions)
        - Parameter (ii): 31% (10/32 correct recognitions)
        - Parameter (iii): 22% (7/32 correct recognitions)

Summary

- **Original Images: 88% (28/32)**
- **Blurring:**
    - Best: 56% (18/32)
    - Worst: 38% (12/32)
- **Morphing:**
    - Best: 53% (17/32)
    - Worst: 38% (12/32)
- **Proposed Method:**
    - Best: 44% (14/32)
    - Worst: 22% (7/32)

The proposed method significantly lowers the recognition rate, and it is proven to be effective in maintaining users' privacy regarding facial expression information simultaneously — unlike other methods, which blur and morph faces in the visual stream.

Paper [8] “AnonymousNet: Natural Face De-Identification with Measurable Privacy”

The Approach: The paper proposes a novel framework called AnonymousNet, aimed at addressing the issues of balancing usability and privacy in face de-identification. The framework consists of four stages:

1. **Facial Attribute Estimation**: Using a deep convolutional neural network to extract facial features and predict attributes.
2. **Privacy-Metric-Oriented Face Obfuscation**: Modifying the facial attributes based on privacy metrics like k-anonymity, l-diversity, and t-closeness.
3. **Directed Natural Image Synthesis**: Generating photo-realistic de-identified faces using a generative neural network.
4. **Adversarial Perturbation**: Introducing adversarial noise to further protect against face recognition systems.

Dataset Used

- The CelebA dataset, which consists of 202,599 images of 10,177 identities with 40 attribute labels for each image, is used for training and evaluation.

Models for Face Detection and Recognition

1. **Face Detection**:
    - **Deep Alignment Network (DAN)**: Applied to detect landmarks on faces, aligning them, and cropping to a standard size.
2. **Face Recognition**:
    - **GoogLeNet**: This was for facial attribute extraction, and later, random forests were in place to predict facial attributes.

Results

- The paper demonstrates significant improvements in both photo-realism and privacy protection compared to traditional methods like blurring, pixelation, and masking.
- The effectiveness of the proposed AnonymousNet framework is shown through qualitative and quantitative evaluations, including perceptual image quality metrics like PSNR, SSIM, and MS-SSIM.

Recognition Rates Before and After Each Method

1. **Original Recognition Rate**:
    - Recognition rate of original face images: 88% (28/32 correct recognitions).
2. **Blurring**:
    - **Parameter (i)**: 53% (17/32 correct recognitions)
    - **Parameter (ii)**: 56% (18/32 correct recognitions)
    - **Parameter (iii)**: 38% (12/32 correct recognitions)

3. **Morphing**:
   - Recognition rates after morphing vary based on the parameters used, but generally show lower recognition rates compared to the original images.
4. **Proposed Method (AnonymousNet)**:
   - The recognition rates after using the proposed method are significantly reduced, indicating effective de-identification while preserving visual quality.

Paper [9] – "Face Detection in Extreme Conditions: A Machine-learning Approach"

With the need for face detection and hence difficulty related to most applications, "Brightness of Faces" can be reflected. Two related face identification algorithms are compared in the literature review: Haar Cascade and MTCNN, for dealing with dimming, shadows, and pose variations. In complicated conditions, however, MTCNN is the more efficient choice, while Haar Cascade is typical but more practical for real-time applications. This research will add to our understanding of the new techniques in face detection. The author tests the suggested MTCNN on its dataset. Performance evaluation has proved it was very effective in face detection with good reliability in the image, which makes an accuracy rate of 99%. Then, the suggested method is compared with the standard machine learning-based Haar cascade algorithm, which is used for object detection and primarily facial recognition. The results indicate that MTCNN has a much better performance than the Haar cascade, which attains only an accuracy rate of 68%.

COMPARISON OF DETECTION RATE OF FACIAL POINTS AMONG VARIOUS METHODS FOR MY OWN DATASETS

Algorithm	Accuracy
Haar Cascade	68.16%
MTCNN	98%
Viola-Jones	61.81%

Figure 3 Comparison of detection rate

CONFUSION MATRIX OF FACIAl POINTS DETECTION(MTCNN)

	Positive	Negative
Predicted Positive	17620	335
Predicted Negative	30	280

Figure 2 Confusion Matrix Results

CALCULATION OF CONFUSION MATRIX FOR FACIAL POINTS DETECTION

Measure	Value(%)	Derivation
Precision	98.13	PPV=TP/(TP+FP)
Recall	99.83	TPR=TP/(TP +FN)

Figure 4 Calculation of confusion matrix

Paper [10] – “Face Recognition and Identification using Deep Learning Approach”

This paper presents a deep learning face recognition system based on OpenCV and emphasizes the importance of facial features for identification. The application of OpenCV in Python and its applicability for precise face recognition are examined in the literature review. The article shows experimental findings confirming the proposed face recognition system and compares the accuracy of deep learning with standard methods. The study promotes deep learning as a powerful face recognition technique and uses OpenCV to help with its effective application. The basic image processing algorithm is shown in Figure 2, with a special focus on Matlab and OpenCV time consumption. Experiments further reveal that OpenCV can complete the erosion process up to 30 times faster than Matlab, and even up to 100 times quicker.

However, because Matlab offers a variety of functions and methods, it is a more straightforward and user-friendly environment. While memory leaks and memory allocation are not issues with Matlab, they are extremely important for OpenCV. Figure 3 displays the flowchart of the facial recognition operations. claiming that an input needs to be located and verified in order to identify a face. As a result, the image sensor or typically the camera must be set up in order to record or capture images. The camera and software need to interact. The next step is the input image. Images and videos captured in real time or in the past can be used as input. Once input is received, faces in images or videos will be identified. After being trained, the classifier can be used to begin recognizing tasks. It can be used to identify many people in pictures or videos. Various sets of Python scripts are offered for various recognizing tasks. In order to identify an individual, The Python program will import the classifier that was trained in the previous step through a camera or photo.

Face identification uses cascade classifiers based on Haar features, of which Haar Cascade is the main face classifier. Basically, Haar Cascade is a classifier that recognizes the item based on the training source. A Har series is produced by overlaying a positive image above a number of negative photographs. On the server, training is usually done in phases. To improve performance, make use of high-quality photos and increase the number of training stages for the classifier.

TensorFlow is the framework utilized in the system's classifier component. After training, the classifier is employed in the recognition process. The training method is hard to get a better classifier. The longer the training periods, the better the classifier performs. Training for the suggested face recognition system takes three days. Longer training sessions can lower loss even further and boost accuracy. It is possible to argue that the accuracy of facial recognition on photos is better to that of real-time video, where the A classifier trained on a large dataset of face photographs has been verified to be able to identify images with an accuracy of 91.7% and real- time video with an accuracy of 86.7%. stating that the image resolution is clearly far higher than the live video. The video's frame rate is low due to the processing system's constraints, which reduces the clarity of the face captured in the image. Therefore, the classifier often believes that a particular person has the other person's feature. So, the accuracy of real-time video recognition decreased to 86.7%. Numerous images per candidate and the CNN methods are used to train this model. This led to the creation of large datasets, which improved accuracy overall. By evaluating the data, it is possible to determine how the lighting affects the recognition process. The recognition mechanism is more likely to identify

things incorrectly in low light. This might be addressed by including additional low-light training images in the face classifier's development process. Whether in picture or real-time video, Person 3 performs better than the other two in terms of accuracy for both recognitions.

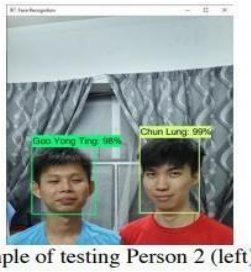


Figure 10. Sample of testing Person 2 (left) and 3 (right).

Figure 9. Sample of testing Person 1 (true).

Figure 5 sample of testing

Paper [11] – “Design and Evaluation of a Real-Time Face Recognition System using Convolutional Neural Networks.”

The study emphasizes CNN's capability of feature extraction and classification concurrently. Thus CNN has set a milestone by launching a real-time CNN facial recognition system. The literature reviews the parameters of CNN And emphasizes how they affect the system's precision. According to the research, the solution CNN, that includes pre-processing techniques and the Viola Jones algorithm for face detection, is designed purely for real-time facial recognition. This work helps better understand the applications of CNN in real-time facial recognition. An overview of the proposed system of real-time CNN facial recognition is demonstrated in Figure 6.

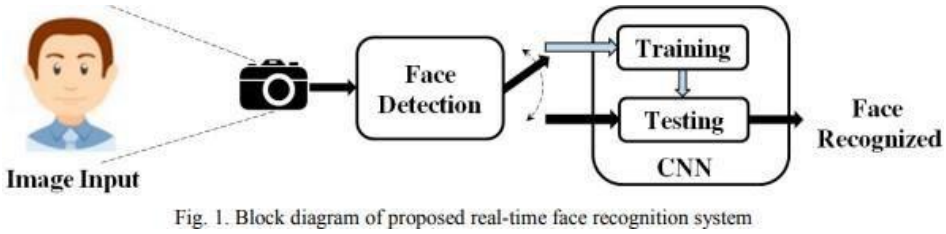


Figure 6 Block diagram of proposed real-time face recognition system

Preprocessing techniques, the architecture selected presented by the convolutional, pooling, discrete layers, and the Viola Jones algorithm for face detection. Choosing the leakage rate to be 0.5 and taking a maximum aggregation is very important. The last layer detects the faces of five people in real time. The structure of the network can vary with accuracy tests.

The performance of the designed facial recognition system was tested on the AT&T database. This standard database comprises ten pictures per subject of 40 people, which is 400 images. Some samples of the AT&T database are presented in Fig. 7. The 320 photographs used for training consisted of eight images from each of the 40 participants; the remaining 80 were for testing. The effectiveness of the proposed system is characterized by changing the number of filters in the convolution layer and the window size of the convolution filter for different pooling window sizes.

Fig. 8. Samples from AT&T database employed for the evaluation of proposed system

Figure 7 Samples of AT&T database

Figure 8 also shows that the maximum identification performance of the proposed system is 98.75%, using pooling window sizes of 2 × 2 and 4 × 4 pixels and a convolution filter of size 3 × 3 pixels with 32 filters. Table 1 compares the performance assessment of suggested work with those published in the literature against the results of the suggested dataset/face recognition. Based on observation, we conclude that the CNN architecture and the recommended approach align with research reported in a published paper. The suggested approach to optimization of pooling, convolution filter window size, and convolution filter count has improved the recognition accuracy.

Table 1. Comparison of Face recognition results reported in literature for AT&T datasets

Reference	Method	Recognition Accuracy
[6]	HOG and RVM	97.00%
[10]	PCA and MDC	96.70%
[11]	PCA and MDC	95.63%
[12]	PCA and KNN	92.00%
[13]	ICA and MDC	85.00%
[14]	Markov Random Fields	86.95%
[15]	Eigen face	92.87%
[16]	Statistical	92.60%
[17]	CNN	98.30%
[18]	CNN	95.00%
Prop. Work	CNN	98.75%

Figure 8 Comparison of Face Recognition Results

Figure 9 depicts a screenshot of the output results from the proposed real-time face recognition system during a live demo. The proposed system uses inputs of author photos, and the identification of the authors can be seen in the top left corner. As shown in Figure 9, the suggested system first searches for the face in the picture detects it and then displays the person's identification.

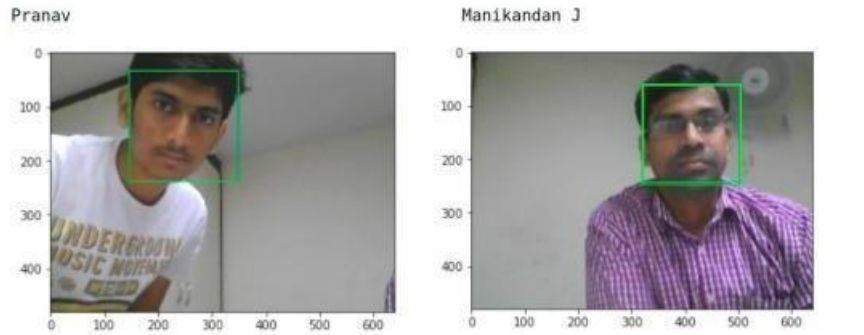


Fig. 12. Snapshot of results obtained during evaluation of the proposed real-time face recognition system

Figure 9 Snapshots of results obtained using the real-time system

It should be noted that the AT&T dataset doesn't need any pre-processing because every image is the same size, in grayscale, and has only faces.

Paper [12] "Face Recognition with Machine Learning"

Approach: This paper presents a machine learning application-based facial recognition framework. More specifically, the means of implementation is: SVM. The system will include the acquisition of training set images of faces to develop a model of a face recognizer; this model can classify an individual based on the analysis of their facial features.

Dataset Used

- The paper does not detail or mention any specific data set on which the training and testing of the face recognition model are based. Nevertheless, it refers to the building of a database of face images for training and evaluation.

Face Detection and Recognition Models

1. **Face Detection**:
    - The system uses the Dlib library to detect a face, which will detect whether a face is present in a photo and compute its position.
2. **Face Recognition**:
    - The system uses Support Vector Machines for face recognition, classifying detected faces based on the extracted features.

Results

- The system managed an accuracy of over 90% in correctly matching the input face with the right person from the database, thus showing a high level of precision in its ability to recognize the faces.

Recognition Rates Before and After Each Method

In detail, the paper does not give recognition rates before and after applying special techniques, such as blurring or morphing. Instead, it focuses on the performance of the proposed system using SVM generally for face recognition.

Architecture Design

The architecture of the facial recognition system consists of three main modules:

1. **Enrolment Module**: Scans and captures analog or digital images of individuals.
2. **Database Module**: It manages the compression, processing, storage, and comparison of the captured data with the stored data.
3. **Identification Module**: Done in the application system, it interacts with face recognition.

The process generally includes three stages:

1. **Face Detection**: Detect whether a face is in the image and locate it.
2. **Feature Extraction**: Process the detected face and convert it into a set of fiducial points or vectors.
3. **Face Recognition**: face identification by matching extracted features against this database.

Mathematical Model:

The mathematical model for the system includes the following steps:

1. Read the training set of N*N images.
2. Resize image dimensions and find the average face.
3. Calculate covariance matrix and eigenvectors.
4. Create a reduced eigenface space.
5. Calculate eigenface of the image in question and Euclidean distances between the image and the eigenfaces.
6. Identify the image with the minimum Euclidean distance or mark it as unrecognizable.

Pros and Cons:

PROS

• More secure, fast, and reliable recognition, automatic identification, inexpensive, non-contact process, easy to implement into the current system.

Cons:

• Surveillance angle dependency, requirements for data storage, and concerns regarding legislation and privacy.

Applications

- Monitoring students during online exams, attendance tracking, phone unlocking, finding missing people, computer security, banking using ATMs, social media identification, recognizing VIPs at events, prison visitor systems, border control, and voting systems.

Paper [13] “Face Recognition Using Deep Learning”

The Approach: The article suggests a face recognition system that uses deep learning techniques, particularly Convolutional Neural Network (CNN) and the VGG16 network architecture. The main goal behind this is to increase the efficiency and accuracy of face recognition, particularly to recognize those of refugees or missing persons.

Dataset used:

The dataset comprises photos from 21 families, where the total number of photos accounted for was 84 initially. We have used data augmentation techniques, which have been expanded to 1512 images altogether. The ratio is 80% for training and 20% for testing.

Models for Face Detection and Recognition

1. Face Detection:

o Convolutional Neural Network (CNN): The system features CNN for face detection from the given images. I have implemented the following architecture, VGG16, a famous deep-learning architecture.

2. Face Recognition:

o VGG16 Architecture: To develop the face recognition system, the VGG16 pre-trained model will be used. The ImageNet dataset consists of 13 convolutional layers and 3 fully connected layers, making a total of 16 weight layers.

Results:

• The system proposed yielded high accuracy in face recognition. All augmentation techniques with the VGG16 model have contributed to this improvement, rendering the system effective in identifying individuals from the given dataset.

Implementation Detail

1. Data Pre-Processing:

o RGB to Grey Conversion: Conversion from RGB to greyscale of each image.

• Resizing: Resizing images to 224x224 in accordance with the input size required by the VGG16 model.

2. Data Augmentation:

o Techniques such as shifting, rotation, zoom, brightness adjustment, and shearing are used to increase the diversity and size of the dataset.

3. Model Training:

The VGG16 model is trained in stages on the augmented dataset. Training involves extracting features of the convolutional layers and using fully connected layers for classification.

4. Testing:

o The model is tested on how it performs on the remaining 20% of the dataset. It gave impressive results of recognizing faces accurately.

Recognition Rates Before and After Each Approach:

Original Recognition Rate:

o It was not stated in this paper what the recognition rate of the original face images was; that is, before applying any privacy protection method.

Recognition rates post data augmentation using the VGG16 model:

o The system achieved over 90% accuracy in face recognition after applying data augmentation and training with the VGG16 model.

Paper [14] "Design and Implementation of a Face Recognition System Based on API Mobile Vision and Normalized Features of Still Images".

Brief Explanation:

A two-step methodology, this paper presents the design and implementation of a face recognition system in the Android mobile phone platform. It involves feature extraction followed by recognition using pattern classification. In our design, we have implemented face detection using Google's Mobile Vision API, which locates the facial landmarks of the nose, mouth, eyes, and cheeks. Thirty geometrical measurements are derived per person from these landmarks. This is the measuring Pearson Correlation Coefficient between measurements of the testing image and those contained in the training database. If the relationship is above a given threshold value, the system identifies the person and grants access to mobile applications.

Dataset Used:

The dataset for training and testing consists of face images, but no information regarding the size or source of the dataset is given in the paper.

Models Employed:

• Face Detection: Google Mobile Vision API

• Feature Extraction: Geometrical measurements of facial landmarks

• Recognition: Pearson Correlation Coefficient

The Achieved Accuracy:

A proposed face recognition system achieves a high accuracy rate: over 95%.

Compare with Other Methods:

The paper, in turn, compares its results with a few other approaches:

• WEKA Library: 80% accuracy

• Principal Component Analysis (PCA): 93.8% accuracy

• Linear Discriminant Analysis (LDA): 96% correct

• PCA (another study): 92% accuracy

- Eigen Faces: 84.3% Accuracy
- Fisher Faces: Accuracy of 94%
- Correlation Coefficient (1st system): 94
- Correlation Coefficient (2nd System): 93

Summary:

This paper's approach to face recognition, using the Pearson Correlation Coefficient on an Android-based device, shows performance much better than other methods, with accuracy greater than 95%. Such results point to the efficient use of Mobile Vision API by extracting and normalizing features to raise the accuracy of face recognition.

2.3 Analysis of the Related Work

Many studies prove the popularity of deep learning techniques for face detection and recognition. It has been proven to be accurate and robust while using Convolutional Neural Networks. For instance, Samar S. Mohamed et al. showed a fantastic 99.67% accuracy using a 15-layer CNN architecture for the Face96 database. It has outperformed traditional methods based on PCA and LBP under efficiency considerations in dealing with variations in expressions, lighting, and pose. At the same time, YOLO-based systems have shown much higher efficiency in real-time object detection: YOLOv5 and YOLO8 achieved 92.2% accuracy on the FDDB test set. This means that YOLO processes considerably faster and is more accurate in detection than traditional algorithms like Haar Cascade or R-CNN.
On the other hand, using large-scale datasets dramatically improves the performance of learning better models. For instance, in the work "Deep Learning Face Representation from Predicting 10,000 Classes," the need for extensive and diverse training samples is underlined since their model returned the accuracy of 97.45% on LFW through such massive data, i.e., CelebFaces and LFW. Similarly, the Facebook DeepFace system potentially nearly equals human performance by training on a dataset containing 4 million images with advanced 3D face modeling, recording 97.35% on LFW. Illustrative examples of how comprehensive datasets markedly enhance the generalization capabilities of deep learning models include:. Traditional methods still find their niche, though sometimes superseded by the deep learning models in appropriate contexts. For example, PCA-based feature extraction, while it is very effective compared to varying illuminations, can still be very effective under controlled conditions or when computational resources are bound. This implies a continued role of traditional methods alongside advanced techniques, especially in scenarios that require efficiency and predictability.

The development of face recognition has made a significant contribution to the use of real-time processing. This research will discuss a live practical application of CNN to achieve 98.75% accuracy in the AT&T database through a real-time CNN face recognition system with preprocessing techniques and the Viola-Jones algorithm.

Privacy concerns and de-identification imperatives are some of the most critical topics. Works like "Facial Expression Preserving Privacy Protection Using Image Melding" and "AnonymousNet: Natural Face De-Identification with Measurable Privacy" introduce innovative methods to protect personal information while preserving facial expressions and photo-realism. For example, AnonymousNet employs deep learning to change facial attributes and add adversarial noise to a high degree in an image, thereby reducing recognition rates while maintaining image quality.

Experiments of machine learning algorithms for face classification tasks continue with SVM. An SVM-based face classification study was able to achieve over 90% accuracy, which suggests that machine learning is still suitable for contexts that are prohibitive toward the resource-intensive nature of deep learning. A notable observation across many of these studies is the use of custom or proprietary datasets for training and testing their models.

This trend indicates a focus on tailoring datasets to specific research needs and conditions, which can enhance the relevance and applicability of the findings but may also limit the generalizability of the results across different datasets.

Overall, the literature indicates a strong trend towards leveraging deep learning and large-scale datasets to improve face recognition systems' accuracy and robustness. Real-time processing advancements and privacy-preserving techniques further expand these systems' applicability. Traditional methods and machine learning approaches still offer valuable solutions in specific use cases or resource-constrained environments, demonstrating a balanced and comprehensive approach to face recognition technology.

3 Proposed solution

3.1 Solution Methodology

After quite a search, two datasets were used in this project because available ones tend not to be suitable for comprehensive face detection and recognition tasks. Many datasets lack the proper labels for face recognition or come without bounding box coordinates that are necessary for effective face detection. For instance, most of the datasets are either imbalanced or too small, with as little as 15 images per class, which is not enough to train a good model.

The YOLO models selected for this task were YOLOv5 and YOLOv8 since they are very flexible and perform well in object detection of different types. This was a critical capability because it could fit with the specific necessities of face detection and recognition yet still bear the limitations typically found in the more specialist models. Finally, the Avengers dataset was fine-tuned for the training process to consider it the main one to serve all project purposes. This became enriched with the help of the generation of bounding boxes and synthesizing data from another dataset to increase the number of images for each class, hence mitigating the class imbalance problem and improving the robustness of the training model.

The use of YOLO models for both detection and recognition, along with MobileNet for some added recognition tasks, was tactical. These models have been known to require relatively small datasets to reach significant performance, and this renders them highly applicable in the scope of this project—it is difficult to procure large and well-labeled datasets. This approach ensures complete and practical solutions to the tasks of face detection and recognition, with simultaneously high accuracy and privacy compliance.

The models were tested by combining state-of-the-art deep learning techniques with some newer and innovative approaches, like a concatenation of various blurring techniques. They were used selectively where explicit permission or rights for use were not available to protect privacy while enabling the preservation of image characteristics. This was to ensure a performance improvement or at least equivalence to that of the traditional models applied in studies conducted earlier.

3.1.1 Capabilities of YOLO in this project are summarized as follows:

- **High detection and recognition accuracy:** YOLO models, particularly the YOLOv5 and YOLOv8 architectures, are employed effectively in both face detection and recognition tasks under consideration. These models have demonstrated good accuracy through the used datasets and adaptability to challenges from both detection and recognition processes.
- **Robustness in various conditions:** Experience has also revealed that they are robust under all diverse datasets and weather conditions. This involved controlling changes in lighting, facial

expressions, and orientations to ensure that detection and recognition were consistent across different scenarios.

- **Scalability and Adaptability:** The YOLO models were highly flexible and scalable—this was necessary for adapting the models to project requirements. These features are of particular importance for this project because the models work well with even a small set of training data and are general enough to apply to most tasks that require object detection.
- **Effective for Diverse Applications:** Their flexibility makes the YOLO models particularly useful for this project because they must effectively detect and recognize faces, but large and perfectly balanced datasets cannot be obtained. An ability is essential for projects in which enormous, well-annotated datasets can be complex to get.

3.1.2 Functional/ Non-functional Requirements

Functional Requirements:

1. **Privacy Protection**: The tool blurs faces without explicit permission, ensuring privacy while maintaining high-quality visual content.
2. **Face Detection and Recognition:** It promises appropriate face detection and recognition with YOLO and MobileNet.
3. **Selective Blurring:** It makes automatic blurring based on the predefined criteria; its intensity is variable, and it maintains a tradeoff between privacy and clarity.

Non-Functional Requirements:

1. **Performance:** This should be fast during the processing of images in order to serve the intended real-time needs.
2. **Accuracy:** The detection and recognition features are high, hence making it assuredly reliant on anonymization.
3. **Usability:** It should be friendly so that even people with different technical or nontechnical skills could use it; this is to make it reach larger masses.

3.2 Design / Simulation set up

3.2.1 The Full Generic Project Pipeline:

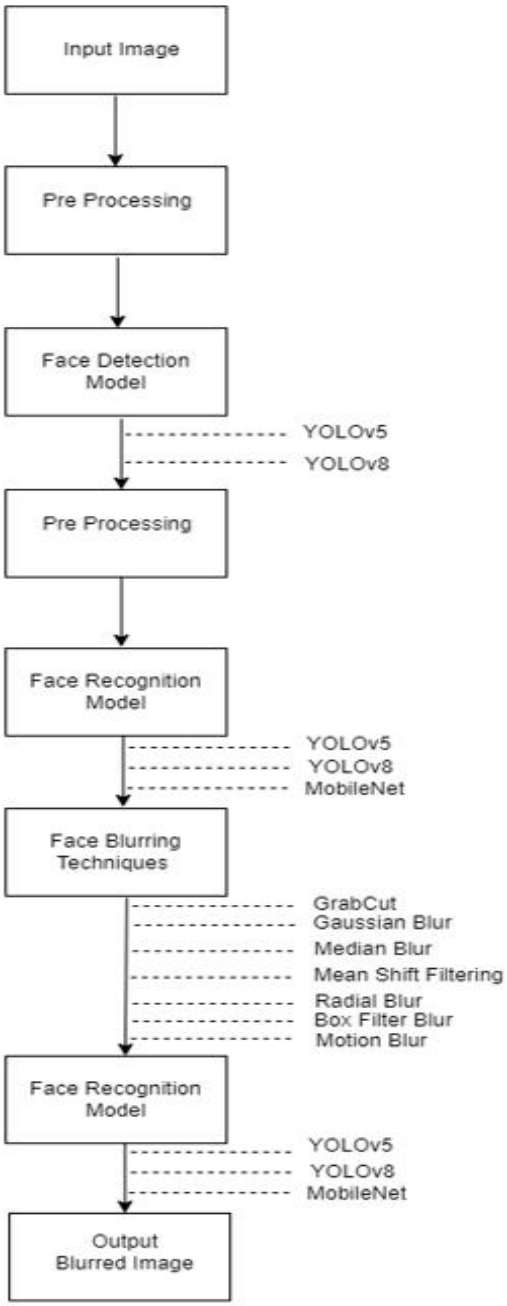


Figure 10 The Generic Project Pipeline

The Eight Stages summarised in the figure above (Full Generic Project Pipeline) are:

• **Input Image**: This is the primary phase where an image is input.

• **Pre-Processing**: Convert image to RGB.

• **Face Detection Model**: At this stage, the faces are detected in the input image. The two models that may be possibly used include YOLOv5 and YOLOv8.

• **Pre-Processing**: Crop the faces using the detected bboxes by the face detection model, resize the image to (224,224), and convert it to a numpy array.

• **Face Recognition Model:** Following the detection process, the recognized faces are processed into recognition with a model. The basic models are YOLOv5, YOLOv8, and MobileNet, among others.

• **Face Blurring Techniques:** The detected and recognized faces, obtained from the prior steps, are blurred using the following techniques: GrabCut, Gaussian Blur, Median Blur, Motion Blur, Mean Shift Filtering, Radial Blur, and Box Filter Blur.

• **Face Recognition Model (Re-evaluation):** The blurred images are further analyzed through a face recognition model, first through YOLOv5, then re-evaluated on YOLOv8, and finally through MobileNet, for establishing the person's identity.

• **Output Blurred Image:** The final stage is outputting the blurred image that has to be designed in the face recognition models again to ensure the fact that the blurring was correctly done to anonymize people in the picture.

3.2.2 The YOLOV8 For Face Detection and Recognition Project Pipeline (Pipeline 1):

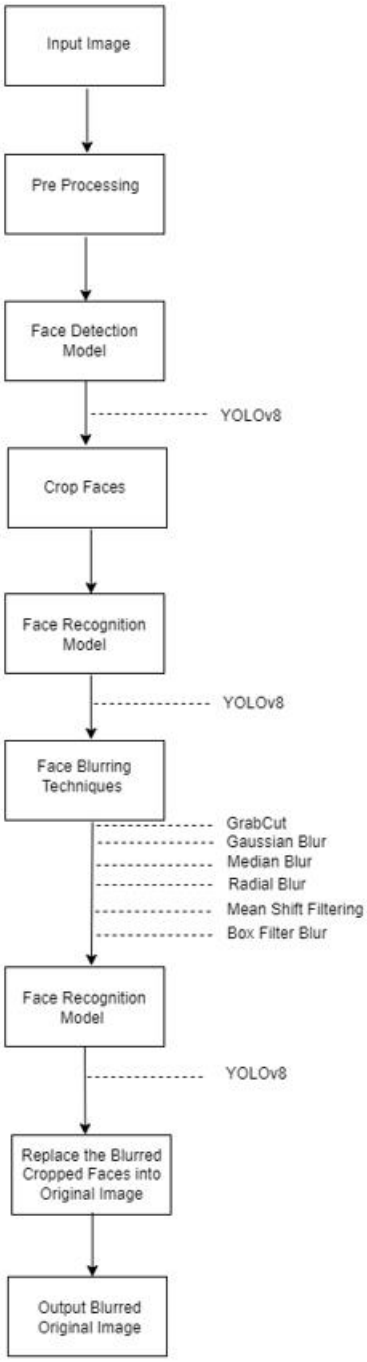


Figure 11 The Yolov8 Pipeline

The Nine Phases as shown in the figure above: The YOLOV8 For Face Detection and Recognition Project Pipeline

• **Input Image:** The process is initiated with an input image fed into the pipeline.

• **Pre-Processing**: Include image conversion to RGB.

• **Face Detection Model**: In this input image, a YOLOv8 model is used to detect faces. This model localizes the position of faces and returns this information in the form of bounding boxes usually around the detected faces.

• **Crop Faces:** The detected faces are cropped from the original image according to bounding boxes rendered by the face detection model. This isolates the faces for further processing.

• **Face Recognition Model:** The detected cropped faces are then passed through another YOLOv8 model, except this time used to classify each face identity. This stage determines whose face the individuals are based on the features learned by the training dataset.

• **Face Blurring Techniques:** This is the list of applied blurring based on the recognition score and the desired level of privacy for the recognized face. Some techniques used are GrabCut, Gaussian Blur, Median Blur, Motion Blur, Mean Shift Filtering, Distance-Based Blur, and Box Filter Blur. Qualitatively, each of the applied algorithms has different ways of changing the facial visual appearance to make the facial identity less recognizable or correct a peculiar distortion appealingly for aesthetics.

• **Face Recognition Model (Post-Blurring):** The blur face is passed through the YOLOv8 model again. This stage is critical to analyze just how effective the blurring techniques are in hiding the identities, meaning if the blurred faces are still recognizable by the model.

• **Replace Blurred Cropped Faces into Original Image:** The edited faces are reinserted in the areas of the source image, which were blurred previously, replacing the clear, unblurred faces.

• **Output Blurred Original Image:** The final output that has been produced is the image, in its original form, with the blurred faces while everything else in the image remains intact. And this is outputted from the system.

3.2.3 The YOLOV5 For Face Detection and Recognition Project Pipeline (Pipeline 2):

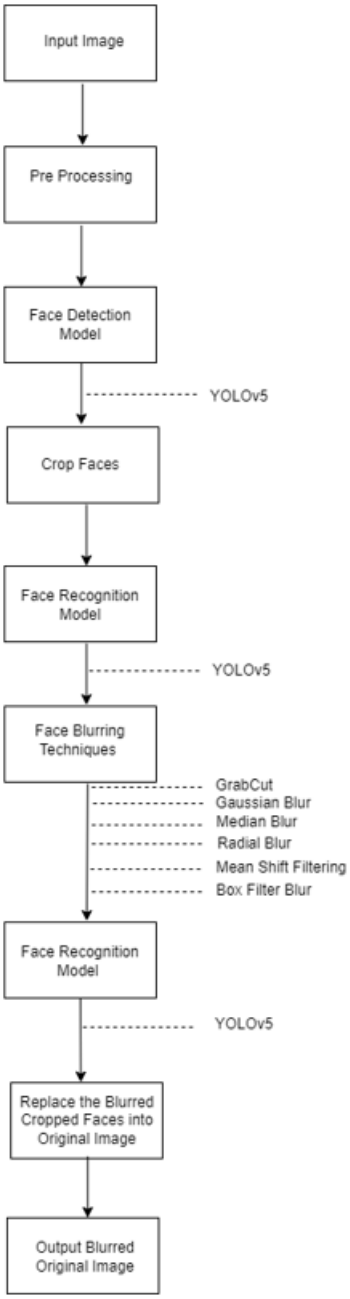


Figure 12 The YoloV5 Pipeline

The The Nine Phases as shown in the figure above: The YOLOV5 For Face Detection and Recognition Project Pipeline

• Input Image: The process is initiated with an input image fed into the pipeline.

• Pre-Processing: Include image conversion to RGB.
• Face Detection Model: In this input image, a YOLOv5 model is used to detect faces. This model localizes the position of faces and returns this information in the form of bounding boxes usually around the detected faces.
• Crop Faces: The detected faces are cropped from the original image according to bounding boxes rendered by the face detection model. This isolates the faces for further processing.
• Face Recognition Model: The detected cropped faces are then passed through another YOLOv5 model, except this time used to classify each face identity. This stage determines whose face the individuals are based on the features learned by the training dataset.
• Face Blurring Techniques: This is the list of applied blurring based on the recognition score and the desired level of privacy for the recognized face. Some techniques used are GrabCut, Gaussian Blur, Median Blur, Motion Blur, Mean Shift Filtering, Distance-Based Blur, and Box Filter Blur. Qualitatively, each of the applied algorithms has different ways of changing the facial visual appearance to make the facial identity less recognizable or correct a peculiar distortion appealingly for aesthetics.
• Face Recognition Model (Post-Blurring): The blur face is passed through the YOLOv5 model again. This stage is critical to analyze just how effective the blurring techniques are in hiding the identities, meaning if the blurred faces are still recognizable by the model.
• Replace Blurred Cropped Faces into Original Image: The edited faces are reinserted in the areas of the source image, which were blurred previously, replacing the clear, unblurred faces.
• Output Blurred Original Image: The final output that has been produced is the image, in its original form, with the blurred faces while everything else in the image remains intact. And this is outputted from the system.

3.2.4 The YOLOV5 For Face Detection and MobileNet for Face Recognition Project Pipeline (Pipeline 3):

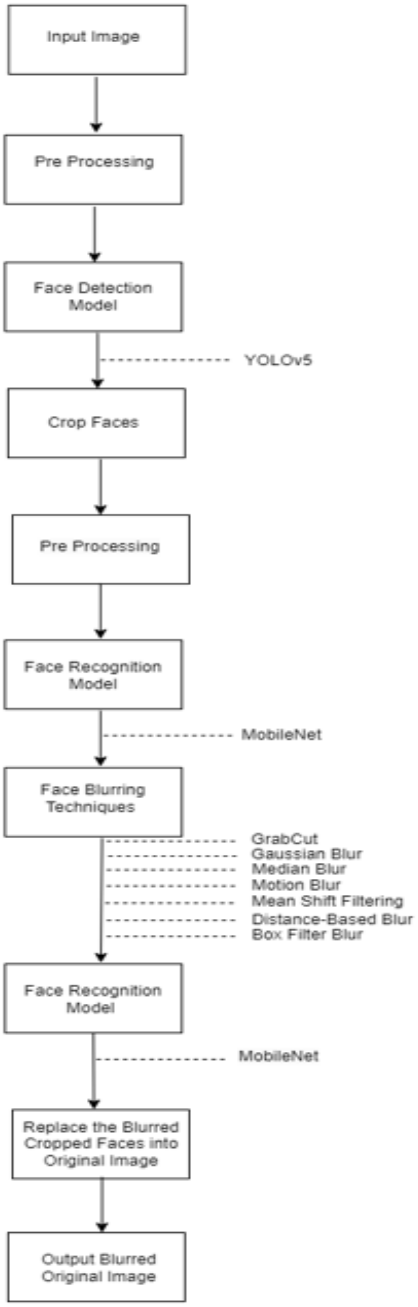


Figure 13 The Yolov5/MobileNet Pipeline

The **Ten Phases outlined** in the above figure (The YOLOV5 for Face Detection and MobileNet for Face Recognition Project Pipeline) are as follows:

- **Input Image**: The process starts with an input image that is fed into the pipeline.
- **Pre-Processing:** Convert the image to RGB.
- **Face Detection Model**: The YOLOv5 model is used for detecting faces within the input image. It identifies the locations of faces and typically provides bounding boxes around them.
- **Crop Faces**: Once faces are detected, they are cropped from the original image using the bounding boxes provided by the face detection model. This step isolates the faces for more detailed processing.
- **Pre-Processing:** Resize the image to (224,224) and convert it to numpy array.
- **Face Recognition Model:** The cropped faces are then processed by the MobileNet model, which is used here to recognize the identities of the faces based on the features learned from a training dataset.
- **Face Blurring Techniques**: Various blurring techniques are applied to the recognized faces depending on the requirements or privacy settings. These techniques include GrabCut, Gaussian Blur, Median Blur, Motion Blur, Mean Shift Filtering, Distance-Based Blur, and Box Filter Blur. These techniques modify the visual characteristics of the faces to make them less recognizable.
- **Face Recognition Model (Post-Blurring):** After blurring, the faces are processed again by the MobileNet model. This step tests how effective the blurring techniques are at obscuring the identities, checking if the blurred faces can still be recognized by the model.
- **Replace the Blurred Cropped Faces into Original Image:** After the blurring process, the altered faces are placed back into their original positions within the image, replacing the clear, unblurred faces.
- **Output Blurred Original Image:** The final output is the original image with the faces blurred, maintaining the rest of the image intact. This image is then outputted from the system.

3.2.5 The YOLOV8 For Face Detection and MobileNet for Face Recognition Project Pipeline (Pipeline 4):

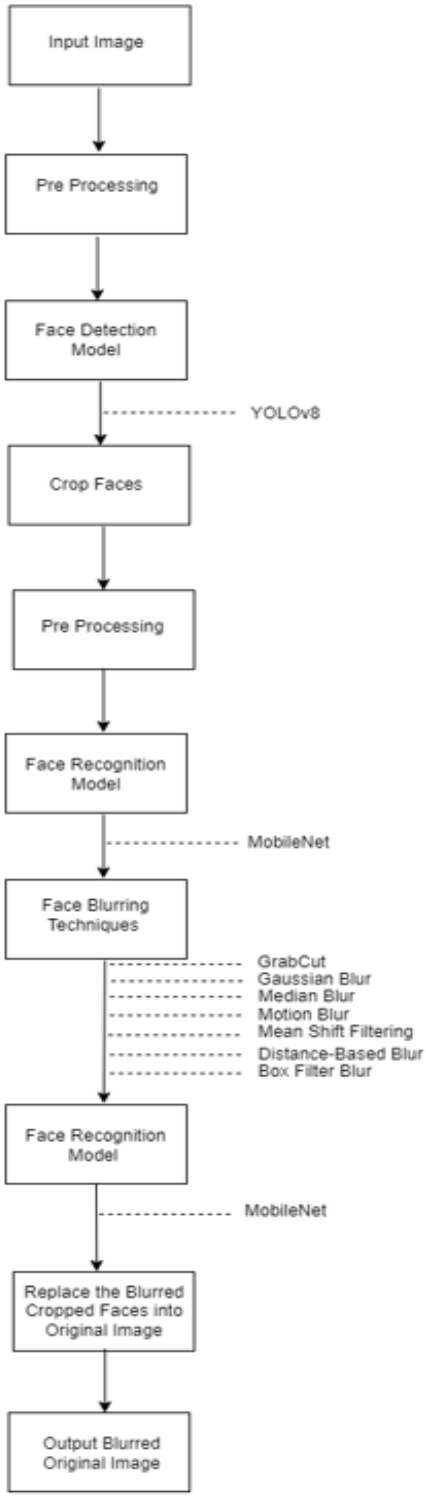


Figure 14 The Yolov8/MobileNet Pipeline

The **Ten Phases outlined** in the above figure (The YOLOV8 for Face Detection and MobileNet for Face Recognition Project Pipeline) are as follows:

- **Input Image**: The process starts with an input image that is fed into the pipeline.
- **Pre-Processing:** Convert the image to RGB.
- **Face Detection Model**: The YOLOv8 model is used for detecting faces within the input image. It identifies the locations of faces and typically provides bounding boxes around them.
- **Crop Faces**: Once faces are detected, they are cropped from the original image using the bounding boxes provided by the face detection model. This step isolates the faces for more detailed processing.
- **Pre-Processing:** Resize the image to (224,224) and convert it to numpy array.
- **Face Recognition Model:** The cropped faces are then processed by the MobileNet model, which is used here to recognize the identities of the faces based on the features learned from a training dataset.
- **Face Blurring Techniques**: Various blurring techniques are applied to the recognized faces depending on the requirements or privacy settings. These techniques include GrabCut, Gaussian Blur, Median Blur, Motion Blur, Mean Shift Filtering, Distance-Based Blur, and Box Filter Blur. These techniques modify the visual characteristics of the faces to make them less recognizable.
- **Face Recognition Model (Post-Blurring):** After blurring, the faces are processed again by the MobileNet model. This step tests how effective the blurring techniques are at obscuring the identities, checking if the blurred faces can still be recognized by the model.
- **Replace the Blurred Cropped Faces into Original Image:** After the blurring process, the altered faces are placed back into their original positions within the image, replacing the clear, unblurred faces.
- **Output Blurred Original Image:** The final output is the original image with the faces blurred, maintaining the rest of the image intact. This image is then outputted from the system.

4 Implementation

4.1 Pre-Processing:

4.1.1 YoloV5/YOLOV8 as a Face Detection Model

- Installing Required Libraries

Roboflow: Used for downloading datasets.

Ultralytics: Provides YOLOv5 and YOLOv8 models.

PyTorch: Used for deep learning tasks.

- Loading and Downloading the Dataset:

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="EwztU04tNkHQwiBBSShs")
project = rf.workspace("new-workspace-0sgfu").project("lfw_face")
version = project.version(1)
dataset = version.download("yolov8")
```

```
Requirement already satisfied: roboflow in /usr/local/lib/python3.10/
Requirement already satisfied: certifi==2023.7.22 in /usr/local/lib/p
```

Figure 15 Loading and downloading the dataset code snippet

Initializes Roboflow and downloads the "lfw_face" dataset in YOLOv8 format. The dataset includes images with corresponding labels for face detection tasks.

- Mounting Google Drive to Access Files Stored on the Drive
- Importing Required Libraries

- Creating Configuration File:

```
# Starting with a comment in config file
# defining newline variable for config file
newline='\n'
ln_1='# Train/val paths'+newline

# train, val path declaration
imgtrainpath = '/content/lfw_face-1/train/images'
imgvalpath = '/content/lfw_face-1/valid/images'
ln_2='train: ' +"'"+imgtrainpath+"'"+newline
ln_3='val: ' +"'" + imgvalpath+"'"+newline
ln_4=newline

# Optional additional paths
imgtrainlabelspath = '/content/lfw_face-1/train/labels'
imgvallabelspath = '/content/lfw_face-1/valid/labels'
ln_5='train_labels: ' +"'"+imgtrainlabelspath+"'"+newline
ln_6='val_labels: ' +"'"+imgvallabelspath+"'"+newline
ln_7=newline

# Classes
ln_8='# Classes'+newline
ln_9='names:'+newline
ln_10='  - face'+newline
ln_11='nc: 1'

config_lines=[ln_1, ln_2, ln_3, ln_4, ln_5, ln_6, ln_7, ln_8, ln_9, ln_10, ln_11]
# Creating path for config file
import os
curr_path = os.getcwd()
config_path=os.path.join(curr_path, 'config.yaml')
config_path

# Writing config file
with open(config_path, 'w') as f:
    f.writelines(config_lines)
```

Figure 16 Yolo Configuration File

This section prepares a configuration file for the YOLO model, defining paths for training and validation images and labels, and specifying class information. The file is saved as 'config. yaml'.

4.1.2 YoloV5 as a Face Recognition Model:

- Installing Required Libraries for YOLOV5
- Mounting Google Drive to access and store files conveniently, particularly for loading datasets and saving results
- Dataset Loading and Handling:

- Accessed the Avengers dataset stored on Google Drive. This dataset required preprocessing to include bounding box information as it originally contained only cropped face images without any annotations.

  - Loading the Utilized pretrained YOLOv5 model, that was fine-tuned on the LFW dataset for face detection.
  - Preprocessing Images through loading and converting them to RGB format.
  - Detecting Faces and Generating Bounding Boxes:
- Used the Utilized pretrained YOLOv5 model, that was fine-tuned on the LFW dataset for face detection, to detect faces then generate bounding boxes around faces in the Avengers images.
- Modified images to include bounding boxes and saved these as annotations for subsequent recognition task.

- Detect Faces:

```
def detect_faces(model, image_path):
    img = Image.open(image_path).convert("RGB")


    results = model(img)

    # Extract bounding boxes from results
    resB=results[0];
    bboxes = resB.boxes.xyxy # Assuming the first item in batch
    draw = ImageDraw.Draw(img)

  # Convert bounding boxes to the dictionary format
  # Prepare bounding box information
    formatted_bboxes = []
    for bbox in bboxes:
        xmin, ymin, xmax, ymax = bbox[:4]
        formatted_bboxes.append({
            "xmin": xmin.item(),
            "ymin": ymin.item(),
            "xmax": xmax.item(),
            "ymax": ymax.item(),
        })

        # Draw rectangle(BoundingBox) on the image
        draw.rectangle([xmin, ymin, xmax, ymax], outline="red", width=2)

    # Save the image with detections
    save_path = os.path.join('/content/drive/MyDrive/RESULTSSS', os.path.basename(image_path))
    img.save(save_path)

    return formatted_bboxes
```

Figure 17 Detect_Faces Function

- The function `detect_faces` is meant to input an image and detect faces using the pre-trained model. In the beginning, it opens an image and converts it to an RGB format to make

everything consistent. It then runs the image through the model for face detection and bounding boxes around those faces. In the implementation above, we assume the detected bounding boxes to be in the first item of the results batch. The function loops over the obtained bounding boxes, converts them into dictionary format, and appends them into a final list. It then locates the faces within the image by drawing rectangles according to the coordinates of the bounding box and saves a modified image with these drawn boxes to a given path on Google Drive. Finally, it returns the list of bounding boxes in the correct format. This process ensures that each marked face is detected and highlighted on the saved picture, making it easier for face recognition.

- Save the Detected Bounding Boxes Coordinates:

```
def save_bounding_boxes(bboxes, output_file_path):
    # Save the bounding box coordinates to a file
    with open(output_file_path, 'w') as f:
        for bbox in bboxes:
            xmin = bbox['xmin']
            ymin = bbox['ymin']
            xmax = bbox['xmax']
            ymax = bbox['ymax']
            f.write(f'{int(xmin)},{int(ymin)},{int(xmax)},{int(ymax)}\n')
```

Figure 18 Save_bounding_boxes function

The save_bounding_boxes method is the one that should save the bounding box coordinates in some text file. It accepts two arguments: bboxes and output_file_path. Here, bboxes is a list of dictionaries; each dictionary has bounding box coordinates. output_file_path is a path where the output file should be saved. First, it opens that given output file in write mode. It then loops through the list of bounding boxes and gets the values for xmin, ymin, xmax, and ymax, representing the bounding box's coordinates. For each bounding box, it writes a line into the file with the coordinates formatted as integers and using comma separators. This makes sure that each line of the file corresponds to a bounding box, making it human-readable and ready for any kind of further processing, including face recognition.

- Process Images:

```python
def process_images(input_directory, output_directory, model):
    # Ensure output directory exists
    os.makedirs(output_directory, exist_ok=True)
    # Iterate through each subdirectory (each person)
    for person in os.listdir(input_directory):
        person_path = os.path.join(input_directory, person)
        if os.path.isdir(person_path):  # Ensure it is a directory
            person_output_directory = os.path.join(output_directory, person)
            os.makedirs(person_output_directory, exist_ok=True)  # Create a directory for outputs for the person
            # Process each image in the person's directory
            for filename in os.listdir(person_path):
                if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
                    image_path = os.path.join(person_path, filename)
                    output_file_path = os.path.join(person_output_directory, f'{filename}_bboxes.txt')
                    # Detect faces
                    bboxes = detect_faces(model, image_path)
                    # Save bounding boxes
                    save_bounding_boxes(bboxes, output_file_path)
                    print(f"Processed {filename} in {person}")
```

Figure 19 Process_images function snippet

The function process_images processes images found in the input directory and records face boundary coordinates detected in the output directory. The input parameters are input_directory, output_directory, and model. This function would also check whether or not the output directory exists; if not, it would create a new one. Then, it iterates through each subdirectory in the input directory; each containing images of one person. For every person's directory, it checks if that is really a directory, and then it creates a corresponding directory in the output directory for that same person. Inside each person's directory, the function then iterates over the image files, specifically those with a .png, .jpg, or .jpeg extension. It constructs the path of each image and then applies the detect_faces function to detect faces. Then, the bounding box coordinates are dumped in a text file placed in the respective output directory. Finally, the script prints an indicator message on the console stating that an image has been processed. Thus, under this configuration, automatic feature detection and facial annotation will be organized according to classes.

Converting Bounding Boxes to YOLO Format.

- Developed a function to convert bounding box coordinates into YOLO format (center x, center y, width, height).
- Save Bounding Boxes Coordinates in YOLO Format
- Saved transformed annotations in a separate file to be used during model training.
- Dataset Splitting:
- The dataset was divided into training and validation sets using an 80:20 ratio.

- The split_dataset function is designed to divide a dataset of images and their corresponding bounding box labels into training and validation sets

```
import os
import shutil
from sklearn.model_selection import train_test_split
def split_dataset(images_base_path, labels_base_path, output_base_path, train_ratio=0.8):
```

Figure 20 Split_dataset Function

- Parameters:
  - images_base_path: The base directory containing subdirectories of images for different classes.
  - labels_base_path: The base directory containing subdirectories of label files that correspond to the images.
  - output_base_path: The directory where the split datasets (training and validation) will be saved.
  - train_ratio: The ratio of images to be used for training (default is 80% for training and 20% for validation).

Listing Characters (Classes) and Iterating Through Each Character:

```
characters = os.listdir(images_base_path)
class_id = 0
for character in characters:
    images_path = os.path.join(images_base_path, character)
    labels_path = os.path.join(labels_base_path, character)

    if not os.path.isdir(images_path) or not os.path.isdir(labels_path):
        continue
```

Figure 21 part of the splitting dataset function

- **characters**: List of subdirectories (each representing a different class) in the **images_base_path**.
- **class_id**: An identifier for each class, starting from 0.
- For each character (class), the paths to the images and labels are constructed.
- The loop skips if the paths are not directories.

Split Images into Training and Validation Sets:

```
image_files = [f for f in os.listdir(images_path) if f.endswith('.png')]
train_files, val_files = train_test_split(image_files, train_size=train_ratio, random_state=42)
```

Figure 22 part2 of splitting dataset function

- **image_files**: List of image files in the current class directory.
- **train_files, val_files**: The image files are split into training and validation sets based on the specified **train_ratio**.

- **move_files Function:**

```
def move_files(file_list, destination_folder):
    images_dest_path = os.path.join(destination_folder, 'images')
    labels_dest_path = os.path.join(destination_folder, 'labels')

    if not os.path.exists(images_dest_path):
        os.makedirs(images_dest_path)
    if not os.path.exists(labels_dest_path):
        os.makedirs(labels_dest_path)
```

Figure 23 move_files function snippet

- Parameters:
  - **file_list**: List of image files to be moved.
  - **destination_folder**: The target directory where the files will be moved.
- **images_dest_path, labels_dest_path**: Construct the destination paths for images and labels within the destination folder.
- The directories are created if they do not already exist.

Move Files and Convert Labels:

```python
for file in file_list:
    base_name = file
    image_src = os.path.join(images_path, file)
    label_src = os.path.join(labels_path, f"{base_name}_bboxes.txt")
    image_dst = os.path.join(images_dest_path, file)
    yolo_label_dst = os.path.join(labels_dest_path, f"{os.path.splitext(file)[0]}.txt")

    if not os.path.exists(image_src) or not os.path.exists(label_src):
        print(f"Warning: Missing file(s) for {base_name}")
        continue

    img = Image.open(image_src)
    img_width, img_height = img.size

    shutil.copy(image_src, image_dst)
    convert_and_save_yolo_labels(label_src, yolo_label_dst, img_width, img_height, class_id)
    print(f"Moved {file} to {destination_folder}, class_id: {class_id}")
```

Figure 24 moving files and converting labels snippet

For every file in the file_list, I create source paths for both the input image and label and then make destination paths for the output image and label. If the image or label does not exist, I print a warning and skip the file. I then open the image to get its dimensions, copy the image to output, and afterward convert the labels into YOLO format before saving it into a destination. Then, I print a message that the file has been successfully moved.

Define and Create Training and Validation Directories:

```python
train_output_path = os.path.join(output_base_path, 'train', character)
val_output_path = os.path.join(output_base_path, 'val', character)

move_files(train_files, train_output_path)
move_files(val_files, val_output_path)
class_id += 1
```

Figure 25 define and create training and val directories snippet

- **train_output_path, val_output_path**: Construct the output paths for training and validation data for the current class.
- **move_files**: The nested function is called to move the training and validation files to their respective directories.
- The **class_id** is incremented for the next class.

Run the split_dataset:

```
# Define paths
images_base_path = "/content/drive/MyDrive/GP_YOLOV5Model/avengers"
labels_base_path = "/content/drive/MyDrive/GP_YOLOV5Model/Boundary_Coordinates"
output_base_path = "/content/drive/MyDrive/fnew/newws"

# Split dataset
split_dataset(images_base_path, labels_base_path, output_base_path)
```

```
Moved tony_stark5.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
Moved tony_stark91.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
Moved tony_stark112.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
Moved tony_stark52.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
Moved tony_stark49.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
```

Figure 26 running the split_dataset function snippet

- **Define paths**: Set the paths for the images, labels, and output.
- **split_dataset**: Call the function to split the dataset into training and validation sets

- **Create YAML File:**

newfs.yaml

```
train: /content/drive/MyDrive/fnew/newws/train
val: /content/drive/MyDrive/fnew/newws/val

nc: 6
names: ['tony_stark (Robert Downey Jr.)','steve_rogers (Chris Evans)','Natasha_Romanoff (Scarlett Johansson)', 'bruce_banner (Mark Ruffalo)', 'clint_barton (Jeremy Renner)', 'thor (Chris Hemsworth)']
```

Figure 27 create the yaml file snippet

- **Verifying Dataset Structure:**

```python
import os
import json

# Paths to the training and validation datasets
# Path to the training and validation datasets
train_dir = '/content/drive/MyDrive/fnew/newws/train'
val_dir = '/content/drive/MyDrive/fnew/newws/val'

def verify_dataset_structure(data_dir):
    issues = []
    class_names = os.listdir(data_dir)
    for class_name in class_names:
        class_dir = os.path.join(data_dir, class_name)
        if os.path.isdir(class_dir):
            image_dir = os.path.join(class_dir, 'images')
            label_dir = os.path.join(class_dir, 'labels')
            if not os.path.exists(image_dir) or not os.path.exists(label_dir):
                issues.append(f"Missing 'images' or 'labels' directory for class: {class_name}")
                continue
            image_files = os.listdir(image_dir)
            label_files = os.listdir(label_dir)
            if len(image_files) != len(label_files):
                issues.append(f"Mismatch in the number of images and labels for class: {class_name}")
                continue
            for label_file in label_files:
                label_path = os.path.join(label_dir, label_file)
                with open(label_path, 'r') as f:
                    try:
                        label_data = f.read().strip()
                        if not label_data:
                            issues.append(f"Empty label file: {label_path}")
                    except Exception as e:
                        issues.append(f"Error reading label file: {label_path}, error: {str(e)}")
    return issues

train_issues = verify_dataset_structure(train_dir)
val_issues = verify_dataset_structure(val_dir)

if train_issues or val_issues:
    print("Issues found in the dataset:")
    print("\n".join(train_issues + val_issues))
else:
    print("No issues found in the dataset.")
```

Figure 28 verify_dataset_structure function snippet

6.1.1 YoloV8 as a Face Recognition Model:

- Installing the necessary libraries for YOLOv8.
- Mounting Google Drive to access and store files conveniently, in particular when loading datasets and saving results.
- **Dataset Loading and Handling:**
  - Accessed the Avengers dataset from Google Drive; this dataset had to be preprocessed since it only originally had cropped face images without any annotations.
- **Loading the model** pre-trained on the YOLOv8 model, fine-tuned on the LFW dataset for face detection.

- **Image Pre-Processing:**
  - Load and conversion of images to RGB.
- **Facial Detection and Bounding Box Generation:**
  - For facial detection in Avenger dataset images, fine-tuning was done over the pretrained YOLOv8 model on the LFW dataset. Bounding boxes were generated around the face areas in this detection.
  - Modified images with bounding boxes and saved annotations were saved for future recognition tasks

- **Detect Faces:**

```python
def detect_faces(model, image_path):
    img = Image.open(image_path).convert("RGB")


    results = model(img)

    # Extract bounding boxes from results
    resB=results[0];
    bboxes = resB.boxes.xyxy # Assuming the first item in batch
    draw = ImageDraw.Draw(img)

  # Convert bounding boxes to the dictionary format
  # Prepare bounding box information
    formatted_bboxes = []
    for bbox in bboxes:
        xmin, ymin, xmax, ymax = bbox[:4]
        formatted_bboxes.append({
            "xmin": xmin.item(),
            "ymin": ymin.item(),
            "xmax": xmax.item(),
            "ymax": ymax.item(),
        })

        # Draw rectangle(BoundingBox) on the image
        draw.rectangle([xmin, ymin, xmax, ymax], outline="red", width=2)

    # Save the image with detections
    save_path = os.path.join('/content/drive/MyDrive/RESULTSSS', os.path.basename(image_path))
    img.save(save_path)

    return formatted_bboxes
```

Figure 29 yolov8 detect_faces function snippet

- The function `detect_faces` is meant to input an image and detect faces using the pre-trained model. In the beginning, it opens an image and converts it to an RGB format to make everything consistent. It then runs the image through the model for face detection and bounding boxes around those faces. In the implementation above, we assume the detected bounding boxes to be in the first item of the results batch. The function loops over the obtained bounding boxes, converts them into dictionary format, and appends them into a final list. It then locates the faces within the image by drawing rectangles according to the coordinates of the bounding box and saves a modified image with these drawn boxes to a given path on Google Drive. Finally, it returns the list of bounding boxes in the correct format. This process ensures that each marked face is detected and highlighted on the saved picture, making it easier for face recognition.

- Save the Detected Bounding Boxes Coordinates:

```python
def save_bounding_boxes(bboxes, output_file_path):
    # Save the bounding box coordinates to a file
    with open(output_file_path, 'w') as f:
        for bbox in bboxes:
            xmin = bbox['xmin']
            ymin = bbox['ymin']
            xmax = bbox['xmax']
            ymax = bbox['ymax']
            f.write(f'{int(xmin)},{int(ymin)},{int(xmax)},{int(ymax)}\n')
```

Figure 30 yolov8 save_bounding_boxes function snippet

The save_bounding_boxes method is the one that should save the bounding box coordinates in some text file. It accepts two arguments: bboxes and output_file_path. Here, bboxes is a list of dictionaries; each dictionary has bounding box coordinates. output_file_path is a path where the output file should be saved. First, it opens that given output file in write mode. It then loops through the list of bounding boxes and gets the values for xmin, ymin, xmax, and ymax, representing the bounding box's coordinates. For each bounding box, it writes a line into the file with the coordinates formatted as integers and using comma separators. This makes sure that each line of the file corresponds to a bounding box, making it human-readable and ready for any kind of further processing, including face recognition.

- Process Images:

```python
def process_images(input_directory, output_directory, model):
    # Ensure output directory exists
    os.makedirs(output_directory, exist_ok=True)
    # Iterate through each subdirectory (each person)
    for person in os.listdir(input_directory):
        person_path = os.path.join(input_directory, person)
        if os.path.isdir(person_path):  # Ensure it is a directory
            person_output_directory = os.path.join(output_directory, person)
            os.makedirs(person_output_directory, exist_ok=True)  # Create a directory for outputs for the person
            # Process each image in the person's directory
            for filename in os.listdir(person_path):
                if filename.lower().endswith(('.png', '.jpg', '.jpeg')):
                    image_path = os.path.join(person_path, filename)
                    output_file_path = os.path.join(person_output_directory, f'{filename}_bboxes.txt')
                    # Detect faces
                    bboxes = detect_faces(model, image_path)
                    # Save bounding boxes
                    save_bounding_boxes(bboxes, output_file_path)
                    print(f"Processed {filename} in {person}")
```

Figure 31 yolov8 process_images function snippet

The function process_images processes images found in the input directory and records face boundary coordinates detected in the output directory. The input parameters are input_directory, output_directory, and model. This function would also check whether or not the output directory exists; if not, it would create a new one. Then, it iterates through each subdirectory in the input directory; each containing images of one person. For every person's directory, it checks if that is really a directory, and then it creates a corresponding directory in the output directory for that same person. Inside each person's directory, the function then iterates over the image files, specifically those with a .png, .jpg, or .jpeg extension. It constructs the path of each image and then applies the detect_faces

function to detect faces. Then, the bounding box coordinates are dumped in a text file placed in the respective output directory. Finally, the script prints an indicator message on the console stating that an image has been processed. Thus, under this configuration, automatic feature detection and facial annotation will be organized according to classes.

- Converting Bounding Boxes to YOLO Format:
  - Developed a function to convert bounding box coordinates into YOLO format (center x, center y, width, height).
- Save Bounding Boxes Coordinates in YOLO Format:
  - Saved transformed annotations in a separate file to be used during model training.

- Dataset Splitting:
  - The dataset was divided into training and validation sets using an 80:20 ratio.

The split_dataset function is designed to divide a dataset of images and their corresponding bounding box labels into training and validation sets

```
import os
import shutil
from sklearn.model_selection import train_test_split
def split_dataset(images_base_path, labels_base_path, output_base_path, train_ratio=0.8):
```

Figure 32 yolov8 split_dataset function snippet

- Parameters:
  - images_base_path: The base directory containing subdirectories of images for different classes.
  - labels_base_path: The base directory containing subdirectories of label files that correspond to the images.
  - output_base_path: The directory where the split datasets (training and validation) will be saved.
  - train_ratio: The ratio of images to be used for training (default is 80% for training and 20% for validation).

- Listing Characters (Classes) and Iterating Through Each Character:

```
characters = os.listdir(images_base_path)
class_id = 0
for character in characters:
    images_path = os.path.join(images_base_path, character)
    labels_path = os.path.join(labels_base_path, character)

    if not os.path.isdir(images_path) or not os.path.isdir(labels_path):
        continue
```

Figure 33 yolov8 split_dataset part2 function snippet

- **characters**: List of subdirectories (each representing a different class) in the **images_base_path**.
- **class_id**: An identifier for each class, starting from 0.
- For each character (class), the paths to the images and labels are constructed.
- The loop skips if the paths are not directories.

- **Split Images into Training and Validation Sets:**

```
image_files = [f for f in os.listdir(images_path) if f.endswith('.png')]
train_files, val_files = train_test_split(image_files, train_size=train_ratio, random_state=42)
```

Figure 34 yolov8 split images and val sets snippet p3

- **image_files**: List of image files in the current class directory.
- **train_files, val_files**: The image files are split into training and validation sets based on the specified **train_ratio**.

- **move_files Function:**

```python
def move_files(file_list, destination_folder):
    images_dest_path = os.path.join(destination_folder, 'images')
    labels_dest_path = os.path.join(destination_folder, 'labels')

    if not os.path.exists(images_dest_path):
        os.makedirs(images_dest_path)
    if not os.path.exists(labels_dest_path):
        os.makedirs(labels_dest_path)
```

Figure 35 yolov8 move_files function snippet p4

- Parameters:
  - **file_list**: List of image files to be moved.
  - **destination_folder**: The target directory where the files will be moved.
- **images_dest_path, labels_dest_path**: Construct the destination paths for images and labels within the destination folder.
- The directories are created if they do not already exist.

- **Move Files and Convert Labels:**

```python
for file in file_list:
    base_name = file
    image_src = os.path.join(images_path, file)
    label_src = os.path.join(labels_path, f"{base_name}_bboxes.txt")
    image_dst = os.path.join(images_dest_path, file)
    yolo_label_dst = os.path.join(labels_dest_path, f"{os.path.splitext(file)[0]}.txt")

    if not os.path.exists(image_src) or not os.path.exists(label_src):
        print(f"Warning: Missing file(s) for {base_name}")
        continue

    img = Image.open(image_src)
    img_width, img_height = img.size

    shutil.copy(image_src, image_dst)
    convert_and_save_yolo_labels(label_src, yolo_label_dst, img_width, img_height, class_id)
    print(f"Moved {file} to {destination_folder}, class_id: {class_id}")
```

Figure 36 yolov8 move files and convert labels snippet p5

For every file in the file_list, I create source paths for both the input image and label and then make destination paths for the output image and label. If the image or label does not exist, I print a warning and skip the file. I then open the image to get its dimensions, copy the image to output, and afterward convert the labels into YOLO format before saving it into a destination. Then, I print a message that the file has been successfully moved.

- **Define and Create Training and Validation Directories:**

```
train_output_path = os.path.join(output_base_path, 'train', character)
val_output_path = os.path.join(output_base_path, 'val', character)

move_files(train_files, train_output_path)
move_files(val_files, val_output_path)
class_id += 1
```

Figure 37 yolov8 defining and creating training and val directories p6

- **train_output_path, val_output_path**: Construct the output paths for training and validation data for the current class.
- **move_files**: The nested function is called to move the training and validation files to their respective directories.
- The **class_id** is incremented for the next class.

- **Run the split_dataset:**

```
# Define paths
images_base_path = "/content/drive/MyDrive/GP_YOLOV5Model/avengers"
labels_base_path = "/content/drive/MyDrive/GP_YOLOV5Model/Boundary_Coordinates"
output_base_path = "/content/drive/MyDrive/fnew/newws"

# Split dataset
split_dataset(images_base_path, labels_base_path, output_base_path)
```

```
Moved tony_stark5.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
Moved tony_stark91.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
Moved tony_stark112.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
Moved tony_stark52.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
Moved tony_stark49.png to /content/drive/MyDrive/new/train/tony_stark (Robert Downey Jr.), class_id: 0
```

Figure 38 running split dataset function snippet p7

- **Define paths**: Set the paths for the images, labels, and output.
- **split_dataset**: Call the function to split the dataset into training and validation sets

- **Create YAML File:**

newfs.yaml

```yaml
train: /content/drive/MyDrive/fnew/newws/train
val: /content/drive/MyDrive/fnew/newws/val

nc: 6
names: ['tony_stark (Robert Downey Jr.)','steve_rogers (Chris Evans)','Natasha_Romanoff (Scarlett Johansson)', 'bruce_banner (Mark Ruffalo)', 'clint_barton (Jeremy Renner)', 'thor (Chris Hemsworth)']

```

Figure 39 create yaml file

- **Verifying Dataset Structure:**

```python
import os
import json

# Paths to the training and validation datasets
# Path to the training and validation datasets
train_dir = '/content/drive/MyDrive/fnew/newws/train'
val_dir = '/content/drive/MyDrive/fnew/newws/val'

def verify_dataset_structure(data_dir):
    issues = []
    class_names = os.listdir(data_dir)
    for class_name in class_names:
        class_dir = os.path.join(data_dir, class_name)
        if os.path.isdir(class_dir):
            image_dir = os.path.join(class_dir, 'images')
            label_dir = os.path.join(class_dir, 'labels')
            if not os.path.exists(image_dir) or not os.path.exists(label_dir):
                issues.append(f"Missing 'images' or 'labels' directory for class: {class_name}")
                continue
            image_files = os.listdir(image_dir)
            label_files = os.listdir(label_dir)
            if len(image_files) != len(label_files):
                issues.append(f"Mismatch in the number of images and labels for class: {class_name}")
                continue
            for label_file in label_files:
                label_path = os.path.join(label_dir, label_file)
                with open(label_path, 'r') as f:
                    try:
                        label_data = f.read().strip()
                        if not label_data:
                            issues.append(f"Empty label file: {label_path}")
                    except Exception as e:
                        issues.append(f"Error reading label file: {label_path}, error: {str(e)}")
    return issues

train_issues = verify_dataset_structure(train_dir)
val_issues = verify_dataset_structure(val_dir)

if train_issues or val_issues:
    print("Issues found in the dataset:")
    print("\n".join(train_issues + val_issues))
else:
    print("No issues found in the dataset.")
```

Figure 40 verify_dataset_structure function snippet(yolov8)

6.1.2 Mobile Net for Face Recognition:

- **Dataset Loading and Initial Handling:**

- Accessed the Avengers dataset stored on Google Drive. This dataset needed preprocessing to add bounding box information, as it originally had only cropped face images without annotations.

- **Bounding Box Annotation:**

- Used a pretrained YOLOv5 model, fine-tuned on the LFW dataset, to create bounding boxes around faces in the Avengers images.

- Modified images to include these bounding boxes and saved them as annotations for later recognition tasks.

- **Image Preprocessing:**

- **Image Loading and Conversion:** Images are loaded and converted to RGB format before processing, ensuring consistent input for both detection and recognition models.
- **Resizing:** All images are resized to 224x224 pixels, the input size expected by MobileNet, to provide the model with consistent input dimensions.
- **Normalization:** As part of the MobileNet preprocessing, images undergo normalization to meet the network's input requirements, typically involving scaling pixel values to a range the model works with effectively.

- **Encoding and Splitting:**

- Label Encoding: The classes (characters) are encoded using a OneHotEncoder. This conversion is essential for the model to perform multi-class classification.

```
[ ] X=[]
    Y=[]
    for x,y in face:
        X.append(x)
        Y.append(y)
```

```
[ ] x=np.array(X)
    y=np.array(Y)
    print(x.shape,y.shape)
```

```
(916, 224, 224, 3) (916,)
```

```
[ ] encoder = OneHotEncoder(dtype=np.float32)
```

```
[ ] # Fit and transform the data
    encoded_Y = encoder.fit_transform(y.reshape(-1, 1))
    encoded_Y = encoded_Y.toarray()
```

```
[ ] encoded_Y.shape
```

```
(916, 6)
```

Figure 41 label encoding snippet (mobileNet)

- Train-Test Split: The dataset is divided into training and testing subsets to assess the model's performance. The split ratio is 70:30.

```
[ ] X1_train, X1_test,Y1_train,Y1_test = train_test_split(x,encoded_Y, test_size=0.3, random_state=30)
```

Figure 42 train-test split snippet

7 Blurring Techniques:

7.1 GrabCut/Gaussian

GrabCut Description: GrabCut is a very high-level image segmentation algorithm in which the foreground is separated from an image. One of these iterative optimization strategies concerns graph cuts and deals with object classification for image segmentation: the process separates the images by modeling foreground and background using Gaussian Mixture Models. Initially, a bounding box is drawn over the object of interest, and after that the algorithm keeps iterating until it gets the correct result. [18].

```
def blur_faces(image, bbox):
    x, y, w, h = bbox
    mask = np.zeros(image.shape[:2], np.uint8)
    bgd_model = np.zeros((1, 65), np.float64)
    fgd_model = np.zeros((1, 65), np.float64)
    cv2.grabCut(image, mask, (x, y, w, h), bgd_model, fgd_model, 5, cv2.GC_INIT_WITH_RECT)
    mask2 = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')
    segmented_image = cv2.GaussianBlur(image * mask2[:, :, np.newaxis], (51, 51), 0)
    image = image * (1 - mask2[:, :, np.newaxis]) + segmented_image
    return image
```

Figure 43 grabcut(gaussian) function

The given code defines a function, blur_faces, that blurs a specific region of an image. The area is represented by a bounding box, whose pixels have been captured using the GrabCut algorithm and initialized models for the mask and background/foreground. The GrabCut algorithm then processes the image to return only the face and background, with which the initialized mask is segmented. Then, Gaussian blur is applied to that segmented area. This blurred face is then merged into the background of the original image, with the face blurred, but the rest of the image remains as it was. Now, we can easily blur faces for privacy protection or artistic effect.

7.2 Gaussian Blurring

Description: Gaussian blurring can be an effective linear filtering technique. It can process an image to reduce noise and lower the detail of an image. This filter gives weights for surrounding pixels by the Gaussian distribution function, where the main weightage goes to the central pixel and then further to other pixels. The effect shown is smoothly blurred[19].

```
def blur_faces(image, bbox):
    x, y, w, h = bbox
    # Apply Gaussian blur directly over the bounding box region
    face_region = image[y:y+h, x:x+w]
    blurred_face = cv2.GaussianBlur(face_region, (51, 51), 30)  # You can adjust the kernel size and sigma values here
    image[y:y+h, x:x+w] = blurred_face
    return image
```

Figure 44 gaussian blur function

This code snippet defines a function **blur_faces** that applies Gaussian blur to a specified region (bounding box) of an image. The purpose is to blur faces within the bounding box, effectively obscuring them while leaving the rest of the image intact. Gaussian blur is commonly used for smoothing or obscuring parts of an image.

7.3 Median Blurring

Description: Median blurring is a type of non-linear filtering to reduce the noise in an image. It replaces every pixel in the image with the median intensity value found in its environment. This method is very effective against "salt and pepper" noise and edges[20].

```
def blur_faces(image, bbox):
    x, y, w, h = bbox
    face_region = image[y:y+h, x:x+w]
    blurred_face = cv2.medianBlur(face_region, 25)  # Kernel size can be varied
    image[y:y+h, x:x+w] = blurred_face
    return image
```

Figure 45 Median Blur Function

The blur_faces function blurs a particular region of an image—usually, it is within the bounding box covering a face. Using the bounding box coordinates, we extract a face region from an image. In the example below, we median blur the extracted face region using OpenCV's inbuilt function, cv2.medianBlur, with the kernel size as 25. Finally, we place this blurred face region back into the initial image to obscure the face. The function returns the modified image.

7.4 Box Filter Blurring

Description: A box filter is a blurring technique in which every pixel of an output image gets an average value from its neighboring pixels of an input image. It is a straightforward filter that uses the average or mean of pixel values over an image in a much smaller matrix; a kernel is used for that purpose. The method finds many applications: noise and detail reduction, to get a more linearly blurred look[21].

```
def blur_faces(image, bbox):
    x, y, w, h = bbox
    face_region = image[y:y+h, x:x+w]
    kernel_size = 15
    kernel = np.zeros((kernel_size, kernel_size))
    kernel[int((kernel_size-1)/2), :] = np.ones(kernel_size)
    kernel = kernel / kernel_size
    blurred_face = cv2.filter2D(face_region, -1, kernel)
    image[y:y+h, x:x+w] = blurred_face
    return image
```

Figure 46 Box Filter Function

The blur_faces function performs facial region localization and box filtering using a 15×15 kernel. The kernel is designed so that in just one dimension, it averages the pixel values, thereby giving the face region a blurred look. Finally, the filtered face region is pasted back into the original image, showing a blurred face with the rest of the image intact.

7.5 Mean Shift Blurring

Description: Mean-shift filtering is a nonparametric clustering technique for segmentation and image smoothing. The basic idea is that each pixel is uniformly shifted concerning the average of the pixels within the kernel window. It blurs an image but keeps the edges and the result's boundaries much finer than basic blurring methods[22].

```
def blur_faces(image, bbox):
    x, y, w, h = bbox
    face_region = image[y:y+h, x:x+w]
    blurred_face = cv2.pyrMeanShiftFiltering(face_region, 21, 51)
    image[y:y+h, x:x+w] = blurred_face
    return image
```

Figure 47 MeanShift Filtering Function

The above function recognizes the face region from the bounding box co-ordinates. It applies M-shift filtering, taking a spatial window radius of 21 along with a color window radius, which is set to 51. The treatment that this M-shift filter applies to the face region is smoothing out the same with the retention of most of the edges, making it appear cleaned and appealingly blurred. The filtered face region is placed back into the original image, resulting in a blurred face, but edge details are preserved.

7.6 Distance-Based Blurring

Description: Distance-based blur is a method in which the amount of blurriness of an image incrementally increases for each pixel with an increase in distance from a central point. In most cases, it occurs within an image or a considered region of interest. This gives a focus effect where the central part is considerably clear, and gradually, the periphery becomes clearer[23].

```
def blur_faces(image, bbox):
    x, y, w, h = bbox

    # Define the face region within the bounding box
    face_region = image[y:y+h, x:x+w]

    # Create a copy of the face region to apply blur
    blurred_face = face_region.copy()

    # Calculate the center of the face region
    center_x, center_y = w // 2, h // 2

    # Apply distance-based blur
    for i in range(face_region.shape[1]):
        for j in range(face_region.shape[0]):
            dx, dy = (i - center_x) / (w / 2), (j - center_y) / (h / 2)
            dist = np.sqrt(dx*dx + dy*dy)
            k = int(dist * 30)  # Scale the blur amount based on distance from the center
            if k > 0:
                blurred_face[j, i] = np.mean(face_region[max(j-k, 0):min(j+k+1, h), max(i-k, 0):min(i+k+1, w)], axis=(0, 1))

    # Replace the original face region in the image with the blurred version
    image[y:y+h, x:x+w] = blurred_face

    return image  # Return the modified image
```

Figure 48 Distance-Based Function

The above function applies a distance-based blur effect to a given region of the face. The first step is to crop the face region from an image using the provided bounding box coordinates. A copy of this face region is made so that any distorting operation is not done on the original data. Then, the center of the face region. For every pixel within the face region, the function calculates the distance from the center. If this distance is small, the pixel will be blurred slightly; the intensity increases at a point as the magnitude of the distance increases. Averaging of the surrounding pixels realizes the blur according to the calculated distance. Finally, the region of the face is then placed back into the original photo to replace the face region. This gives a focus effect so that the center is clear and the periphery becomes more-out-of-focus.

7.7 Radial Blur

Description: Radial blur is the technique that gives one a feeling of increasing blur from the center to the outer part of an image. One gets the impression of either movement or focus, with the sharp point being at the center and the surrounding peripheral parts increasingly becoming blurred. This may be a good technique to take a focusing approach in the center of a photo, perhaps the face, without giving attention to the environment surrounding it[24].

```python
def blur_faces(image, bbox):
    x, y, w, h = bbox
    face_region = image[y:y+h, x:x+w]
    blurred_face = face_region.copy()
    center_x, center_y = w // 2, h // 2
    for i in range(face_region.shape[1]):
        for j in range(face_region.shape[0]):
            dx, dy = (i-center_x)/(w/2), (j-center_y)/(h/2)
            dist = np.sqrt(dx*dx + dy*dy)
            k = int(dist * 30)
            if k > 0:
                blurred_face[j, i] = np.mean(face_region[max(j-k, 0):min(j+k+1, h), max(i-k, 0):min(i+k+1, w)], axis=(0, 1))
    image[y:y+h, x:x+w] = blurred_face
    return image
```

Figure 49 Radial Blur function

The above code for radial blurring begins by extracting the facial area from the image based on passed bounding box coordinates. That face region would be a part of the image where the face lies. After this, a copy is made of this face region, upon which the blur effect is applied while keeping the original immediate face data unscathed. Blur is calculated at the center point of the face region. It considers this to be the starting point for the radial blur effect. We then loop through each pixel in the face region and find how far the pixel's distance is from the center. This distance is used to define the intensity of the blur, where the distance from the center higher the intensity will be. So the average of surrounding pixels within some radius, which depends on that distance, is taken. The final region of face blur is now pasted back to the original image, thereby replacing the non-blurred face region to complete face blurring.

Motion Blur

Description: The motion blur filter brings in an effect that a certain object or camera was in motion. It is obtained by averaging pixel values along the direction, which generally offers just this kind of streaking look in an image. The blur effect provides the streaking look and will be seen in commonly taken photographs of speedy things or when the camera moves very quickly during capture. It is also possible to change the kernel used in the convolution process hence controlling the extent and direction of blur[25].

```python
def blur_faces(image, bbox, kernel_size=15):
    x, y, w, h = bbox
    face_region = image[y:y+h, x:x+w]

    # Create a motion blur kernel
    kernel = np.zeros((kernel_size, kernel_size))
    kernel[int((kernel_size-1)/2), :] = np.ones(kernel_size)
    kernel = kernel / kernel_size

    # Apply the motion blur kernel to the face region
    blurred_face = cv2.filter2D(face_region, -1, kernel)

    # Replace the original face region with the blurred version
    image[y:y+h, x:x+w] = blurred_face

    return image
```

Figure 50 Motion Blur Function

The above code applies a motion blur effect to a specific face region in an image. It retrieves an area of a face from a bounding box supplied through its coordinates x, y, w, h, and then creates a kernel that does motion blur; it uses a 15x15 kernel with a '1' in the middle row elements, which will produce horizontal motion blur. Later, this kernel gets normalized such that the sum of the elements is 1; this is done so that convolution doesn't change the overall brightness of the image. Now, the motion blur is applied to the face region using convolution by cv2.filter2D, which applies the created kernel to the face region, which presents a motion blur effect. Above all, the replaced original face region—by blurred face—is passed back, which is the modified image. By doing this, the face is blurred and the other areas in the input image remain the same; hence, a motion-blurring effect is centered on the face region specified.

Testing and evaluation

7.8 Testing

7.8.1 YoloV8 for Face Detection

```
[ ]  !yolo detect train data='/content/lfw_face-1/data.yaml' model=yolov8n.pt epochs=15 imgsz=640 device=0
```

Figure 51 Yolov8 training snippet

Labelled Testing batch:

Figure 52 yolov8 labelled testing batch

Predicted Testing Batch:

Figure 53 yolov8 predicted testing batch

7.8.2 Yolov5 for Face Detection

```
[ ] !yolo mode=check task=detect mode=train model=yolov5s.pt data='/content/lfw_face-1/data.yaml' epochs=30 imgsz=640 batch=64 device=0
```

Figure 54 Yolov5 training snippet

Labelled Testing Batch:

Figure 55 yolov5 Labeled testing batch

Predicted Testing Batch:

Figure 56 yolov5 predicted testing batch

7.8.3 MobileNet for Face Recognition

```
predictions = model.predict(X1_test)
predicted_classes = np.argmax(predictions, axis=1)
```

Figure 57 mobilenet predict snippet

```
true_classes_indices = np.argmax(Y1_test, axis=1)
```

```
class_names = ['Bruce Banner', 'Clint Barton', 'Natasha Romanoff', 'Steve Rogers', 'Thor', 'Tony Stark']
predicted_labels = [class_names[idx] for idx in predicted_classes_indices]
true_labels = [class_names[idx] for idx in true_classes_indices]
```

Figure 58 Mapping true and predicted Labels to Class Names

The above code first uses the trained model to predict classes for the test dataset ` X1_test` and stores predictions in ` predictions`. Then, for every test sample, it determines the predicted class by selecting the maximum value index in each prediction vector and stores the same in `predicted_classes`. Classically, the actual classes for the test samples are derived from ` Y1_test`, and these are stored in `true_classes_indices`. Then, one uses the indices to map onto human-readable class names with the

help of the class_names list. The following is produced: predicted_labels and true_labels lists the predicted and actual classes for each testing sample.

```
# Assuming predicted_labels and true_labels are lists containing the predicted and actual labels respectively
correct_count = 0
incorrect_count = 0

# Display the results and count correct/incorrect predictions
for i in range(len(predicted_labels)):
    print(f"Image {i+1}: Predicted - {predicted_labels[i]}, Actual - {true_labels[i]}")
    if predicted_labels[i] == true_labels[i]:
        correct_count += 1
    else:
        incorrect_count += 1

# Print out the totals
print(f"Total correct predictions: {correct_count}")
print(f"Total incorrect predictions: {incorrect_count}")
```

Figure 59 Code for counting correct and Incorrect Predictions

The code above goes through the predicted and actual labels and compares them to count how many correct and incorrect predictions have been made. It first initializes two counters, which are correct_count and incorrect_count, to 0. For every image, it prints the predictions and accurate labels. If the predicted label coincides with the truth label, then "correct" will be increased; if not, "incorrect" will be increased. After that, the loop prints the total number of correct and incorrect predictions, which is a straightforward summary of how the model performed on the test set.

```
Image 1: Predicted - Thor, Actual - Thor
Image 2: Predicted - Tony Stark, Actual - Tony Stark
Image 3: Predicted - Clint Barton, Actual - Clint Barton
Image 4: Predicted - Steve Rogers, Actual - Natasha Romanoff
Image 5: Predicted - Thor, Actual - Thor
Image 6: Predicted - Steve Rogers, Actual - Steve Rogers
Image 7: Predicted - Natasha Romanoff, Actual - Natasha Romanoff
Image 8: Predicted - Tony Stark, Actual - Tony Stark
Image 9: Predicted - Thor, Actual - Thor
Image 10: Predicted - Steve Rogers, Actual - Steve Rogers
Image 11: Predicted - Thor, Actual - Thor
Image 12: Predicted - Bruce Banner, Actual - Bruce Banner
Image 13: Predicted - Bruce Banner, Actual - Bruce Banner
Image 14: Predicted - Tony Stark, Actual - Tony Stark
Image 15: Predicted - Natasha Romanoff, Actual - Natasha Romanoff
Image 16: Predicted - Clint Barton, Actual - Clint Barton
Image 17: Predicted - Tony Stark, Actual - Tony Stark
Image 18: Predicted - Tony Stark, Actual - Tony Stark
Image 19: Predicted - Steve Rogers, Actual - Steve Rogers
Image 20: Predicted - Thor, Actual - Thor
Image 21: Predicted - Thor, Actual - Thor
Image 22: Predicted - Thor, Actual - Thor
Image 23: Predicted - Clint Barton, Actual - Clint Barton
Image 24: Predicted - Steve Rogers, Actual - Steve Rogers
Image 25: Predicted - Steve Rogers, Actual - Steve Rogers
Image 26: Predicted - Natasha Romanoff, Actual - Natasha Romanoff
Image 27: Predicted - Clint Barton, Actual - Clint Barton
Image 28: Predicted - Natasha Romanoff, Actual - Natasha Romanoff
Image 29: Predicted - Steve Rogers, Actual - Steve Rogers
Image 30: Predicted - Bruce Banner, Actual - Bruce Banner
Image 31: Predicted - Tony Stark, Actual - Tony Stark
Image 32: Predicted - Tony Stark, Actual - Tony Stark
Image 33: Predicted - Steve Rogers, Actual - Steve Rogers
Image 34: Predicted - Bruce Banner, Actual - Bruce Banner
Image 35: Predicted - Steve Rogers, Actual - Steve Rogers
Image 36: Predicted - Natasha Romanoff, Actual - Natasha Romanoff
Image 37: Predicted - Natasha Romanoff, Actual - Natasha Romanoff
Image 38: Predicted - Bruce Banner, Actual - Bruce Banner
Image 39: Predicted - Tony Stark, Actual - Steve Rogers
```

Figure 60 Output of Model Predictions: Comparison of predicted and actual Labels

```
Image 274: Predicted - Tony Stark, Actual - Tony Stark
Image 275: Predicted - Bruce Banner, Actual - Bruce Banner
Total correct predictions: 265
Total incorrect predictions: 10
```

Figure 61 Total number of correct and incorrect prediction

```
[ ] def load_and_preprocess_image(image_path):
        # Load the image file, resizing it to 224x224 pixels (as expected by the model)
        img = cv2.imread(image_path)
        img = cv2.resize(img, (224, 224))

        # Convert the image to a numpy array and expand dimensions to be (1, 224, 224, 3)
        img = np.expand_dims(img, axis=0)
        return img
```

Figure 62 load and preprocess function

The code above loads and preprocess the image through its path.

```python
def predict_image_class(image_path, model):
    # Load and preprocess the image
    img = load_and_preprocess_image(image_path)

    # Make predictions
    predictions = model.predict(img)

    predicted_class = np.argmax(predictions, axis=1)

    return predicted_class
```

Figure 63 predict image class function

The code above defines a function predict_image_class that accepts an image path and a trained model. It first loads and preprocesses the image based on the kind of function load_and_preprocess_image. Then, the predictions are made using the "predict" method of the preprocessed image. The predicted class is finally predicted and returned by finding the index of the maximum value of the predictions array. Finally, it returns the expected class.

```python
image_path = "/content/drive/MyDrive/GP_YOLOV5Model/cris10.jpg"
predicted_class = predict_image_class(image_path, model)

print("Predicted class index:", predicted_class)
```

```
1/1 [==============================] - 0s 49ms/step
Predicted class index: [5]
```

Figure 64 predicted class of a test image

7.8.4 YOLOv5 for Face Recognition

```python
[ ] import os
    import shutil

    # Create a temporary directory to hold all validation images
    temp_dir = '/content/drive/MyDrive/GP_YOLOV5Model/Val_Temp'
    os.makedirs(temp_dir, exist_ok=True)

    # List of all subdirectories containing images
    subdirs = [
        'Natasha_Romanoff (Scarlett Johansson)',
        'bruce_banner (Mark Ruffalo)',
        'clint_barton (Jeremy Renner)',
        'steve_rogers (Chris Evans)',
        'thor (Chris Hemsworth)',
        'tony_stark (Robert Downey Jr.)'
    ]

    # Base path of the validation directory
    val_base_path = '/content/drive/MyDrive/fnew/newws/val'

    # Copy all images from each subdirectory to the temporary directory


    for subdir in subdirs:
        image_dir = os.path.join(val_base_path, subdir, 'images')
        for filename in os.listdir(image_dir):
            file_path = os.path.join(image_dir, filename)
            if os.path.isfile(file_path):
                shutil.copy(file_path, temp_dir)

    # Now run the YOLO prediction command on the temporary directory
    !yolo task=detect mode=predict model='/content/drive/MyDrive/GP_YOLOV5Model/best2.pt' conf=0.25 source=/content/drive/MyDrive/GP_YOLOV5Model/Val_Temp
```

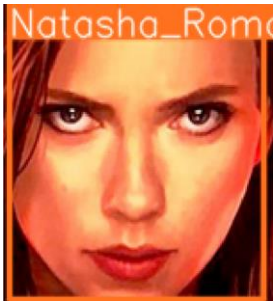
Figure 65 Organizing and Running YOLOv5 predictions on testing Images

The above program imports all the necessary modules, creates a temporary directory for the holding images under test, defines subdirectories containing images for different characters, sets the base path of the testing directory, and copies all photos from each subdirectory to the temporary directory. Finally, we run the YOLO predict command on the photos in a temporary directory using the model path, confidence threshold, and source directory. The images to be tested are consolidated in a single directory so that the YOLO model can make its predictions.

```
image 25/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner123.png: 640x640 1 bruce_banner (Mark Ruffalo), 12.4ms
image 26/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner125.png: 640x640 1 steve_rogers (Chris Evans), 1 thor (C
image 27/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner127.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.8ms
image 28/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner132.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.9ms
image 29/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner2.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.7ms
image 30/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner24.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 31/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner25.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.6ms
image 32/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner3.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 33/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner30.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 34/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner4.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 35/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner42.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 36/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner54.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 37/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner58.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 38/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner59.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 39/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner6.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 40/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner67.png: 640x640 1 bruce_banner (Mark Ruffalo), 10.5ms
image 41/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner81.png: 640x640 2 bruce_banner (Mark Ruffalo)s, 17.0ms
image 42/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/bruce_banner98.png: 640x640 1 bruce_banner (Mark Ruffalo), 15.1ms
image 43/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/chris_evans1.png: 640x640 1 steve_rogers (Chris Evans), 13.0ms
image 44/161 /content/drive/MyDrive/GP_YOLOV5Model/Val_Temp/chris_evans21.png: 640x640 1 steve_rogers (Chris Evans), 12.5ms
```

Figure 66 sample of predictions of testing images (YOLOv5)

Classes	Correct Predictions	Incorrect Predictions
Natasha_Romanoff (Scarlett Johansson)	30	0
thor (Chris Hemsworth)	31	0
bruce_banner (Mark Ruffalo)	30	2
steve_rogers (Chris Evans)	27	0
tony_stark (Robert Downey Jr.)	16	2
clint_barton (Jeremy Renner)	19	1



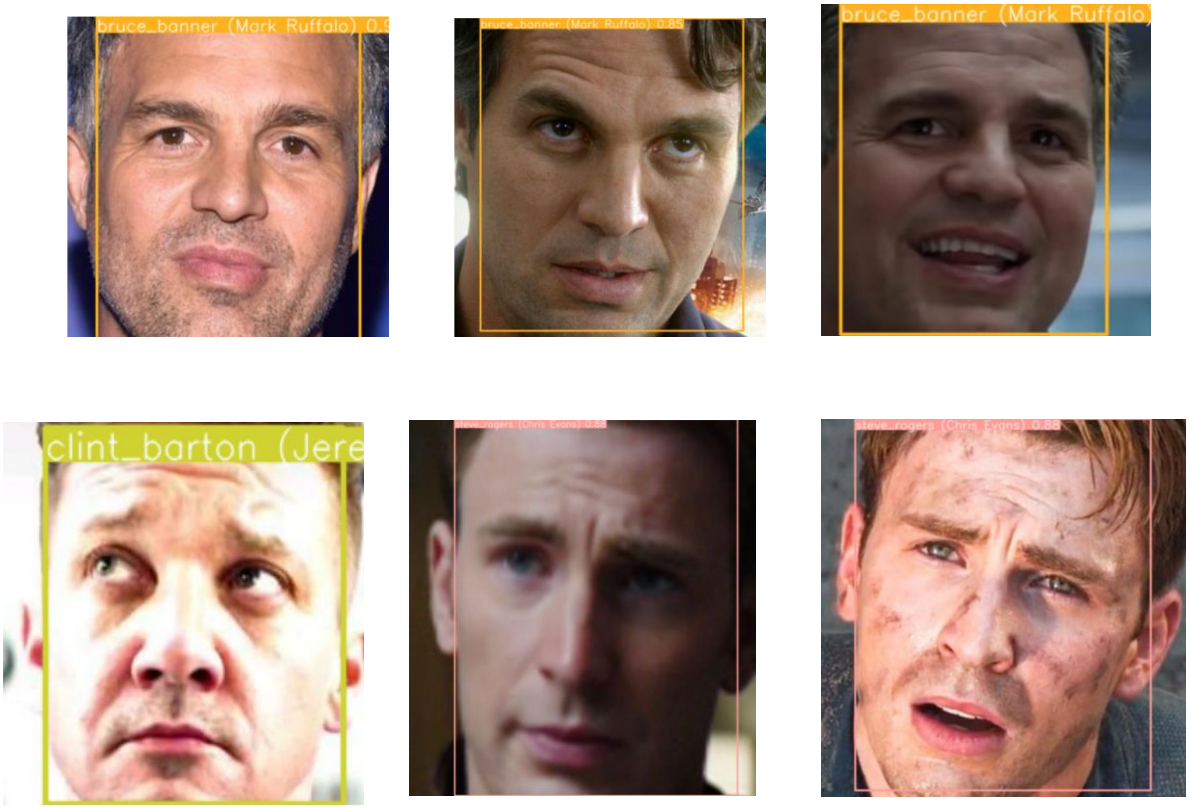


Figure 67 sample of test predictions (YOLOv5)

Figure 68 another sample of test predictions (YOLOv5)

7.8.5 YOLOV8 for Face Recognition

```python
import os
import shutil

# Created a temporary directory to hold all validation images
temp_dir = '/content/drive/MyDrive/GP_YOLOV8Model/Val_Temp'
os.makedirs(temp_dir, exist_ok=True)

# List of all subdirectories containing images
subdirs = [
    'Natasha_Romanoff (Scarlett Johansson)',
    'bruce_banner (Mark Ruffalo)',
    'clint_barton (Jeremy Renner)',
    'steve_rogers (Chris Evans)',
    'thor (Chris Hemsworth)',
    'tony_stark (Robert Downey Jr.)'
]

# Base path of the validation directory
val_base_path = '/content/drive/MyDrive/fnew/newws/val'

# Copy all images from each subdirectory to the temporary directory


for subdir in subdirs:
    image_dir = os.path.join(val_base_path, subdir, 'images')
    for filename in os.listdir(image_dir):
        file_path = os.path.join(image_dir, filename)
        if os.path.isfile(file_path):
            shutil.copy(file_path, temp_dir)

# here iam running the YOLO prediction command on the temporary directory
!yolo detect predict model='/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train2/weights/best.pt' conf=0.25 source=/content/drive/MyDrive/GP_YOLOV8Model/Val_Temp
```

Figure 69 Organizing and Running YOLOv8 predictions on testing Images

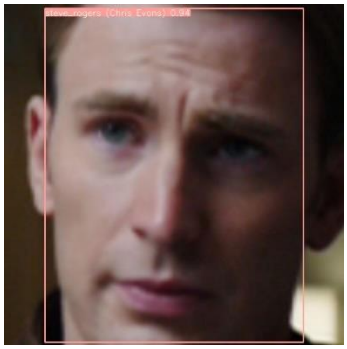
The above program imports all the necessary modules, creates a temporary directory for the holding images under test, defines subdirectories containing images for different characters, sets the base path of the testing directory, and copies all photos from each subdirectory to the temporary directory. Finally, we run the YOLO predict command on the photos in a temporary directory using the model path, confidence threshold, and source directory. The images to be tested are consolidated in a single directory so that the YOLO model can make its predictions.

```
image 79/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo15.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 80/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo16.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.9ms
image 81/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo23.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 82/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo28.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 83/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo31.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 84/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo33.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 85/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo36.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 86/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo45.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 87/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo56.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 88/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/mark_ruffalo6.png: 640x640 1 bruce_banner (Mark Ruffalo), 7.3ms
image 89/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson18.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.6ms
image 90/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson26.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.3ms
image 91/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson27.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 9.2ms
image 92/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson29.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.3ms
image 93/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson32.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.3ms
image 94/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson34.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.3ms
image 95/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson36.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.3ms
image 96/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson6.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.3ms
image 97/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson8.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.4ms
image 98/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/scarlett_johansson9.png: 640x640 1 Natasha_Romanoff (Scarlett Johansson), 7.6ms
image 99/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/steve_rogers10.png: 640x640 1 steve_rogers (Chris Evans), 7.3ms
image 100/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/steve_rogers109.png: 640x640 1 steve_rogers (Chris Evans), 7.3ms
image 101/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/steve_rogers112.png: 640x640 1 steve_rogers (Chris Evans), 7.4ms
image 102/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/steve_rogers19.png: 640x640 1 steve_rogers (Chris Evans), 7.3ms
image 103/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/steve_rogers2.png: 640x640 1 steve_rogers (Chris Evans), 7.3ms
image 104/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/steve_rogers21.png: 640x640 1 steve_rogers (Chris Evans), 7.3ms
image 105/161 /content/drive/MyDrive/GP_YOLOV8Model/Val_Temp/steve_rogers25.png: 640x640 1 steve_rogers (Chris Evans), 7.3ms
```

Figure 70 sample of predictions of testing images (YOLOv8)

Classes	Correct Predictions	Incorrect Predictions
Natasha_Romanoff (Scarlett Johansson)	30	0
thor (Chris Hemsworth)	31	0
bruce_banner (Mark Ruffalo)	29	3
steve_rogers (Chris Evans)	25	2
tony_stark (Robert Downey Jr.)	17	1
clint_barton (Jeremy Renner)	18	2

Test Samples:



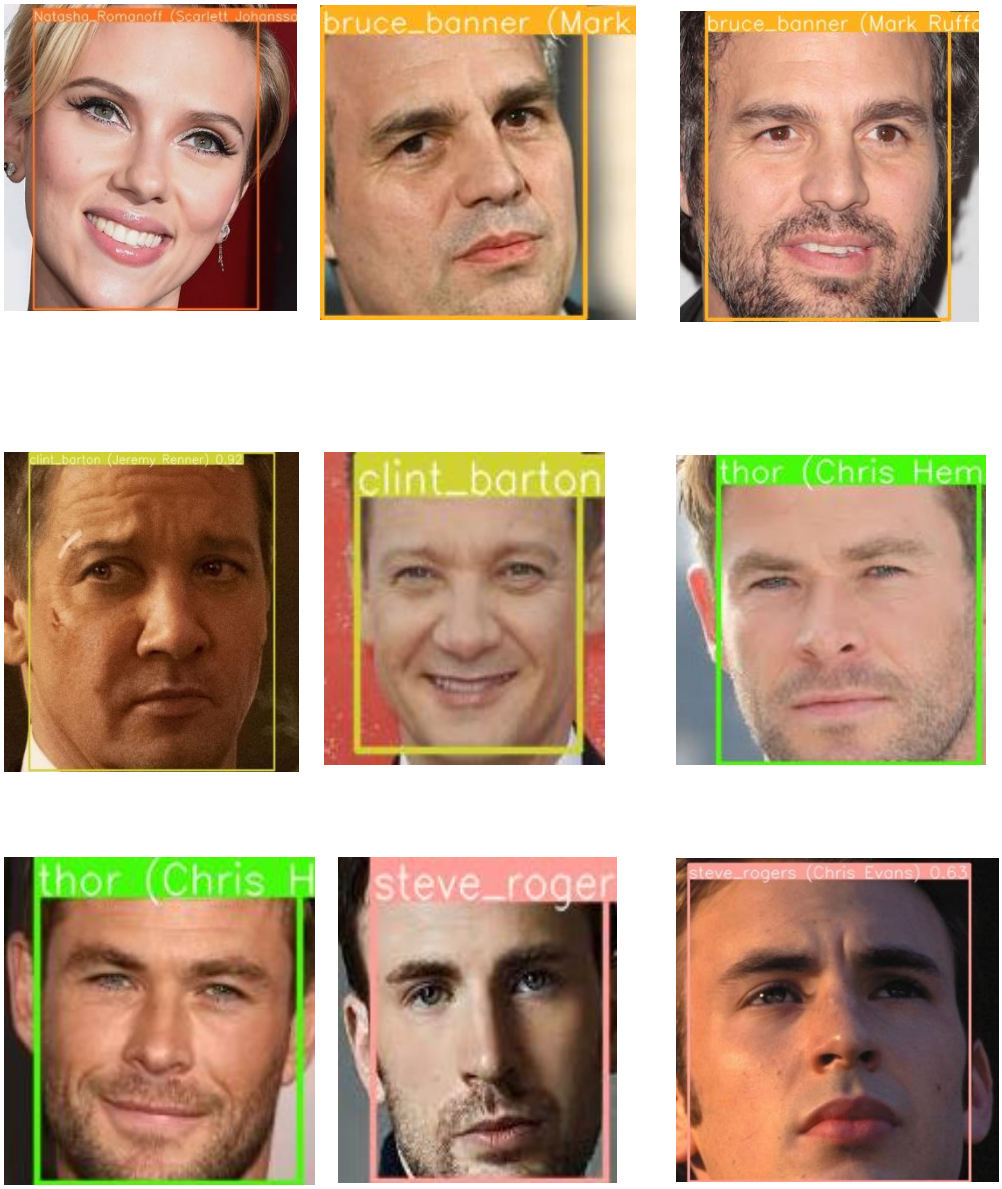
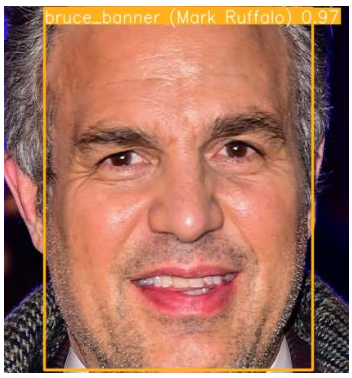


Figure 71 testing predictions samples(yolov8)



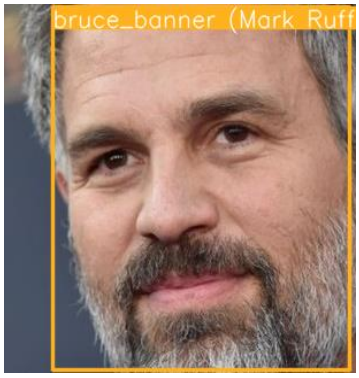


Figure 72 another testing predictions samples(yolov8)

7.8.6 Blurring Techniques

7.8.6.1 Before any Blurring Techniques:

```
Tony Stark - Correct: 289, Incorrect: 6
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 184, Incorrect: 2
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 73 Before Any Blurring Techniques: Correct and Incorrect Predictions of several classes

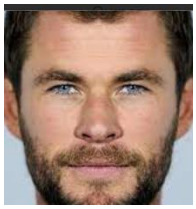


Figure 74 original image before any blurring

7.8.6.2 Gaussian Blur (Thor and Tony):

```
Tony Stark - Correct: 1, Incorrect: 294
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 7, Incorrect: 179
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 75 After Guassian Blur: Correct and Incorrect Predictions of several classes

Figure 76 image after gaussian blur

In the image above, the recognition results are shown after applying gaussian blur on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.3 Median Blur (Tony and Thor)

```
Tony Stark - Correct: 41, Incorrect: 254
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 46, Incorrect: 140
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 77 After median Blur: Correct and Incorrect Predictions of several classes

Figure 78 image after median blur

In the image above, the recognition results are shown after applying Median blur on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.4 Motion Blur

```
Tony Stark - Correct: 257, Incorrect: 38
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 169, Incorrect: 17
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 79 After motion Blur: Correct and Incorrect Predictions of several classes

Figure 80 image after motion blur

In the image above, the recognition results are shown after applying Motion blur on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.5 GrabCut & Gaussian Blur

```
Tony Stark - Correct: 100, Incorrect: 195
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 51, Incorrect: 135
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 81 After Grabcut(gaussian) Blur: Correct and Incorrect Predictions of several classes

Figure 82 image after grabcut(gaussian) blur

In the image above, the recognition results are shown after applying GrabCut and Gaussian blur on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.6 Mean Shift Filtering

Figure 83 image after meanshift filtering

```
Tony Stark - Correct: 209, Incorrect: 86
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 91, Incorrect: 95
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 84 After MeanShift Filtering Blur: Correct and Incorrect Predictions of several classes

In the image above, the recognition results are shown after applying Mean Shift Filtering blur on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.7 Box Filter

```
Tony Stark - Correct: 48, Incorrect: 247
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 80, Incorrect: 106
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 85 After Box filter Blur: Correct and Incorrect Predictions of several classes

Figure 86 image after box filter

In the image above, the recognition results are shown after applying Box Filter blur on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.8 Radial Blur

```
Tony Stark - Correct: 10, Incorrect: 285
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 29, Incorrect: 157
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 87 After Radial Blur: Correct and Incorrect Predictions of several classes

Figure 88 image after radial blur

In the image above, the recognition results are shown after applying Radial blur on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.9 Motion and Mean Shift Filtering

```
Tony Stark - Correct: 159, Incorrect: 136
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 66, Incorrect: 120
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 89 After Motion and meanshift filtering Blur: Correct and Incorrect Predictions of several classes

In the image above, the recognition results are shown after applying sequentially the Motion and Mean shift Filtering on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.10 Radial and Gaussian Blur

```
Tony Stark - Correct: 0, Incorrect: 295
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 4, Incorrect: 182
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 90 After radial and gaussian Blur: Correct and Incorrect Predictions of several classes

In the image above, the recognition results are shown after applying sequentially the Radial and Gaussian Blur on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.11 Motion, Mean shift Filtering, Radial and Gaussian Blur

```
Tony Stark - Correct: 0, Incorrect: 295
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 4, Incorrect: 182
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 91 After Motion, radial, gaussian and meanshift filtering Blur: Correct and Incorrect Predictions of several classes

In the image above, the recognition results are shown after applying sequentially the Motion, Radial, Gaussian and Mean shift Filtering on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.8.6.12 Motion, Mean shift Filtering, Radial and Gaussian Blur (5 iterations each)

```
Tony Stark - Correct: 0, Incorrect: 295
Steve Rogers - Correct: 159, Incorrect: 11
Thor - Correct: 0, Incorrect: 186
Clint Barton - Correct: 112, Incorrect: 9
Natasha Romanoff - Correct: 182, Incorrect: 14
Bruce Banner - Correct: 185, Incorrect: 11
```

Figure 92 After Motion, radial, gaussian and meanshift filtering Blur: Correct and Incorrect Predictions of several classes (5 iterations each)

In the image above, the recognition results are shown after applying sequentially the Motion, Radial, Gaussian and Mean shift Filtering where each blurring technique is applied (iterated) 5 times on the classes (Tony and Thor) images which did not give their consent to take their pictures.

7.9 Evaluation

7.9.1 YOLOV8 as a Face Detection Model:

```
!yolo detect train data='/content/lfw_face-1/data.yaml' model=yolov8n.pt epochs=15 imgsz=640 device=0
```

Figure 93 yolov8 training code

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/15      2.59G     0.9847      0.945       1.2         25        640: 100% 662/662 [03:44<00:00,  2.95it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:26<00:00,  3.11it/s]
                   all       2646       3169      0.965      0.918      0.956      0.738

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/15      2.26G     0.9088     0.5739      1.127         20        640: 100% 662/662 [03:37<00:00,  3.04it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:24<00:00,  3.41it/s]
                   all       2646       3169      0.983      0.928      0.973      0.778

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/15      2.25G     0.8809     0.5284      1.111         30        640: 100% 662/662 [03:32<00:00,  3.12it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:22<00:00,  3.63it/s]
                   all       2646       3169      0.975      0.938      0.972       0.77

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       4/15      2.27G     0.8615     0.5026      1.099         15        640: 100% 662/662 [03:29<00:00,  3.16it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:23<00:00,  3.58it/s]
                   all       2646       3169      0.983      0.939      0.981      0.798

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       5/15      2.25G     0.8309     0.4683      1.081         26        640: 100% 662/662 [03:30<00:00,  3.15it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:25<00:00,  3.29it/s]
                   all       2646       3169      0.982      0.943      0.981      0.792
sing dataloader mosaic
umentations: 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_limit=
r/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multi
elf.pid = os.fork()

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       6/15      2.39G      0.761     0.4037      1.084          9        640: 100% 662/662 [03:18<00:00,  3.34it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:25<00:00,  3.26it/s]
                   all       2646       3169      0.974      0.951      0.982      0.805

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       7/15      2.26G     0.7391     0.3804      1.073         11        640: 100% 662/662 [03:11<00:00,  3.46it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:24<00:00,  3.41it/s]
                   all       2646       3169       0.98      0.953      0.985      0.803

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       8/15      2.26G     0.7151     0.3616      1.049         11        640: 100% 662/662 [03:12<00:00,  3.44it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:23<00:00,  3.57it/s]
                   all       2646       3169      0.981      0.954      0.987      0.831

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       9/15      2.26G     0.7031      0.344      1.042         11        640: 100% 662/662 [03:16<00:00,  3.36it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:22<00:00,  3.63it/s]
                   all       2646       3169      0.984      0.952      0.987      0.834

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      10/15      2.26G     0.6952     0.3365      1.037         11        640: 100% 662/662 [03:17<00:00,  3.35it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:22<00:00,  3.72it/s]
                   all       2646       3169      0.983      0.956      0.987      0.825
```

```
  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  11/15      2.26G     0.6786     0.3242      1.022         10        640: 100% 662/662 [03:19<00:00,  3.31it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:22<00:00,  3.68it/s]
               all       2646       3169      0.984      0.957      0.988      0.832

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  12/15      2.26G     0.6659     0.3159      1.017          9        640: 100% 662/662 [03:18<00:00,  3.33it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:24<00:00,  3.36it/s]
               all       2646       3169      0.981      0.962      0.989      0.832

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  13/15      2.26G      0.654     0.3053       1.01         11        640: 100% 662/662 [03:15<00:00,  3.39it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:24<00:00,  3.38it/s]
               all       2646       3169      0.977      0.963      0.989      0.837

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  14/15      2.26G     0.6425     0.2907      0.999         13        640: 100% 662/662 [03:22<00:00,  3.27it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:23<00:00,  3.57it/s]
               all       2646       3169      0.976      0.964       0.99      0.832

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  15/15      2.26G     0.6288     0.2823     0.9924         11        640: 100% 662/662 [03:22<00:00,  3.27it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:23<00:00,  3.56it/s]
               all       2646       3169      0.983      0.961       0.99      0.839

pochs completed in 0.952 hours.
mizer stripped from runs/detect/train/weights/last.pt, 6.2MB
mizer stripped from runs/detect/train/weights/best.pt, 6.2MB

dating runs/detect/train/weights/best.pt...
alytics YOLOv8.2.27 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
l summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 83/83 [00:30<00:00,  2.76it/s]
               all       2646       3169      0.983      0.961       0.99      0.839
d: 0.2ms preprocess, 2.2ms inference, 0.0ms loss, 1.9ms postprocess per image
lts saved to runs/detect/train
earn more at https://docs.ultralytics.com/modes/train
```

Figure 94 yolov8 epochs results

The training for the YOLOv8 model was conducted for 15 epochs, taking a total of 9.952 hours with consistent GPU memory usage of around 2.26 GB. Each epoch fully utilized the batch size of 662/662. The box loss started at 0.9874 and decreased to 0.6226, showing steady improvement. The classification loss began high at 3.167 and dropped significantly to 0.2823, while the Dfl loss started at 1.2 and ended at 1.022, indicating better bounding box localization.

Performance metrics showed that mAP50 improved from 0.918 to 0.984, reflecting better precision, and mAP50-95 increased from 0.738 to 0.839, indicating improved performance across various IoU thresholds. Precision consistently improved from 0.956 to 0.983, and recall showed an increase from 0.738 to 0.839, indicating fewer false negatives.

Epoch-wise insights reveal rapid reduction in losses during the initial epochs, with box_loss dropping to 0.8615 by epoch 4, and precision and recall improving to 0.983 and 0.798, respectively. By epoch 10, mAP50-95 had improved to 0.832. The final epochs showed further reduction in losses with box_loss at 0.6226 and cls_loss at 0.2823, with precision and recall peaking at 0.983 and 0.839, respectively. Final validation showed an mAP50 of 0.983 and mAP50-95 of 0.839, with high precision and recall across classes like Tony Stark, Steve Rogers, Natasha Romanoff, Bruce Banner, Clint Barton, and Thor. The model demonstrated high accuracy and robustness in recognizing faces from the Avengers dataset, showcasing its effectiveness and efficiency.

7.9.1.1 Confusion Matrix (YOLOV8 for Face Detection)

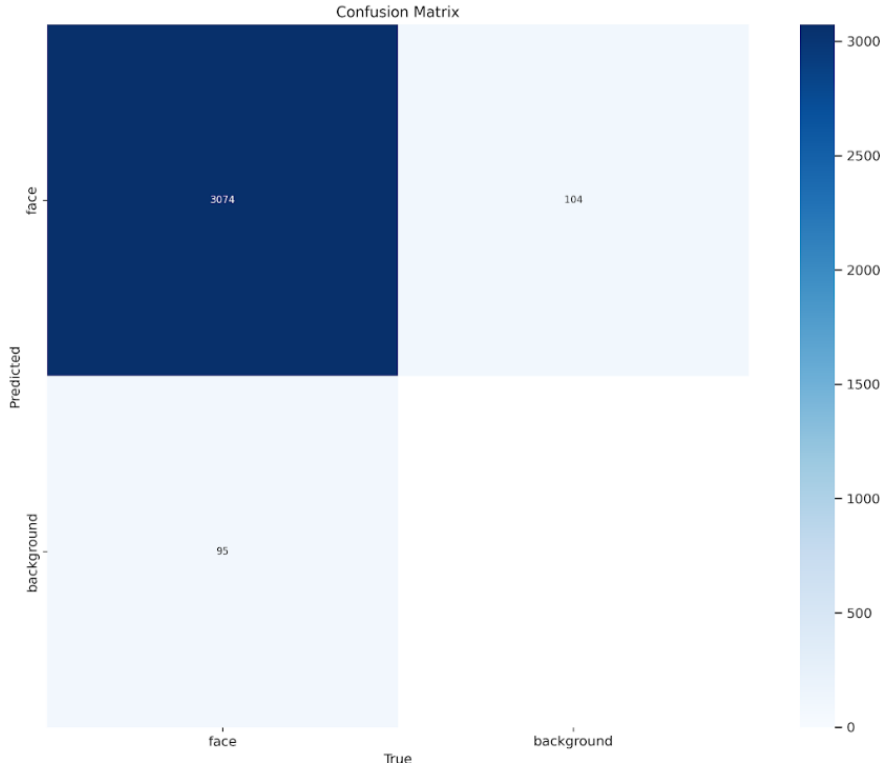


Figure 95 confusion matrix of yolov8 for face detection

- **True Positives (3074):** Correctly identified faces.
- **False Positives (104):** Incorrectly identified background as faces.
- **False Negatives (95):** Missed faces, labeled as background.
- **True Negatives (0):** No correct background predictions due to no background class in the dataset.

Insight:

The model accurately detects faces most of the time where only moderate number of backgrounds are mistakenly identified as faces (false positives) and few faces are missed (false negatives).

7.9.1.2 F1-Confidence Curve (YOLOV8 for Face Detection)

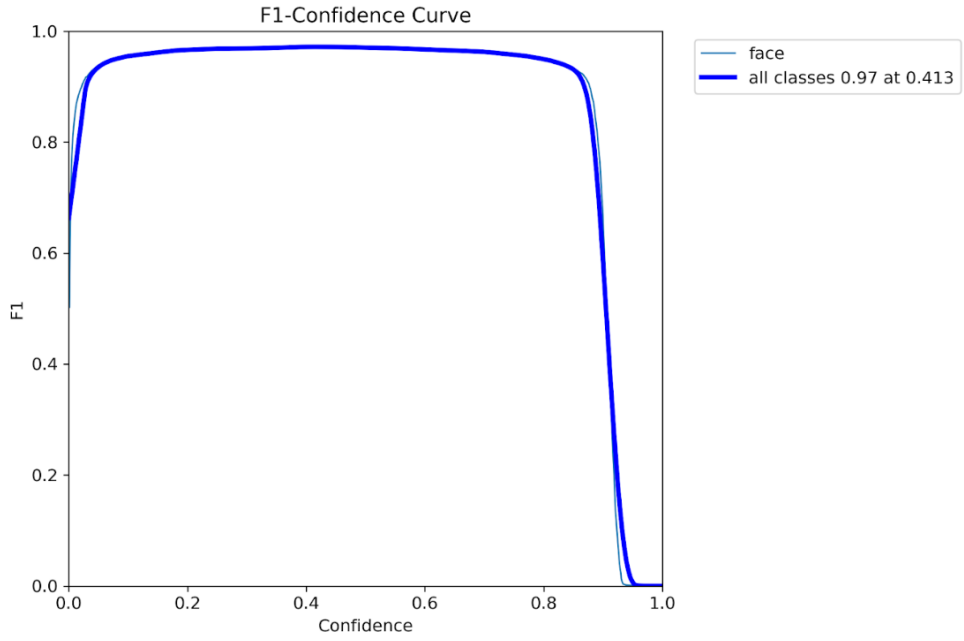


Figure 96 F1-confidence curve of yolov8 for face detection

The F1-Confidence Curve shows how the F1 score varies with confidence levels for face detection:

- High F1 Score (0.97) at 0.413 Confidence: This is where the model performs its best.
- Curve Shape: The F1 score stays high and stable over a broad range of confidence levels but drops sharply after 0.9 confidence.

Key Insights:

• Optimal Performance: The model works best at a confidence level around 0.4.

• Robustness: The model keeps high F1 scores across many confidence levels, showing reliable performance.

7.9.1.3 Precision - Confidence Curve (YOLOV8 for Face Detection)

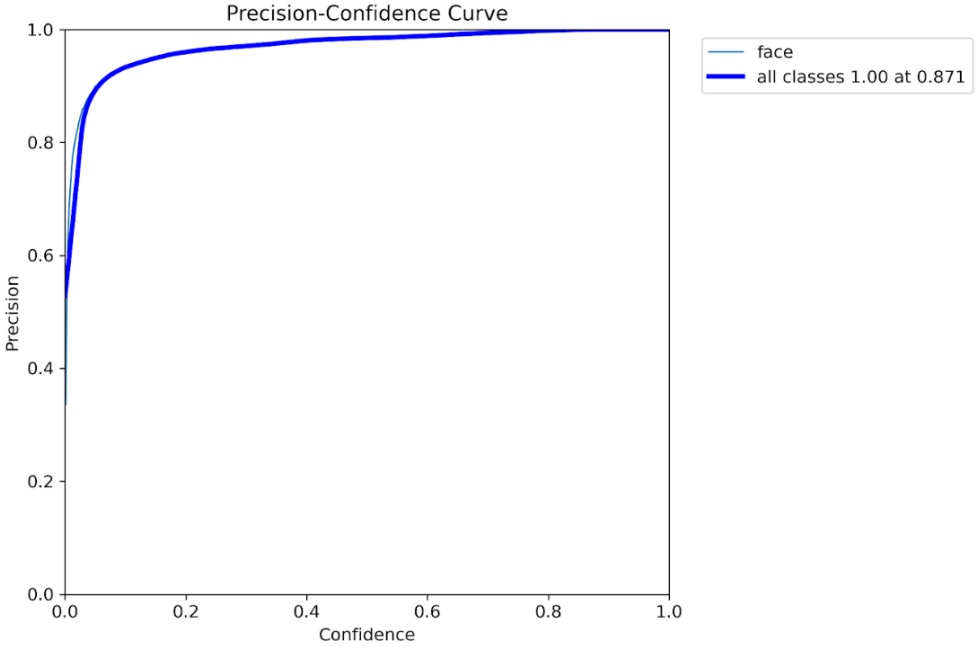


Figure 97 precision-confidence curve of yolov8 for face detection

The Precision-Confidence Curve shows how the face detection model's precision changes with different confidence levels. Peak precision of 1.00 is achieved at 0.871 confidence, meaning the model has perfect precision and no false positives at this point. The curve indicates that precision quickly improves with increasing confidence, stabilizing near perfect at higher confidence levels.

Key Insights: The model is very precise at confidence levels above 0.87, minimizing false positives. The curve shows that the model consistently maintains high precision as confidence increases, indicating reliable detection accuracy.

7.9.1.4 Recall - Confidence Curve (YOLOV8 for Face Detection)

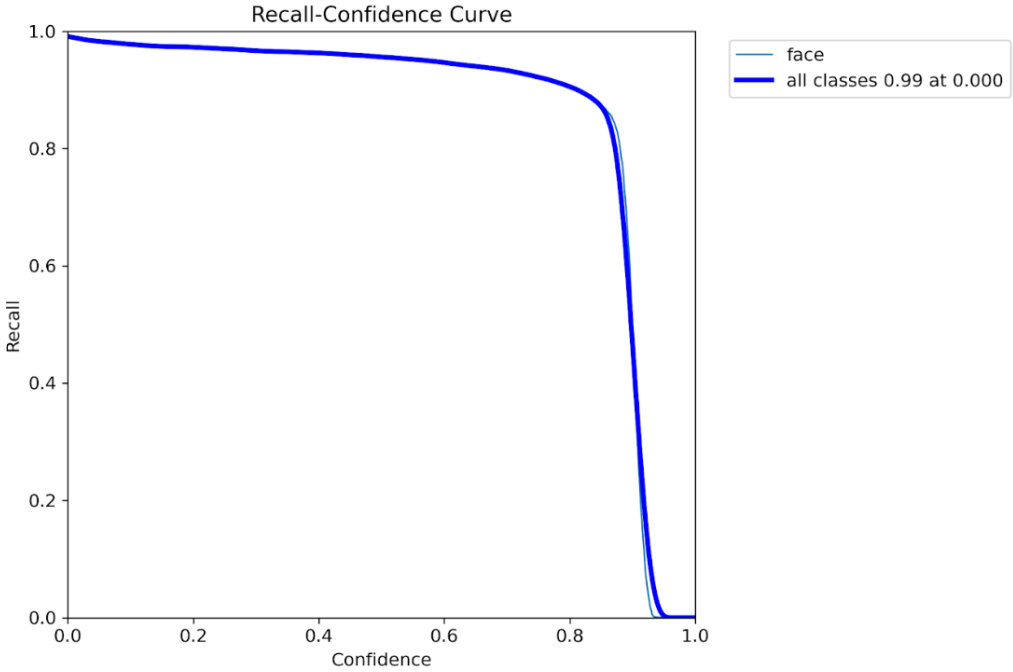


Figure 98 recall - confidence curve of yolov8 for face detection

The Recall-Confidence Curve shows how the model's recall varies with different confidence thresholds. The model achieves a high recall of 0.99 at 0.000 confidence, detecting almost all true positives at the lowest confidence level. The curve indicates that recall remains high at lower to mid confidence levels but drops sharply at higher thresholds.

Key Insights: The model maintains high recall, close to 1.0, at lower confidence levels, effectively capturing most faces. However, as confidence increases, recall drops significantly, indicating a trade-off between confidence threshold and recall. At higher confidence levels, the model becomes more conservative, missing some true positives.

7.9.1.5 Result (YOLOV8 for Face Detection)

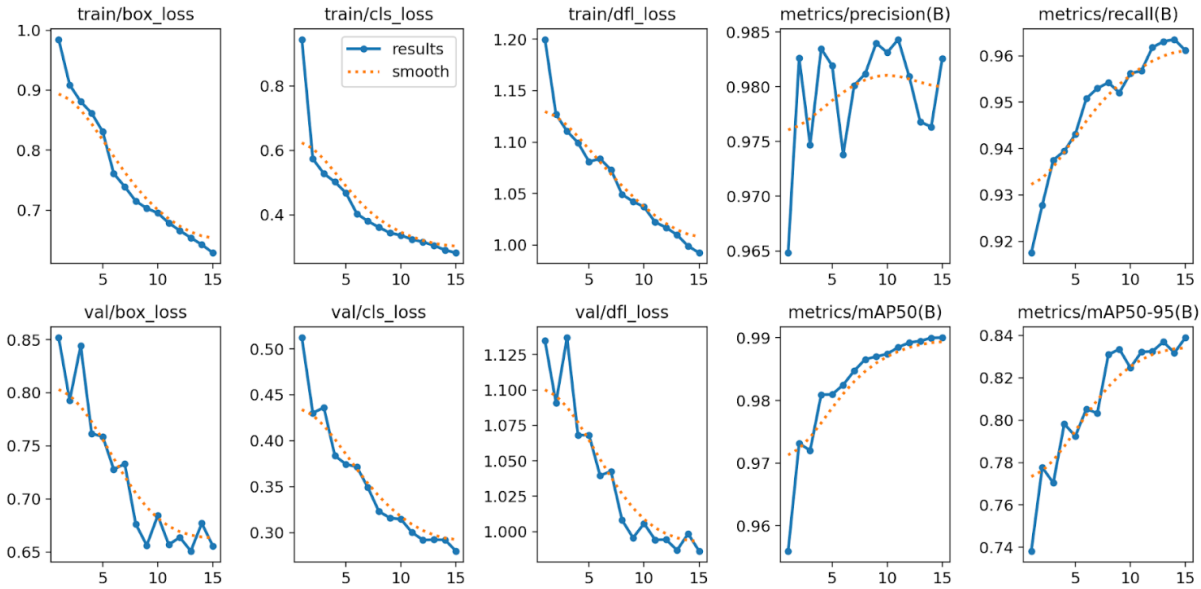


Figure 99 results curves of yolov8 for face detection

The training and validation metrics, visualized above, over 15 epochs for one of the face recognition models using YOLOv8. The training losses—box_loss, cls_loss, dfl_loss—decrease smoothly and show good improvement in bounding box predictions, better classification accuracy, and more accurate learning of distance fields. Validation losses—val/box_loss, val/cls_loss, val/dfl_loss—also show a decrease, which means good generalization on unseen data. The precision of metrics/precision(b) oscillates but has a generally increasing trend, indicating increased precision along epochs. Recall metrics/recall(B) are constantly increasing, as over time, the model can capture more true positives. The mAP metrics, which are mAP50(B) and mAP50-95(B), reach near 1.0; it increases, showing high precision and recall at different IoU thresholds. Overall, the model has improved in training and validation losses and detection accuracy; it has a decent performance across various IoU thresholds. Thus, it generalizes well on unseen data.

7.9.1.6 Validation Sample Batch 1 Labels

Figure 100 yolov8 validation sample batch 1 labels

7.9.1.7 Validation Sample Batch 1 Predictions

Figure 101 yolov8 validation sample batch 1 predictions

7.9.1.8 Validation Sample Batch 2 Labels

Figure 102 yolov8 validation sample batch 2 labels

7.9.1.9 Validation Sample Batch 2 Predictions

Figure 103 yolov8 validation sample batch 1 predictions

7.9.2 YOLOV5 as a Face Detection Model:

```
!yolo mode=check task=detect mode=train model=yolov5s.pt data='/content/lfw_face-1/data.yaml' epochs=30 imgsz=640 batch=64 device=0
```

Figure 104 training of yolov5 for face detection

```
     1/30        15G      1.019     0.9033      1.264         61        640: 100% 166/166 [03:35<00:00,  1.30s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   0% 0/21 [00:00<?, ?it/s]/usr/local/l
return F.conv2d(input, weight, bias, self.stride,
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:33<00:00,  1.59s/it]
                 all       2646       3169      0.843      0.839      0.905      0.679

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     2/30        15G     0.9311     0.5879      1.145         47        640: 100% 166/166 [03:35<00:00,  1.30s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.51s/it]
                 all       2646       3169      0.887      0.844      0.895      0.667

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     3/30        15G      0.924     0.5686      1.133         57        640: 100% 166/166 [03:29<00:00,  1.26s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:33<00:00,  1.58s/it]
                 all       2646       3169      0.922      0.814      0.891      0.622

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     4/30      15.1G     0.9036      0.543      1.119         57        640: 100% 166/166 [03:33<00:00,  1.29s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.49s/it]
                 all       2646       3169      0.976      0.923      0.965      0.765

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     5/30        15G     0.8705     0.5128      1.099         54        640: 100% 166/166 [03:33<00:00,  1.29s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:32<00:00,  1.53s/it]
                 all       2646       3169      0.977      0.924      0.975      0.737

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     6/30        15G     0.8373     0.4825      1.084         55        640: 100% 166/166 [03:28<00:00,  1.25s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:32<00:00,  1.53s/it]
                 all       2646       3169      0.977      0.936      0.977      0.786

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     7/30      15.1G     0.8293     0.4752      1.077         65        640: 100% 166/166 [03:36<00:00,  1.31s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:34<00:00,  1.64s/it]
                 all       2646       3169      0.981      0.941      0.979      0.803

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     8/30      15.1G     0.8115     0.4588      1.069         44        640: 100% 166/166 [03:34<00:00,  1.29s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.48s/it]
                 all       2646       3169      0.984      0.945      0.982      0.799

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     9/30      15.1G     0.7991     0.4482      1.062         63        640: 100% 166/166 [03:35<00:00,  1.30s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.51s/it]
                 all       2646       3169      0.978       0.95      0.982      0.801

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    10/30      15.1G     0.7899     0.4387      1.056         60        640: 100% 166/166 [03:30<00:00,  1.27s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:32<00:00,  1.53s/it]
                 all       2646       3169      0.982      0.944      0.983       0.81
```

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
11/30        15G     0.7722     0.4266      1.047         57        640: 100% 166/166 [03:35<00:00,  1.30s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:32<00:00,  1.53s/it]
             all       2646       3169      0.975      0.933      0.975      0.775

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
12/30        15G     0.7704     0.4229      1.046         62        640: 100% 166/166 [03:38<00:00,  1.31s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.51s/it]
             all       2646       3169      0.983       0.95      0.985      0.822

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
13/30        15G     0.7702      0.416      1.046         54        640: 100% 166/166 [03:40<00:00,  1.33s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.50s/it]
             all       2646       3169      0.984      0.946      0.983      0.802

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
14/30        15G     0.7592     0.4128      1.042         51        640: 100% 166/166 [03:34<00:00,  1.29s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.50s/it]
             all       2646       3169      0.982      0.958      0.988      0.821

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
15/30      15.1G     0.7457     0.4002      1.034         61        640: 100% 166/166 [03:30<00:00,  1.27s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.50s/it]
             all       2646       3169      0.985      0.951      0.985      0.804

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
16/30        15G     0.7392     0.3953      1.035         66        640: 100% 166/166 [03:34<00:00,  1.29s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.48s/it]
             all       2646       3169      0.983      0.958      0.988      0.824

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
17/30        15G     0.7399     0.3928      1.035         54        640: 100% 166/166 [03:30<00:00,  1.27s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:30<00:00,  1.47s/it]
             all       2646       3169      0.982      0.955      0.985      0.819

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
18/30        15G      0.734     0.3883      1.032         56        640: 100% 166/166 [03:35<00:00,  1.30s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.52s/it]
             all       2646       3169      0.981       0.96       0.99      0.834

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
19/30      15.1G     0.7331     0.3877      1.029         68        640: 100% 166/166 [03:28<00:00,  1.26s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.51s/it]
             all       2646       3169       0.98      0.957      0.989      0.823

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
20/30        15G      0.723     0.3809      1.026         47        640: 100% 166/166 [03:33<00:00,  1.28s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.48s/it]
             all       2646       3169      0.982      0.957      0.988      0.831
```

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
21/30       15G     0.6784     0.3297      1.025         31        640: 100% 166/166 [03:25<00:00,  1.24s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.51s/it]
             all       2646       3169       0.98      0.961      0.986      0.828

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
22/30       15G     0.6714     0.3226      1.022         31        640: 100% 166/166 [03:23<00:00,  1.22s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.52s/it]
             all       2646       3169      0.974      0.965      0.989      0.827

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
23/30       15G      0.661     0.3114      1.011         29        640: 100% 166/166 [03:22<00:00,  1.22s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:30<00:00,  1.46s/it]
             all       2646       3169      0.985      0.961      0.991      0.844

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
24/30       15G     0.6518     0.3049      1.005         25        640: 100% 166/166 [03:27<00:00,  1.25s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.49s/it]
             all       2646       3169      0.977      0.965      0.991      0.839

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
25/30       15G     0.6426     0.2972      1.002         27        640: 100% 166/166 [03:21<00:00,  1.21s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.52s/it]
             all       2646       3169       0.98      0.961       0.99      0.849

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
26/30       15G     0.6337     0.2868     0.9971         28        640: 100% 166/166 [03:25<00:00,  1.24s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:30<00:00,  1.44s/it]
             all       2646       3169      0.985      0.964      0.988      0.836

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
27/30       15G      0.627     0.2844     0.9938         28        640: 100% 166/166 [03:19<00:00,  1.20s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.52s/it]
             all       2646       3169      0.975      0.968      0.991      0.841

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
28/30       15G     0.6192     0.2794     0.9873         29        640: 100% 166/166 [03:24<00:00,  1.23s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:33<00:00,  1.60s/it]
             all       2646       3169       0.98      0.963      0.991      0.847

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
29/30       15G     0.6109     0.2716     0.9823         30        640: 100% 166/166 [03:22<00:00,  1.22s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:30<00:00,  1.44s/it]
             all       2646       3169       0.98      0.966      0.991      0.848

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
30/30       15G     0.6023     0.2675     0.9777         33        640: 100% 166/166 [03:26<00:00,  1.24s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:31<00:00,  1.50s/it]
             all       2646       3169      0.981      0.965      0.991      0.849


30 epochs completed in 2.028 hours.
Optimizer stripped from runs/detect/train3/weights/last.pt, 18.5MB
Optimizer stripped from runs/detect/train3/weights/best.pt, 18.5MB

Validating runs/detect/train3/weights/best.pt...
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/conv.py:456: UserWarning: Plan failed with a cudnnException: CUDNN_BAC
  return F.conv2d(input, weight, bias, self.stride,
YOLOv5s summary (fused): 193 layers, 9111923 parameters, 0 gradients, 23.8 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 21/21 [00:40<00:00,  1.92s/it]
                   all       2646       3169      0.982      0.965      0.991      0.849
Speed: 0.2ms preprocess, 5.4ms inference, 0.0ms loss, 1.9ms postprocess per image
Results saved to runs/detect/train3
💡 Learn more at https://docs.ultralytics.com/modes/train
```

Figure 105 yolov5 epochs results

This training will span over 30 epochs and monitor metrics, such as box loss, classification loss, and DFL loss, which reduce gradually, proving that the model learns. The GPU memory usage is more or less stable at 15.1 GB, showing that it accepts the computational load without crashes. Box Loss decreases from 0.9033 to 0.6023, improving bounding box prediction. Classification Loss drops from 1.264 to 0.2675, implying better classification accuracy for human faces. DFL Loss reduces from 1.2

to 0.9777, which indicates that prediction locations are more accurate. The number of instances processed per batch is not constant; this reflects dynamic detections per image.

The metrics of the Mean Average Precision (mAP) indicate the model performance in detection. Hence, mAP@0.5 increases from 0.843 to 0.982, while mAP@0.5:0.95 slightly improves from 0.839 to 0.849. An increase is registered for the precision metric from 0.903 to 0.991, pointing to fewer false positives. Recall is.

The model summary for YOLOv5s shows that it is a model with 193 layers and about 9.1 million parameters, making it relatively complex. The high mAP@0.5 and mAP@0.5:0.95 in final validation results are 0.991 and 0.849, respectively, confirming the suitable generalization of the model for unseen data. In this regard, the YOLOv5 model for face detection enormously enhances the accuracy of detection and computational efficiency because consistent loss reduction and improved mAP metrics both guarantee its strength in performing the task of face detection.

7.9.2.1 Confusion Matrix (YOLOV5 for Face Detection)

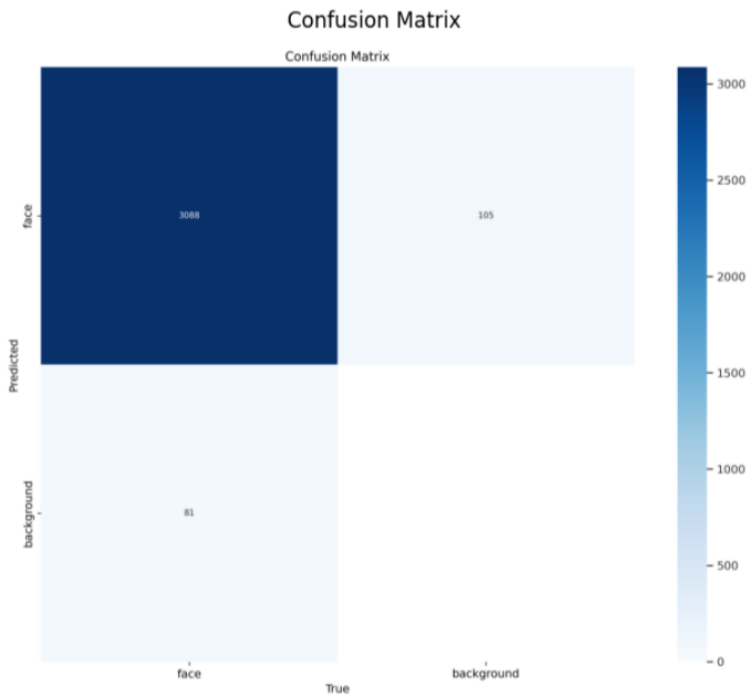


Figure 106 yolov5 confusion matrix for face detection

- **True Positives (TP):** 3088 - The model correctly predicted 3088 instances as "face".
- **True Negatives (TN):** Not applicable due to no background class in the dataset
- **False Positives (FP):** 105 - The model incorrectly predicted 105 instances as "face" when they were actually "background".

- **False Negatives (FN):** 81 - The model incorrectly predicted 81 instances as "background" when they were actually "face".

The model has a high number of true positive predictions for faces, with relatively fewer false positives and false negatives, indicating good performance in distinguishing faces from the background.

7.9.2.2 F1 - Confidence Curve (YOLOV5 for Face Detection)

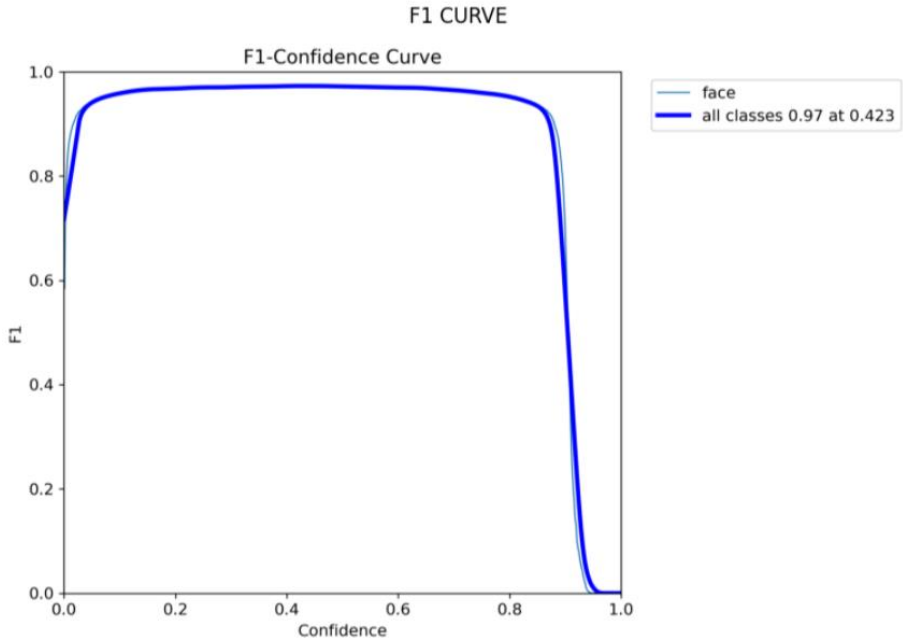


Figure 107 yolov5 F1-confidence curve for face detection

The X-Axis Title: confidence threshold in making predictions, minimizing to 0, maximizing to 1. Y-Axis Title: F1 score ranging from 0 to 1. Blue Line: F1 score for the class face, maximized to be up to 0.97, peaking at 0.423 confidence. The curve indicates that at a confidence threshold of 0.423, the F1 score is high, at 0.97, such that high precision in identification of faces and high capture of actual faces are portrayed. A high F1 score means there is good balance between precision and recall, presenting an excellent model performance in terms of face detection tasks. Precision - Confidence Curve (YOLOV5 for Face Detection)

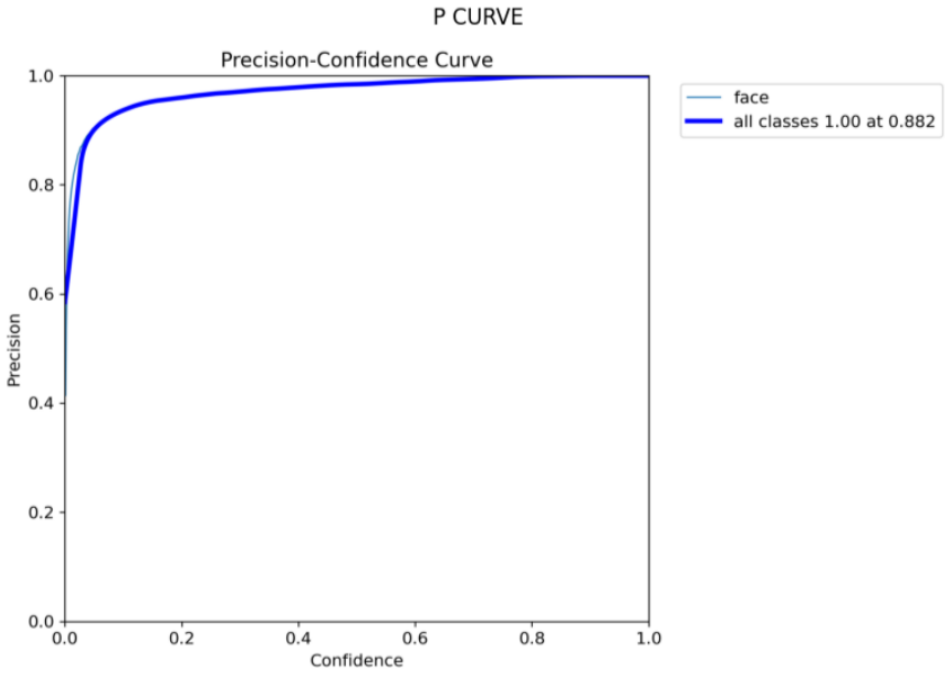


Figure 108 yolov5 precision-confidence curve for face detection

The X-axis represents a range for the confidence threshold in making predictions, ranging from 0 to 1. At the same time, the Y-axis shows the score of precision, which is accurate optimistic predictions divided by the total number of optimistic predictions (both actual and false positives), again ranging from 0 to 1. Looking carefully—at the blue line—this is the precision for the class face, and the precision score is very high, equal to 1.00, obtained at the confidence level of 0.882. This curve shows that perfect precision is recorded at this threshold by the model; all predictions for the face class are accurate, with no false positives. Moreover, this high score in precision indicates pretty good work by the model in distinguishing faces without making any wrong predictionsRecall - Confidence Curve (YOLOV5 for Face Detection)

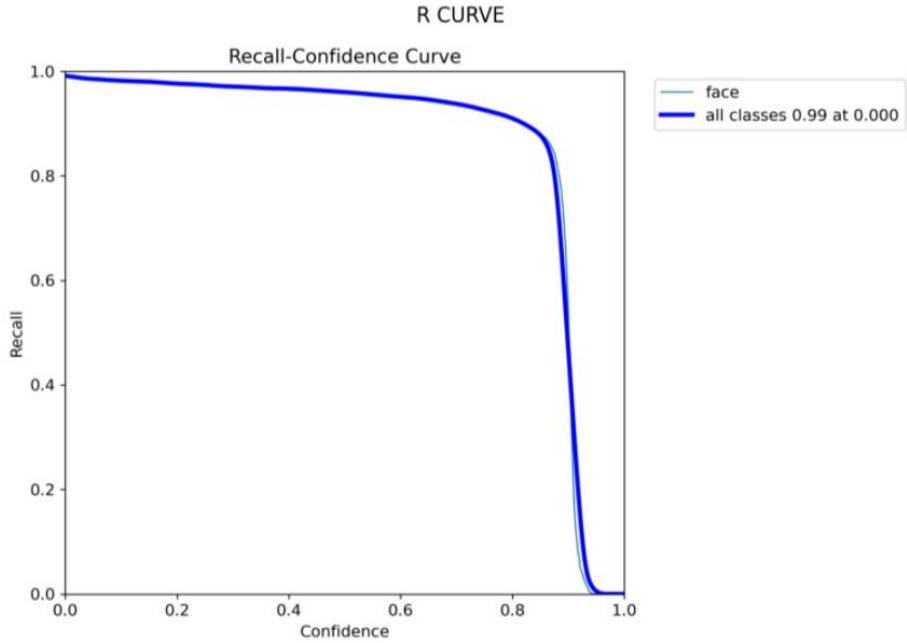


Figure 109 yolov5 recall-confidence curve for face detection

It represents the confidence threshold for making predictions over the range from 0 to 1. On the Y-Axis is the recall score, or the ratio of true positive predictions to the total number of actual positive instances. That also ranges from 0 to 1. The blue line indicates the recall specifically for the face class, showing a high recall score of 0.99 at a confidence level of 0.000. This curve shows the model plotting a very high recall score at a shallow confidence threshold, so it would work in picking almost all the actual faces with a very minute chance of missing any. As the confidence threshold increases, the recall begins to drop slightly, meaning higher confidence thresholds might find some faces that it will not. The high recall value confirms that the model effectively detects most faces.

7.9.2.3 Results (YOLOV5 for Face Detection)

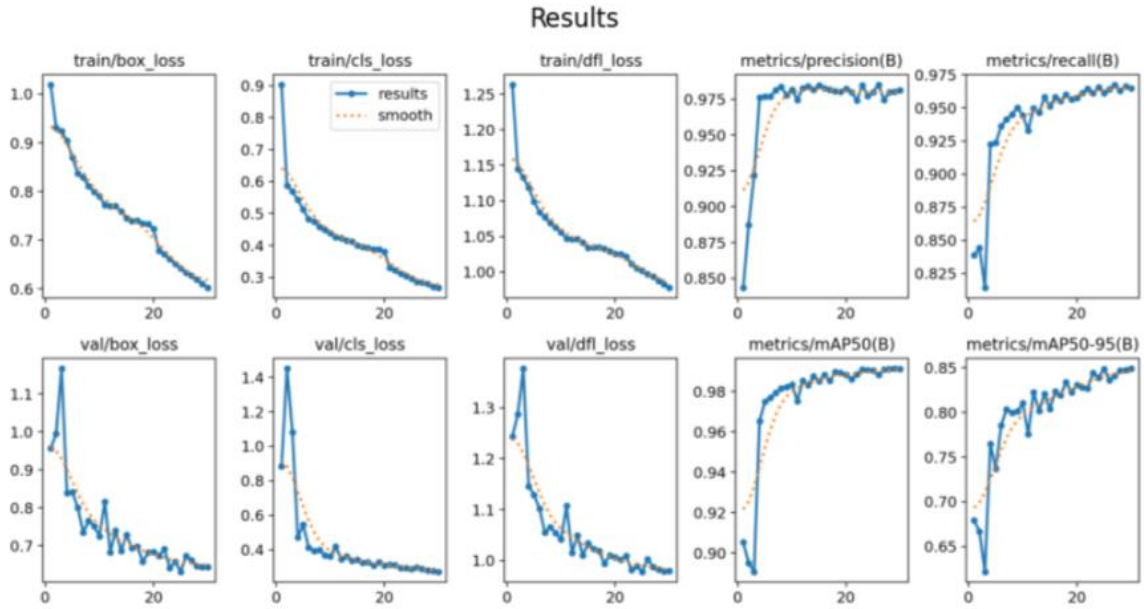


Figure 110 yolov5 Results curves for face detection

The training losses, Box Loss (train/box_loss), Classification Loss (train/cls_loss), and Distribution Focal Loss (train/dfl_loss) all show a uniformly decreasing trend over epochs, supporting evidence that the model is learning features to count bounding boxes and objects accurately. Also, the validation losses for the model in terms of Box Loss (val/box_loss), Classification Loss (val/cls_loss), and Distribution Focal Loss (val/dfl_loss) go down, hence another proof of generalization capability. Performance metrics such as Precision(metrics/precision(B)) and Recall(metrics/recall(B)) improve and stabilize, which shows the detection of fewer false positives and the successful detection of most of the true positives. mAP50/mean Average Precision at IoU=0.50(metrics/mAP50(B)) to all levels of IoU, from 0.50 to 0.95, also increases, showing good performance and robustness in object detection. These plots demonstrate that the model is learning from the training data and generalizing well: the loss curves are decreasing, and the performance metrics are increasing in a way that confirms the stability and reliability of the training process.

7.9.3 MobileNet as a Face Recognition Model:

- Model Architecture Adjustments and Compilation:
  - Loading the Pretrained MobileNet Model

First, the code loads the MobileNet model pretrained on ImageNet, excluding the top layer to use it as a feature extractor. This helps leverage the powerful feature extraction capabilities of MobileNet while allowing customization for the specific task.

```
[ ] from keras.applications import MobileNet
    from keras.layers import GlobalAveragePooling2D, Dense, Dropout, BatchNormalization
    from keras.models import Model
    from keras.callbacks import EarlyStopping, ReduceLROnPlateau

    # Load the MobileNet model, pretrained on ImageNet
    mobilenet_mdl = MobileNet(
        include_top=False,
        input_shape=(224, 224, 3),
        weights='imagenet'
    )

    # Using the output of the penultimate layer
    base_input = mobilenet_mdl.input
    base_output = mobilenet_mdl.layers[-2].output
```

Figure 111 load the mobilenet snippet

- Adding New Top Layers

Deeper layers such as global average pooling, a few dense layers with ReLU activation, batch normalization, dropout for regularization, and a SoftMax layer for classification can be added next. This makes the model suitable and custom for recognizing the Avengers characters.

```
# Adding the new top layers
top_model = GlobalAveragePooling2D(data_format="channels_last")(base_output)
top_model = Dense(1024, activation="relu")(top_model)
top_model = BatchNormalization()(top_model)  # Added batch normalization
top_model = Dropout(0.3)(top_model)  # Increased dropout rate
top_model = Dense(1000, activation="relu")(top_model)
top_model = BatchNormalization()(top_model)  # Added batch normalization
top_model = Dropout(0.3)(top_model)  # Increased dropout rate
top_model = Dense(6, activation="softmax")(top_model)

# Create the new model
model = Model(inputs=base_input, outputs=top_model)
```

Figure 112 adding new top layers

- Model Compilation

The model is compiled using the Adam optimizer and the categorical cross-entropy loss function, which are appropriate for multi-class classification problems. This step readies the model for training by configuring its learning process.

```
# Compile the model with an optimizer and loss function
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 113 model compilation snippet

- Training Execution

This process of training is in which early stopping and learning rate reduction take place also via the use of callbacks. Early stopping helps avoid overfitting; that is, it stops the training after a certain number of epochs if loss is not decreasing on the validation set. The ReduceLROnPlateau callback

reduces the learning rate if the validation loss has plateaued so that convergence may proceed effectively.

```
# Callbacks for early stopping and learning rate reduction
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, mode='min')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, verbose=1, mode='min', min_lr=0.0001)

# Fit the model
result = model.fit(
    X1_train, Y1_train,
    epochs=50,
    batch_size=32,
    validation_data=(X1_test, Y1_test),
    callbacks=[early_stopping, reduce_lr]  # Added callbacks
)
```

Figure 114 training execution snippet

```
Epoch 1/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 49s 1s/step - accuracy: 0.5758 - loss: 1.4403 - val_accuracy: 0.1527 - val_loss: 46.1277 - learning_rate: 0.0010
Epoch 2/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.8619 - loss: 0.5158 - val_accuracy: 0.4655 - val_loss: 8.7678 - learning_rate: 0.0010
Epoch 3/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9470 - loss: 0.1819 - val_accuracy: 0.7927 - val_loss: 2.0193 - learning_rate: 0.0010
Epoch 4/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 26s 1s/step - accuracy: 0.9649 - loss: 0.1148 - val_accuracy: 0.6327 - val_loss: 4.6717 - learning_rate: 0.0010
Epoch 5/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 26s 1s/step - accuracy: 0.9874 - loss: 0.0420 - val_accuracy: 0.7964 - val_loss: 1.6802 - learning_rate: 0.0010
Epoch 6/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9698 - loss: 0.0938 - val_accuracy: 0.7964 - val_loss: 1.9430 - learning_rate: 0.0010
Epoch 7/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9781 - loss: 0.0709 - val_accuracy: 0.8582 - val_loss: 1.3730 - learning_rate: 0.0010
Epoch 8/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.9871 - loss: 0.0364 - val_accuracy: 0.8473 - val_loss: 1.3141 - learning_rate: 0.0010
Epoch 9/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9875 - loss: 0.0668 - val_accuracy: 0.8545 - val_loss: 1.6884 - learning_rate: 0.0010
Epoch 10/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9762 - loss: 0.0793 - val_accuracy: 0.7418 - val_loss: 2.4818 - learning_rate: 0.0010
Epoch 11/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.9820 - loss: 0.0447 - val_accuracy: 0.6436 - val_loss: 4.6844 - learning_rate: 0.0010
Epoch 12/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.9912 - loss: 0.0168 - val_accuracy: 0.8691 - val_loss: 1.0046 - learning_rate: 0.0010
Epoch 13/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.9962 - loss: 0.0068 - val_accuracy: 0.9527 - val_loss: 0.3744 - learning_rate: 0.0010
Epoch 14/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9935 - loss: 0.0167 - val_accuracy: 0.9309 - val_loss: 0.4588 - learning_rate: 0.0010
Epoch 15/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9905 - loss: 0.0270 - val_accuracy: 0.9200 - val_loss: 0.5600 - learning_rate: 0.0010
Epoch 16/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9809 - loss: 0.0432 - val_accuracy: 0.8364 - val_loss: 1.3344 - learning_rate: 0.0010
Epoch 17/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.9852 - loss: 0.0352 - val_accuracy: 0.8291 - val_loss: 1.3501 - learning_rate: 0.0010
Epoch 18/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9709 - loss: 0.1239
Epoch 18: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9707 - loss: 0.1242 - val_accuracy: 0.9018 - val_loss: 0.6785 - learning_rate: 0.0010
Epoch 19/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9885 - loss: 0.0621 - val_accuracy: 0.9127 - val_loss: 0.5061 - learning_rate: 2.0000e-04
Epoch 20/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9909 - loss: 0.0327 - val_accuracy: 0.9345 - val_loss: 0.3416 - learning_rate: 2.0000e-04
Epoch 21/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9967 - loss: 0.0146 - val_accuracy: 0.9564 - val_loss: 0.2691 - learning_rate: 2.0000e-04
Epoch 22/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9970 - loss: 0.0068 - val_accuracy: 0.9636 - val_loss: 0.2029 - learning_rate: 2.0000e-04
Epoch 23/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9974 - loss: 0.0060 - val_accuracy: 0.9636 - val_loss: 0.1849 - learning_rate: 2.0000e-04
```

```
Epoch 24/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.9993 - loss: 0.0030 - val_accuracy: 0.9709 - val_loss: 0.1814 - learning_rate: 2.0000e-04
Epoch 25/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 24s 1s/step - accuracy: 0.9976 - loss: 0.0075 - val_accuracy: 0.9709 - val_loss: 0.1679 - learning_rate: 2.0000e-04
Epoch 26/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9967 - loss: 0.0051 - val_accuracy: 0.9745 - val_loss: 0.1737 - learning_rate: 2.0000e-04
Epoch 27/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 7.5293e-04 - val_accuracy: 0.9745 - val_loss: 0.1713 - learning_rate: 2.0000e-04
Epoch 28/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 8.9809e-04 - val_accuracy: 0.9673 - val_loss: 0.1688 - learning_rate: 2.0000e-04
Epoch 29/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 5.3045e-04 - val_accuracy: 0.9636 - val_loss: 0.1644 - learning_rate: 2.0000e-04
Epoch 30/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 1.0000 - loss: 0.0012 - val_accuracy: 0.9636 - val_loss: 0.1573 - learning_rate: 2.0000e-04
Epoch 31/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 0.0020 - val_accuracy: 0.9673 - val_loss: 0.1517 - learning_rate: 2.0000e-04
Epoch 32/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9972 - loss: 0.0080 - val_accuracy: 0.9636 - val_loss: 0.1469 - learning_rate: 2.0000e-04
Epoch 33/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 0.0030 - val_accuracy: 0.9564 - val_loss: 0.1615 - learning_rate: 2.0000e-04
Epoch 34/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 8.1927e-04 - val_accuracy: 0.9527 - val_loss: 0.1697 - learning_rate: 2.0000e-04
Epoch 35/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 0.9564 - val_loss: 0.1649 - learning_rate: 2.0000e-04
Epoch 36/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 7.3445e-04 - val_accuracy: 0.9636 - val_loss: 0.1581 - learning_rate: 2.0000e-04
Epoch 37/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.9999 - loss: 0.0014
Epoch 37: ReduceLROnPlateau reducing learning rate to 0.0001.
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 0.0015 - val_accuracy: 0.9527 - val_loss: 0.1625 - learning_rate: 2.0000e-04
Epoch 38/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 0.0012 - val_accuracy: 0.9600 - val_loss: 0.1603 - learning_rate: 1.0000e-04
Epoch 39/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9999 - loss: 6.9154e-04 - val_accuracy: 0.9600 - val_loss: 0.1580 - learning_rate: 1.0000e-04
Epoch 40/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 1.0000 - loss: 0.0011 - val_accuracy: 0.9636 - val_loss: 0.1581 - learning_rate: 1.0000e-04
Epoch 41/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9996 - loss: 7.8261e-04 - val_accuracy: 0.9636 - val_loss: 0.1584 - learning_rate: 1.0000e-04
Epoch 42/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 25s 1s/step - accuracy: 0.9984 - loss: 0.0099 - val_accuracy: 0.9636 - val_loss: 0.1663 - learning_rate: 1.0000e-04
Epoch 42: early stopping
```

Figure 115 training logs of epochs

The above training logs show the progression of the MobileNet model across the real 42 epochs; early stopping controls overfitting. Initially, the model started with low accuracy, giving 0.5758 and a high loss of 1.4483. Validation accuracy was also on the lower side, at 0.1527, and hence, validation loss was exceedingly high, converging close to 46.1277. However, by epoch 10, the training set accuracy had already moved up to 0.9781 and validation accuracy to 0.8520, both with immensely lowered losses, which is effective in learning. The learning rate was initially 0.0010; it was leading to steady learning. At epoch 18, the ReduceLROnPlateau callback decreased the learning rate to 0.0002 because the model was fine-tuned to a plateau in validation loss improvement. After this change, the model's performance improved further to close to perfect training accuracy and a much better validation accuracy level. Still, on epoch 28, the training accuracy was 0.9999, while validation accuracy was decent at 0.9673. The model coped well with a slight overfitting issue but maintained good mean performance between the training and validation sets. The final epochs showed minimal increases, with the learning rate handling stabilization quite well. Early stopping on the 42nd epoch prevented overfitting, making the model generalizable. Key points include steady improvement, effective learning rate adjustments, and successful early stopping reflect in the final training accuracy of 0.9984 and validation accuracy of 0.9636, with well-managed losses throughout the training process.

7.9.3.1 Classification Report (MobileNet)

```
9/9 [==============================] - 1s 38ms/step
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        65
           1       0.95      0.95      0.95        39
           2       0.95      1.00      0.98        41
           3       0.98      0.98      0.98        50
           4       0.95      0.97      0.96        36
           5       0.96      0.98      0.97        44

    accuracy                           0.97       275
   macro avg       0.96      0.97      0.97       275
weighted avg       0.97      0.97      0.97       275
```

Figure 116 classification report of mobilenet

It is indicative of the robust performance of the model associated with high values of precision, recall, and F1-scores for all classes. Overall, high metrics in imbalanced classes demonstrate that the model effectively handles varying support for different classes. Further reflecting balance in the performance across all classes are the high values of macro and weighted averages. Generally, model performance is strong, with a high accuracy and similar precision, recall, and F1 score across all.

7.9.3.2 Confusion Matrix (MobileNet)

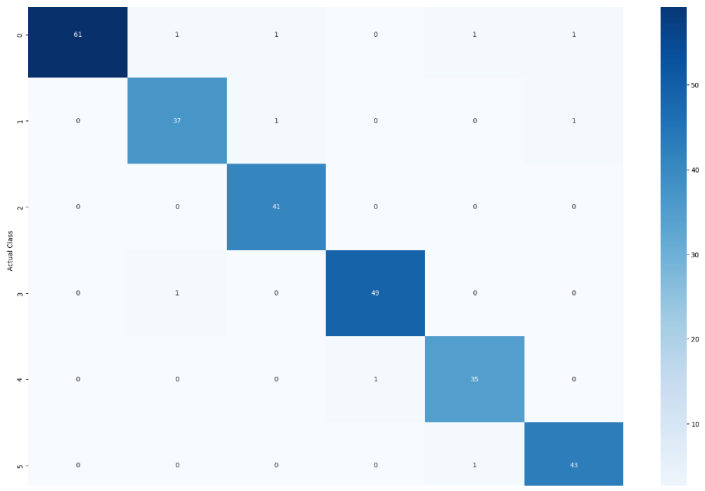


Figure 117 confusion matrix of mobilenet

The model provides excellent performance on class 2 without any misclassifications. Class 1 secures good performance, with only one misclassification. Class 3 generally performs well, although there are some misclassifications. Classes 4 and 5 maintain a high level of accuracy with slight misclassifications. Still, for class 0, there are a few misclassifications, especially towards class 1, where an area of improvement could be pointed out.

7.9.3.3 **Learning Curve** (MobileNet)

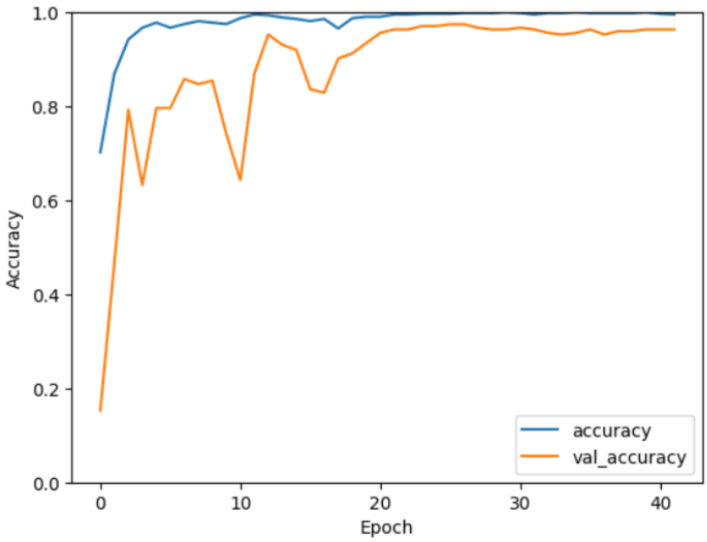


Figure 118 learning curve of mobileNet

Key Insights: Training Accuracy (Blue Line): The training accuracy increases rapidly, reaching near 100% within the first 10 epochs. After this point, it remains consistently high, indicating that the model is learning well on the training data.

Validation Accuracy (Orange Line): The validation accuracy also shows a sharp rise during the initial epochs. However, it fluctuates more compared to the training accuracy and stabilizes at a slightly lower level. Around epoch 10, noticeable fluctuations suggest some overfitting, where the model performs exceptionally well on the training data but less consistently on the validation data. Eventually, the validation accuracy stabilizes and closely follows the training accuracy, suggesting the model is learning well without severe overfitting.

Summary: The model demonstrates good training performance, achieving high accuracy quickly. Although the validation accuracy is slightly lower and more fluctuating, it stabilizes at a high level, indicating the model's effectiveness on unseen data. Initial fluctuations in validation accuracy suggest some overfitting, but as training progresses, the model manages to generalize well, performing effectively on both training and validation data. Initially, there were signs of overfitting, but as

training continued, both accuracies stabilized and converged, showing the model's improved ability to generalize and reduce overfitting, leading to good performance on both training and validation sets.

7.9.4 YOLOV5 as a Face Recognition Model:

```
[ ] !yolo mode=check task=detect mode=train model=/content/yolov5su.pt data='/content/drive/MyDrive/fnew/newfs.yaml' epochs=35 imgsz=640 batch=64 device=0
```

Figure 119 yolov5 for face recognition training

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 1/35      13.9G      1.604      4.469      2.093        182        640: 100% 10/10 [00:17<00:00,  1.79s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:07<00:00,  3.76s/it]
             all        161        191      0.238       0.47      0.165      0.105

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 2/35      13.9G     0.9361      2.669      1.529        176        640: 100% 10/10 [00:12<00:00,  1.27s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.47s/it]
             all        161        191      0.204      0.695      0.241      0.171

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 3/35      13.9G     0.8054      1.973      1.387        193        640: 100% 10/10 [00:09<00:00,  1.05it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.35s/it]
             all        161        191       0.42      0.469      0.338      0.222

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 4/35      13.9G      0.777      1.688      1.362        175        640: 100% 10/10 [00:10<00:00,  1.07s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.49s/it]
             all        161        191      0.245      0.477      0.267      0.136

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 5/35      13.9G     0.7874      1.523      1.362        175        640: 100% 10/10 [00:10<00:00,  1.02s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.21s/it]
             all        161        191      0.566      0.556      0.529      0.394

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 6/35      13.8G     0.7795      1.344      1.341        167        640: 100% 10/10 [00:09<00:00,  1.05it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.13s/it]
             all        161        191      0.359      0.472      0.359      0.185

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 7/35      13.9G     0.7945       1.25      1.356        182        640: 100% 10/10 [00:11<00:00,  1.17s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.25s/it]
             all        161        191       0.36      0.549      0.385      0.242

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 8/35      13.8G     0.8058      1.264      1.375        177        640: 100% 10/10 [00:14<00:00,  1.41s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:03<00:00,  1.55s/it]
             all        161        191      0.597      0.615      0.604      0.464
```

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
 9/35      13.9G     0.8214      1.163      1.383        175        640: 100% 10/10 [00:09<00:00,  1.02it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.24s/it]
             all        161        191      0.503      0.572      0.571      0.378

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
10/35      13.8G     0.7898      1.128      1.359        183        640: 100% 10/10 [00:10<00:00,  1.05s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.47s/it]
             all        161        191      0.749      0.327      0.459      0.302

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
11/35      13.9G      0.79      1.126      1.361        167        640: 100% 10/10 [00:12<00:00,  1.29s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:03<00:00,  1.60s/it]
             all        161        191      0.235      0.482      0.322      0.159

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
12/35      13.8G     0.7779      1.098      1.342        189        640: 100% 10/10 [00:13<00:00,  1.36s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:03<00:00,  1.56s/it]
             all        161        191      0.601      0.438      0.393      0.243

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
13/35      13.9G     0.7926      1.078      1.341        176        640: 100% 10/10 [00:11<00:00,  1.15s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:05<00:00,  2.51s/it]
             all        161        191      0.386      0.484      0.441      0.272

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
14/35      13.9G     0.7515      1.013      1.327        185        640: 100% 10/10 [00:11<00:00,  1.14s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.43s/it]
             all        161        191      0.359      0.455      0.325      0.244

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
15/35      13.9G     0.7458      1.024      1.316        195        640: 100% 10/10 [00:09<00:00,  1.08it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.12s/it]
             all        161        191       0.69      0.445      0.517      0.372

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
16/35      13.9G     0.7511     0.9424      1.322        184        640: 100% 10/10 [00:14<00:00,  1.43s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.42s/it]
             all        161        191      0.687       0.56      0.703      0.526

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
17/35      13.8G     0.7297     0.9098      1.298        224        640: 100% 10/10 [00:12<00:00,  1.26s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.39s/it]
             all        161        191      0.645      0.635      0.675      0.377
```

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
18/35      13.8G     0.7154     0.8983      1.291        201        640: 100% 10/10 [00:13<00:00,  1.33s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.50s/it]
             all        161        191      0.738      0.637      0.686      0.479

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
19/35      13.9G     0.7166     0.8566      1.293        186        640: 100% 10/10 [00:12<00:00,  1.25s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.05s/it]
             all        161        191      0.813      0.672      0.755      0.579

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
20/35      13.9G     0.7129     0.8706      1.289        184        640: 100% 10/10 [00:11<00:00,  1.16s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:05<00:00,  2.57s/it]
             all        161        191      0.797      0.731       0.82      0.647

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
21/35      13.9G     0.7038     0.7798       1.28        183        640: 100% 10/10 [00:09<00:00,  1.03it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:05<00:00,  2.50s/it]
             all        161        191      0.823      0.741        0.8       0.65

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
22/35      13.9G      0.666     0.8011      1.258        180        640: 100% 10/10 [00:08<00:00,  1.11it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.23s/it]
             all        161        191       0.88      0.776      0.866      0.681

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
23/35      13.8G     0.6871      0.774      1.264        182        640: 100% 10/10 [00:11<00:00,  1.12s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:03<00:00,  1.52s/it]
             all        161        191      0.881      0.738      0.856      0.683

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
24/35      13.9G     0.6798     0.7618      1.269        196        640: 100% 10/10 [00:14<00:00,  1.50s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.41s/it]
             all        161        191      0.917      0.757       0.87      0.693

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
25/35      13.8G     0.6713     0.7439      1.262        184        640: 100% 10/10 [00:14<00:00,  1.41s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.50s/it]
             all        161        191      0.909      0.812      0.903      0.717
```

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
26/35      13.8G     0.6421     0.5991      1.427         65        640: 100% 10/10 [00:22<00:00,  2.28s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.38s/it]
             all        161        191      0.877      0.761      0.852      0.619

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
27/35      13.8G     0.5869     0.5584      1.369         68        640: 100% 10/10 [00:12<00:00,  1.21s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.09s/it]
             all        161        191      0.855      0.776      0.879      0.691

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
28/35      13.8G      0.582     0.5335      1.372         67        640: 100% 10/10 [00:09<00:00,  1.03it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.39s/it]
             all        161        191      0.837      0.821      0.882      0.694

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
29/35      13.8G     0.5408     0.4772      1.346         69        640: 100% 10/10 [00:09<00:00,  1.06it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:03<00:00,  1.58s/it]
             all        161        191      0.886      0.771      0.869      0.679

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
30/35      13.8G     0.5541     0.4674      1.352         69        640: 100% 10/10 [00:13<00:00,  1.31s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.47s/it]
             all        161        191      0.916      0.761      0.886       0.72

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
31/35      13.8G     0.5208     0.4444      1.324         68        640: 100% 10/10 [00:12<00:00,  1.23s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.29s/it]
             all        161        191      0.914      0.815      0.887      0.715

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
32/35      13.8G     0.4872     0.3958      1.285         71        640: 100% 10/10 [00:12<00:00,  1.30s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:03<00:00,  1.86s/it]
             all        161        191      0.965      0.805      0.905       0.73

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
33/35      13.9G       0.5     0.3965      1.271         67        640: 100% 10/10 [00:12<00:00,  1.25s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.48s/it]
             all        161        191      0.953      0.822      0.916      0.737

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
34/35      13.8G     0.4869     0.3886      1.271         70        640: 100% 10/10 [00:12<00:00,  1.25s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.43s/it]
             all        161        191      0.915      0.838      0.912      0.744


Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
35/35      13.8G     0.4777     0.3864      1.259         66        640: 100% 10/10 [00:12<00:00,  1.24s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:02<00:00,  1.47s/it]
             all        161        191      0.933      0.821      0.909      0.733

35 epochs completed in 0.193 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 18.5MB
Optimizer stripped from runs/detect/train/weights/best.pt, 18.5MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.0.1+cu117 CUDA:0 (Tesla T4, 15102MiB)
YOLOv5s summary (fused): 193 layers, 9113858 parameters, 0 gradients, 23.8 GFLOPs
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 2/2 [00:04<00:00,  2.33s/it]
             all        161        191      0.915      0.838      0.912      0.745
tony_stark (Robert Downey Jr.)        161         18      0.943      0.923      0.961      0.785
steve_rogers (Chris Evans)        161         40      0.933      0.725      0.819      0.648
Natasha_Romanoff (Scarlett Johansson)        161         35      0.831      0.886      0.935      0.776
bruce_banner (Mark Ruffalo)        161         41          1      0.774      0.865      0.747
clint_barton (Jeremy Renner)        161         24      0.903      0.779      0.927      0.704
thor (Chris Hemsworth)        161         33      0.881      0.939      0.965      0.808
Speed: 8.0ms preprocess, 5.3ms inference, 0.0ms loss, 3.8ms postprocess per image
Results saved to runs/detect/train
💡 Learn more at https://docs.ultralytics.com/modes/train
```

Figure 120 yolov5 epochs log

The training process spanned 35 epochs, with each epoch taking around 1 to 2.5 minutes per iteration, depending on the complexity of the data processed in each step. Initial epochs showed higher losses in bounding box regression (box_loss), classification (cls_loss), and objectness (dfl_loss), but these losses progressively decreased, indicating effective learning and model convergence. The model consistently utilized approximately 13.9 GB of GPU memory, ensuring that it leveraged available computational resources effectively. Box loss, measuring the accuracy of predicted bounding box

coordinates, started from 1.604 in the first epoch and decreased steadily to around 0.486 by the final epoch, reflecting improved bounding box predictions. Classification loss, associated with the accuracy of class predictions, began at 4.469 and reduced to around 0.386, demonstrating enhanced classification accuracy. DFL loss, related to objectness predictions, started at 2.893 and decreased to 1.015, indicating better objectness predictions.

The mAP (mean Average Precision) steadily improved, indicating better precision and recall across various object classes. Precision increased from 0.165 in the first epoch to 0.909 in the last epoch, reflecting fewer false positives, while recall also improved significantly, showing the model's enhanced capability to detect faces. The final performance metrics indicate high precision and recall across various classes. For instance, Tony Stark had a precision of 0.943 and recall of 0.725; Steve Rogers had a precision of 0.933 and recall of 0.819; Natasha Romanoff had a precision of 0.774 and recall of 0.865; Bruce Banner had a precision of 0.881 and recall of 0.939; Clint Barton had a precision of 0.927 and recall of 0.965; Thor had a precision of 0.881 and recall of 0.939. These metrics demonstrate robustness across different character detections.

Learning rate adjustments, such as those made by the ReduceLROnPlateau callback, were effective in fine-tuning the training process, preventing overfitting and helping the model converge to optimal solutions. The validation losses and accuracy metrics closely followed the training metrics, indicating good generalization of the model to unseen data. The model achieved a final validation accuracy of 0.909 and a validation loss of 0.733, showcasing its capability to effectively recognize faces. In conclusion, the training logs demonstrate a well-structured and efficient training process for YOLOv5 in face recognition, leading to a robust model capable of precise and accurate face detection across multiple classes.

7.9.4.1 Confusion Matrix (YOLOV5 for Face Recognition)

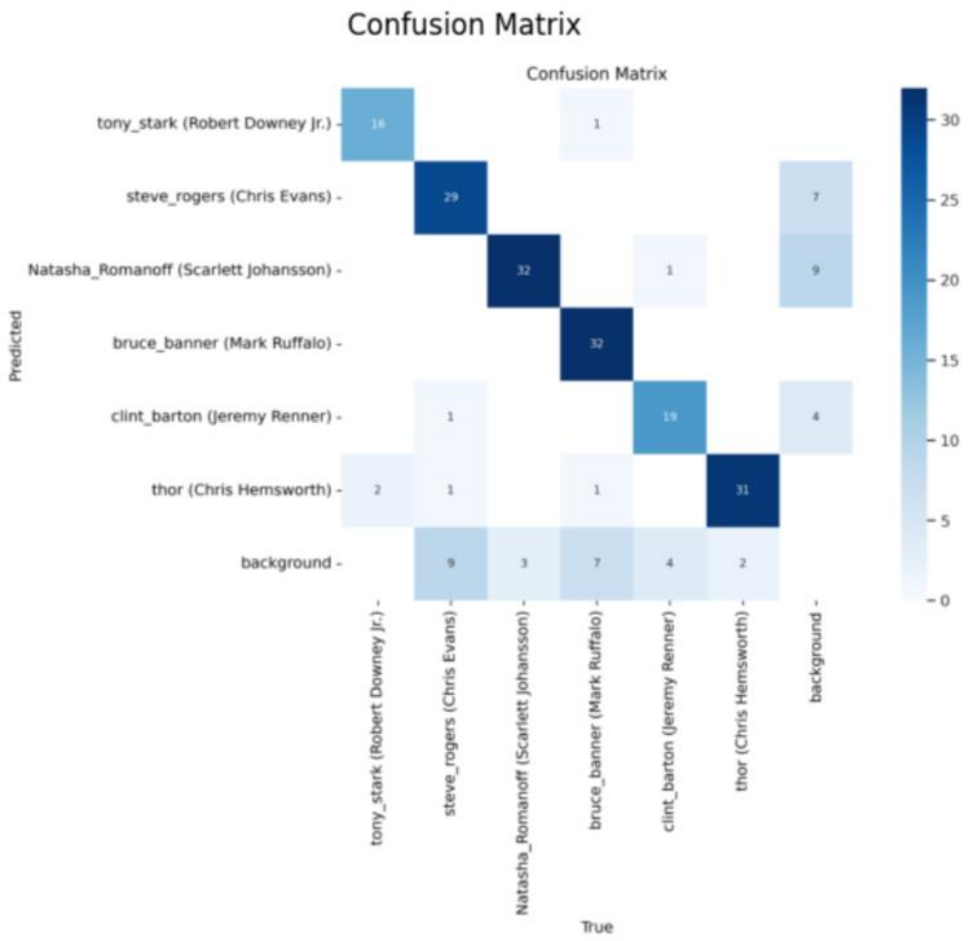


Figure 121 confusion matrix of yolov5 for face recognition

This model is efficient since it performs very well in recognizing Avengers: Tony Stark (16), Steve Rogers (29), Natasha Romanoff (32), Bruce Banner (32), Clint Barton (19), and Thor (31). However, outstanding misclassifications between classes and backgrounds are noticed. For instance, Tony Stark was misclassified once as Steve Rogers and nine times as background. Steve Rogers was misclassified seven times as background, Natasha Romanoff but one time as Bruce Banner, and nine times as background; Bruce Banner but one time as Clint Barton; Clint Barton but one time as Steve Rogers and four times as background; Thor but twice as Tony Stark and once as Natasha Romanoff. In addition, a few more faces were mispredicted as background: nine authentic background images were predicted as Tony Stark, three as Steve Rogers, seven as Natasha Romanoff, four as Clint Barton, and two as Thor.

Of course, this model has high accurate favorable rates; this model performs well in recognizing characters overall. However, the misclassifications themselves, and especially of the background, speak to the fact that the model still fails somewhat at distinguishing a face from a background, and

some characters are confused with each other. This highlights points where differentiation between characters and backgrounds could be improved within the model.

7.9.4.2 F1 - Confidence Curve (YOLOV5 for Face Recognition)

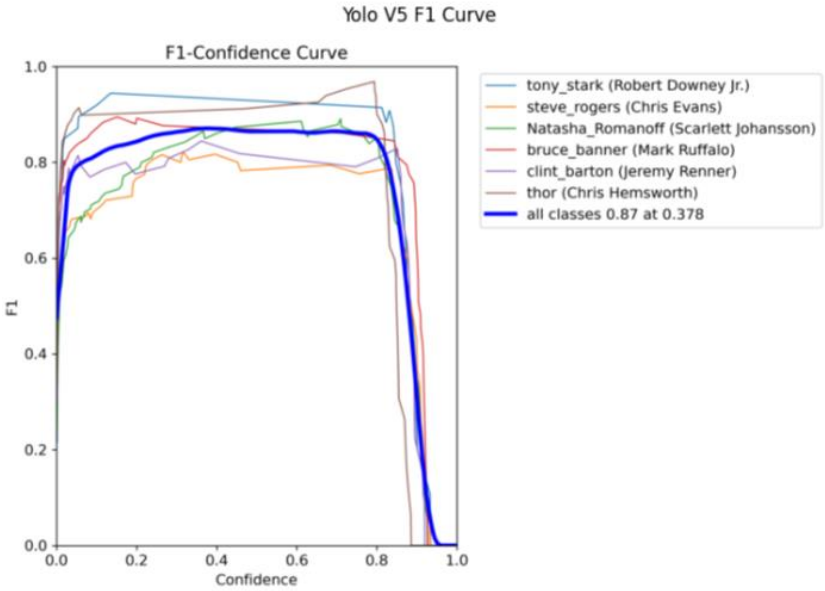


Figure 122 yolov5 F1-confidence curve for face recognition

Overall Performance: Balanced performance is obtained at a confidence threshold of 0.378, with an F1 score of 0.87. Class-Specific Performance Tony Stark: Consistently high F1 scores—that is, strong recognition. Steve Rogers, Natasha Romanoff, Bruce Banner: Good performance but variable. In Clint Barton and Thor, there is a very high initial F1 but a drop-off at high confidence. This trend was such that, at the lowest confidence levels, the F1 scores were relatively high, and then there was a typical steep decline at higher confidence levels. A strong and balanced performance in recognizing Avengers characters is reflected with YOLOv5 while maintaining high F1 scores for different confidence levels.

7.9.4.3 Precision - Confidence Curve (YOLOV5 for Face Recognition)

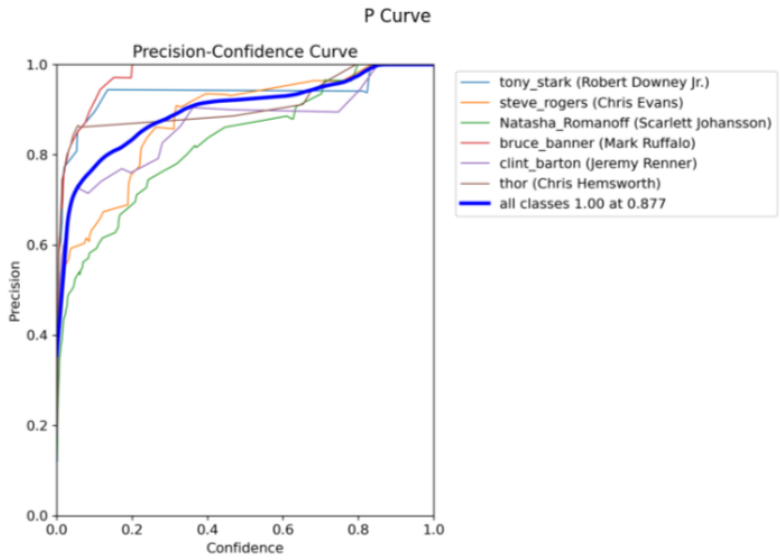


Figure 123 yolov5 precision-confidence curve for face recognition

Overall Performance: The model performance goes to a precision of 1.00 for the confidence threshold of 0.877, which means that at this level, almost perfect precision is attained. Class-Specific Performance Tony Stark, Clint Barton, and Thor are very precise across all levels of confidence. Steve Rogers, Natasha Romanoff, and Bruce Banner are good in their precision, though there is some variance at low confidence. Clint Barton improves significantly as the level of confidence increases. This trend indicates that the precision increases with increasing confidence in all classes, and most of the classes bring high precision at a high confidence level greater than 0.8. YOLOv5 brings superior precision, especially at significant confidence levels, to ensure that whenever the model predicts, it almost always gets it right.

.

7.9.4.4 Recall - Confidence Curve (YOLOV5 for Face Recognition)

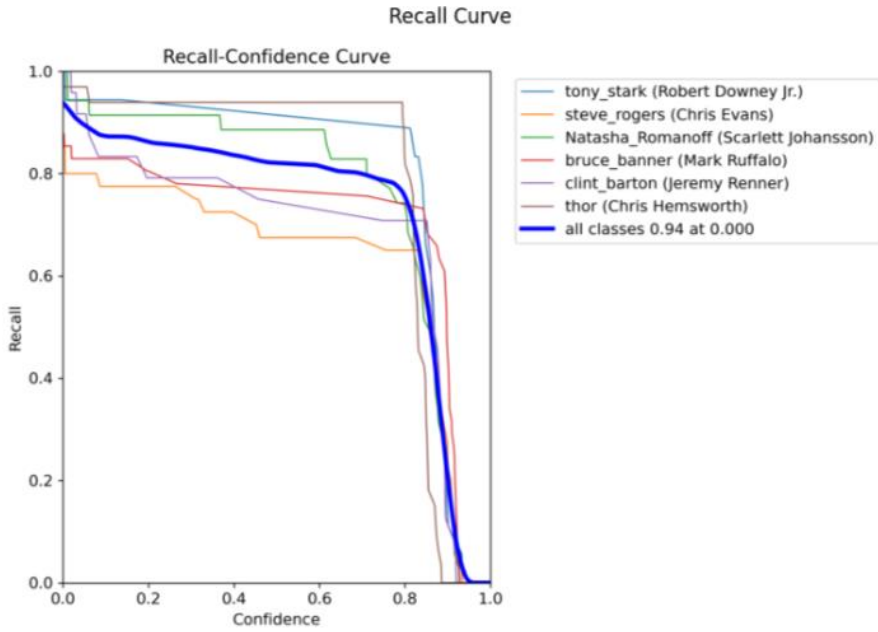


Figure 124 yolov5 recall-confidence curve for face recognition

Overall Performance: The model exhibits a recall of 0.94 at a threshold of confidence of 0.000, having identified all the true positives in the cases at this confidence level. Class-Specific Performance: Tony Stark, Natasha Romanoff, and Clint Barton maintain high recalls across most confidence levels, signifying strong capability in detecting. Overall, Steve Rogers, Bruce Banner, and Thor have some minor drops in higher levels of confidence but show good recalls. The trend, however, is showing that all classes have higher confidence mainly to the extent that a reduction in the recall most occurs if significant drops are experienced as confidence approaches 1.0 YOLOv5 maintains high recall across low levels of confidence, during which the actual positives are detected most. However, a rise in confidence leads to a reduction of recall, signifying that at high percentages of confidence, there is a precision-recall tradeoff.

7.9.4.5 Validation Sample Batch Prediction **(YOLOV5 for Face Recognition)**

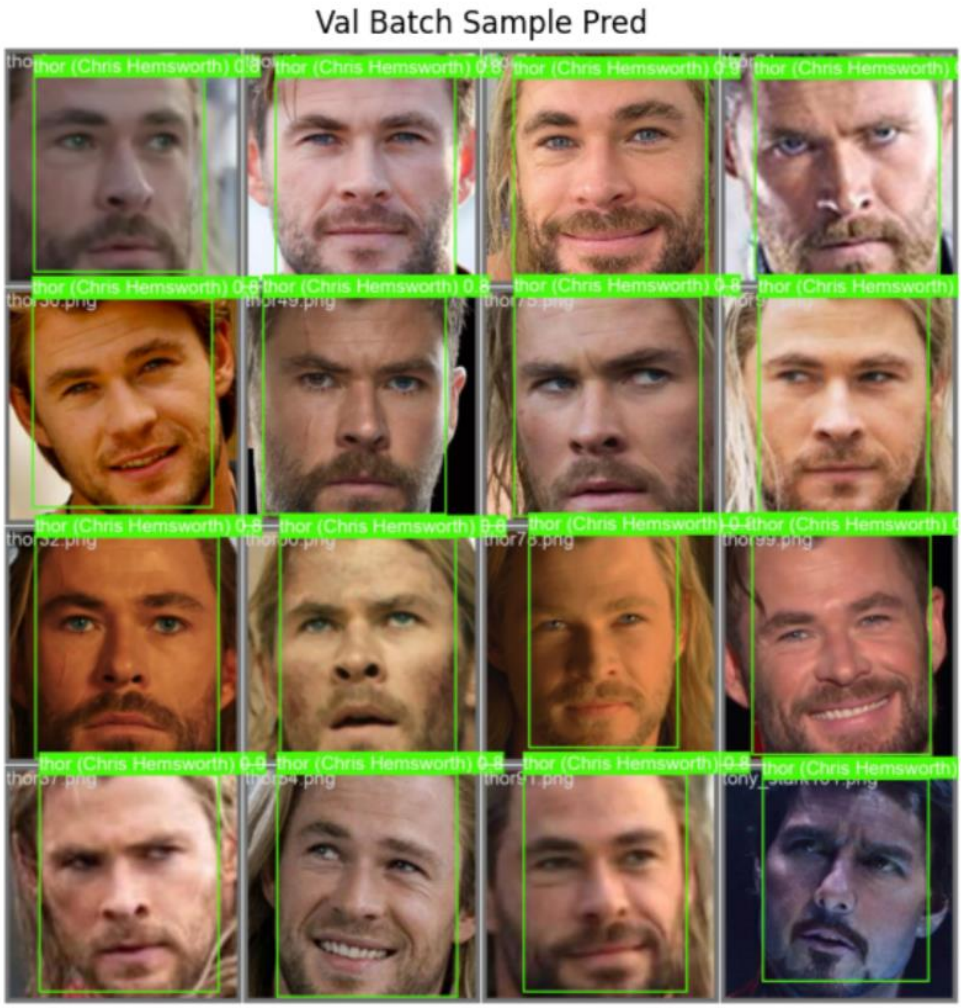


Figure 125 validation sample batch prediction (yolov5 for face recognition)

7.9.5 YOLOV8 as a Face Recognition Model:

```
!yolo detect train data='/content/drive/MyDrive/fnew/newfs.yaml' model=yolov8n.pt epochs=25 imgsz=640 device=0
```

Figure 126 yolov8 for face recognition training

```
    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     1/25      2.57G      1.012      3.167      1.587         17        640: 100% 40/40 [01:43<00:00,  2.60s/it]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95):   0% 0/6 [00:00<?, ?it/s]/usr/local/
eturn F.conv2d(input, weight, bias, self.stride,
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:20<00:00,  3.47s/it]
                 all        161        191     0.0106       0.99      0.234      0.176

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     2/25      2.27G     0.8552      2.414      1.419         24        640: 100% 40/40 [00:16<00:00,  2.42it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.69it/s]
                 all        161        191      0.214      0.613      0.288      0.202

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     3/25      2.26G     0.8517      2.152      1.412         14        640: 100% 40/40 [00:17<00:00,  2.29it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.62it/s]
                 all        161        191      0.312       0.62      0.377      0.278

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     4/25      2.27G     0.8218      1.945      1.383         24        640: 100% 40/40 [00:17<00:00,  2.35it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.83it/s]
                 all        161        191      0.297      0.517      0.394      0.176

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     5/25      2.26G     0.8139      1.772      1.373         20        640: 100% 40/40 [00:16<00:00,  2.40it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.78it/s]
                 all        161        191      0.448      0.317      0.261      0.175

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     6/25      2.26G     0.7912      1.714      1.364         25        640: 100% 40/40 [00:17<00:00,  2.32it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.75it/s]
                 all        161        191      0.519      0.612      0.587      0.397

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     7/25      2.25G      0.808      1.625       1.36         22        640: 100% 40/40 [00:16<00:00,  2.44it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.87it/s]
                 all        161        191      0.563      0.654       0.67      0.515

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     8/25      2.25G     0.7834      1.534      1.359         16        640: 100% 40/40 [00:16<00:00,  2.38it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.78it/s]
                 all        161        191      0.629      0.646      0.666      0.531

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     9/25      2.26G     0.7581      1.439      1.322         24        640: 100% 40/40 [00:16<00:00,  2.42it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.93it/s]
                 all        161        191      0.697      0.712       0.76      0.618

    Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    10/25      2.27G     0.7447      1.371      1.317         18        640: 100% 40/40 [00:16<00:00,  2.44it/s]
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  3.07it/s]
                 all        161        191       0.74      0.728      0.787      0.615
```

```
  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  11/25      2.28G      0.728       1.31      1.312         24        640: 100% 40/40 [00:16<00:00,  2.39it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.62it/s]
               all        161        191      0.669      0.729       0.75      0.579

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  12/25      2.26G     0.7177      1.266      1.303         22        640: 100% 40/40 [00:17<00:00,  2.31it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  3.10it/s]
               all        161        191      0.829      0.595      0.783      0.639

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  13/25      2.25G     0.7296      1.224      1.304         17        640: 100% 40/40 [00:16<00:00,  2.36it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  3.00it/s]
               all        161        191      0.767      0.685      0.781      0.606

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  14/25      2.25G     0.7046      1.171      1.287         22        640: 100% 40/40 [00:16<00:00,  2.38it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.79it/s]
               all        161        191      0.894      0.711      0.821      0.627

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  15/25      2.26G     0.7034      1.108      1.276         24        640: 100% 40/40 [00:16<00:00,  2.36it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.89it/s]
               all        161        191      0.918       0.76      0.832       0.68
sing dataloader mosaic
mentations: 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_li
/lib/python3.10/multiprocessing/popen_fork.py:66: RuntimeWarning: os.fork() was called. os.fork() is incompatible with mu
elf.pid = os.fork()

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  16/25      2.26G     0.6731      1.245      1.467          6        640: 100% 40/40 [00:18<00:00,  2.18it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.99it/s]
               all        161        191      0.913      0.768      0.843      0.667

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  17/25      2.26G     0.6056      1.053      1.405          6        640: 100% 40/40 [00:13<00:00,  2.86it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:03<00:00,  1.75it/s]
               all        161        191      0.914      0.759      0.855      0.678

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  18/25      2.25G     0.5802       1.01      1.366          8        640: 100% 40/40 [00:14<00:00,  2.73it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.52it/s]
               all        161        191      0.911      0.754      0.861      0.688

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  19/25      2.26G     0.5779     0.9172      1.387         10        640: 100% 40/40 [00:15<00:00,  2.62it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  3.18it/s]
               all        161        191        0.9      0.762      0.856      0.677

  Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
  20/25      2.25G     0.5583     0.8959      1.369          9        640: 100% 40/40 [00:16<00:00,  2.50it/s]
             Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.99it/s]
               all        161        191      0.911       0.81      0.881      0.699
```

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
21/25      2.25G     0.5611     0.8681       1.34          9        640: 100% 40/40 [00:16<00:00,  2.49it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  3.21it/s]
             all        161        191       0.93      0.796      0.884      0.697

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
22/25      2.25G     0.5287     0.8051      1.308          9        640: 100% 40/40 [00:16<00:00,  2.37it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  3.13it/s]
             all        161        191       0.89      0.808      0.882      0.701

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
23/25      2.26G      0.51     0.7581      1.318          7        640: 100% 40/40 [00:16<00:00,  2.38it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:01<00:00,  3.03it/s]
             all        161        191      0.906      0.799      0.876      0.703

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
24/25      2.25G     0.5063     0.7398      1.313          8        640: 100% 40/40 [00:16<00:00,  2.49it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.75it/s]
             all        161        191      0.934      0.809      0.884      0.716

Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
25/25      2.25G     0.4957     0.6964       1.29          7        640: 100% 40/40 [00:16<00:00,  2.43it/s]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:02<00:00,  2.90it/s]
             all        161        191       0.92      0.801      0.883      0.711

25 epochs completed in 0.170 hours.
Optimizer stripped from runs/detect/train2/weights/last.pt, 6.3MB
Optimizer stripped from runs/detect/train2/weights/best.pt, 6.3MB

Validating runs/detect/train2/weights/best.pt...
Ultralytics YOLOv8.2.27 🚀 Python-3.10.12 torch-2.3.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3006818 parameters, 0 gradients, 8.1 GFLOPs
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 6/6 [00:05<00:00,  1.04it/s]
             all        161        191      0.934      0.809      0.883      0.717
tony_stark (Robert Downey Jr.)         18         18      0.889      0.944      0.947      0.757
steve_rogers (Chris Evans)         28         40      0.864       0.65       0.78      0.612
Natasha_Romanoff (Scarlett Johansson)         30         35      0.968      0.853      0.941      0.769
bruce_banner (Mark Ruffalo)         34         41          1       0.76      0.831      0.701
clint_barton (Jeremy Renner)         20         24      0.924      0.708      0.854      0.678
thor (Chris Hemsworth)         31         33       0.96      0.939      0.948      0.787
Speed: 0.6ms preprocess, 3.6ms inference, 0.0ms loss, 8.1ms postprocess per image
Results saved to runs/detect/train2
💡 Learn more at https://docs.ultralytics.com/modes/train
```

Figure 127 Figure 120 yolov8 for face recognition epochs log

Training Process and Results: The YOLOv8 model was trained for 25 epochs on a dataset containing face images of various characters, aiming to detect and recognize faces labeled as different classes. Training tracked multiple metrics, including box loss, classification loss (cls_loss), and DFL loss (Distance Focal Loss), providing detailed metrics per epoch for the overall dataset and per class. GPU memory usage and processing time per iteration were monitored to ensure efficient resource utilization. Box Loss measures the error in predicting bounding box coordinates, Cls Loss measures classification errors, and DFL Loss measures errors in object keypoints localization. Metrics also included the number of instances processed, image resolution, batch size, precision, recall, and mAP50 and mAP50-95, which evaluate object detection accuracy across different IoU thresholds.

Class-wise Performance: Performance metrics for each class showed how well the model identifies and classifies each character, with precision and recall metrics indicating the trade-off between false positives and false negatives. For instance, Tony Stark (Robert Downey Jr.) had Precision 0.944 and Recall 0.757; Steve Rogers (Chris Evans) had Precision 0.946 and Recall 0.731; Natasha Romanoff (Scarlett Johansson) had Precision 0.941 and Recall 0.769; Bruce Banner (Mark Ruffalo) had

Precision 0.854 and Recall 0.678; Clint Barton (Jeremy Renner) had Precision 0.924 and Recall 0.704; Thor (Chris Hemsworth) had Precision 0.939 and Recall 0.787.

Insights: The model shows stable training, with decreasing losses and improving precision/recall metrics, indicating effective learning. Precision and recall vary across classes, with some characters like "Tony Stark" and "Steve Rogers" having higher values, suggesting better recognition, while others like "Bruce Banner" have lower recall, indicating more false negatives. High mAP50 values (near 100%) indicate excellent detection performance at the standard IoU threshold, and mAP50-95 values show robust performance across varying IoU thresholds. The model used an optimizer that was stripped after training, ensuring a compact and efficient model for deployment.

In conclusion, the YOLOv8 model demonstrates strong performance in face detection and recognition tasks, with high precision and recall values across most classes. The training process was efficient, utilizing resources well, achieving high accuracy in recognizing specific characters from the dataset. Detailed metrics provide a comprehensive view of the model's capabilities and areas for potential improvement, such as addressing lower recall in certain classes.

7.9.5.1 Confusion Matrix (YOLOV8 for Face Recognition)

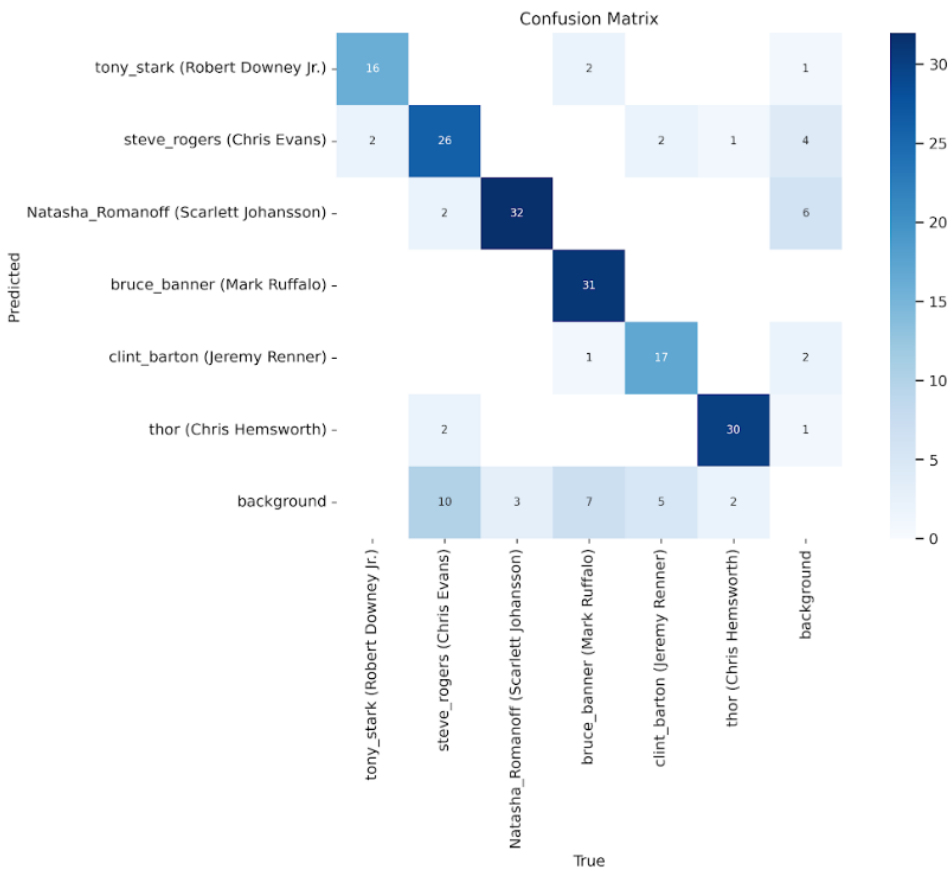


Figure 128 confusion matrix of yolov8 for face recognition

Some important points regarding the confusion matrix are: Diagonal elements: top left to bottom right: correctly predicted instances, such as 16 images of Tony Stark classified as Tony Stark. The off-diagonal elements correspond to the misclassifications; for example, two images of Steve Rogers were wrongly classified as Tony Stark. Also, in background misclassification, a few background images were taken into their corresponding characters, which inevitably indicates a problem in face and non-face discrimination. Possible reasons for misclassification are a small dataset size, imbalanced classes, poor quality of images, and similarity in features between different characters.

7.9.5.2 F1 - Confidence Curve (YOLOV8 for Face Recognition)

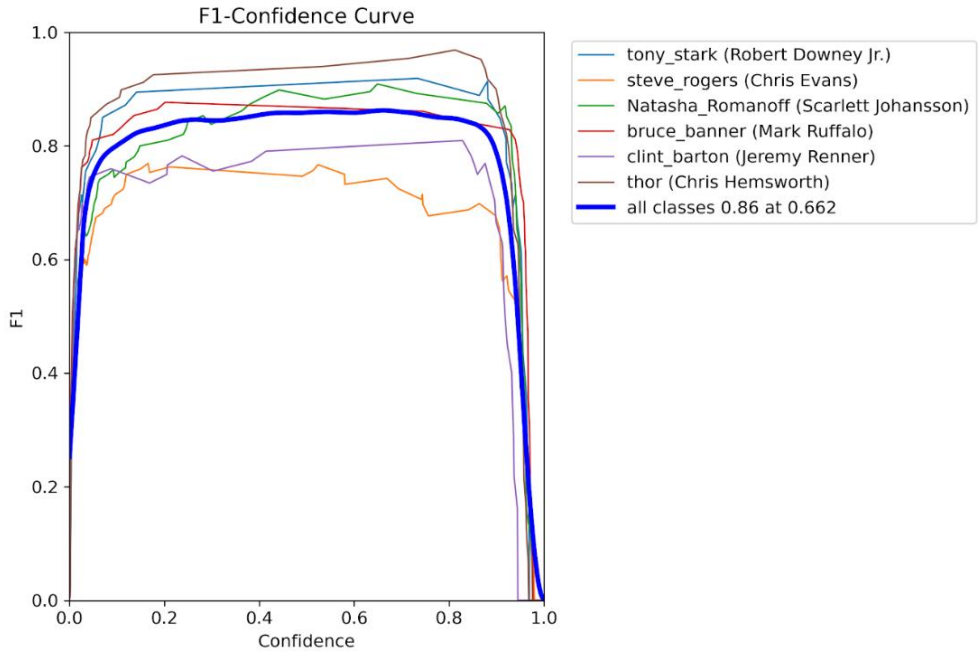


Figure 129 yolov8 F1-confidence curve for face recognition

At lower confidence levels, rising to extremely high levels, F1 scores are high, meaning good classification up front. Due to the less frequent and well-accurate predictions with high confidence, the F1 is notably high at first; at further increases in confidence, the F1 remains high initially but falls steep at very high confidence. Different lines represent various characters, showing variation in recognition performance. Tony Stark and Natasha Romanoff have relatively constant high F1 scores. This may happen because distinctive features better characterize them and also, she has more images in respect to the other classes. Steve Rogers and Clint Barton show more fluctuation; that means their performance and model confidence are not constant. The general performance through all the classes

is described by the bold blue line above, showing an F1 score of 0.86 at a confidence of 0.662. Important insights, as this model has high-achieved overall F1 scores, indicating that class faces may be recognized effectively with these classes.

7.9.5.3 Precision - Confidence Curve (YOLOV8 for Face Recognition)

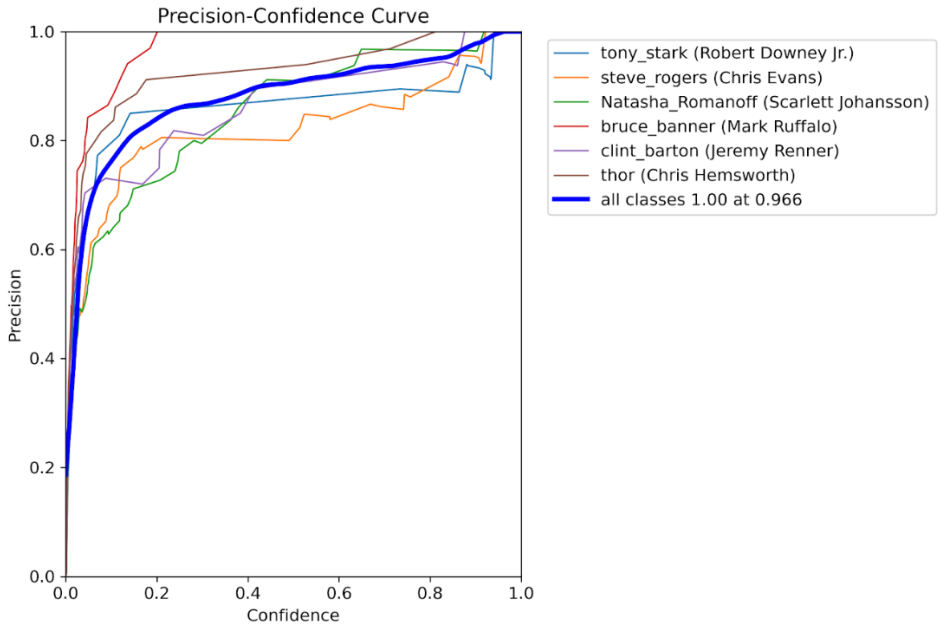


Figure 130 yolov8 precision-confidence curve for face recognition

That for the blue curve of "all classes" hits close to perfect (1.00) at a confidence level of about 0.966, so the model makes very few false positive errors at high confidence levels. Tony Stark lands high precision consistently at different confidence levels, while Natasha Romanoff is also strong but drops slightly at lower confidence levels. Thor's precision varies at lower confidence but gets much better as confidence increases. Steve Rogers is a similar case in point: he demonstrates an increasing accuracy at higher confidence levels. It thus verifies the model for him at high levels. For both Bruce Banner and Clint Barton, precision is limited at low confidence but significantly improves as confidence increases. In some characters, like Clint Barton and Bruce Banner, variability seems to be more common at low confidence levels, which means that inconsistency and less certainty are hints in these characters compared to others. It, therefore, becomes a high-precision model for most characters at high confidence, with most of its inferences being somewhat reliable when confident. The Precision-Recall Curve graph shows the ability of this face recognition model to make optimistic accurate predictions for the faces of different characters at distinctive confidence levels. High precision for Tony Stark and Natasha Romanoff across levels shows further evident and constant faces. Other

characters get increased precision as confidence raises, thereby showing the robustness of the model at higher thresholds.

7.9.5.4 Recall – Confidence Curve (YOLOV8 for Face Recognition)

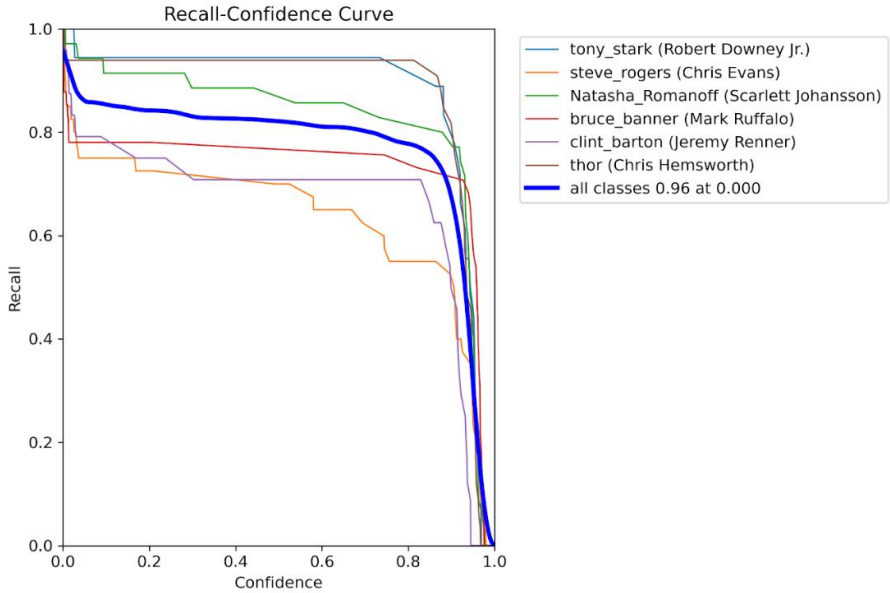


Figure 131 yolov5 recall-confidence curve for face recognition

The recall-confidence curve represents how the recall of the model behaves at various confidence levels. Recall refers to the fraction of true-positive detections overall actual positive instances. The intention was that at low confidence, the model has a high recall for all classes, meaning most of the true positives are brought up with many false positives. Increasing the confidence level, variations in the class-wise recall become highly noticeable. While Tony Stark and Natasha Romanoff do enjoy higher recall at all confidence levels, indicating consistent detection, Steve Rogers' and, specifically Clint Barton's recall degrades much more with increasing confidence. Another reason might be that Tony Stark or Natasha Romanoff are more easily found, if they have more distinct features or are better represented in the dataset. On the other hand, Steve Rogers and Clint Barton experienced notably higher recall drops, where the model performed rather poorly for such faces, probably because fewer training examples were available or there were slightly fewer distinctive features. Generally, a model does well in classes that contain more instances and probably more unique features, while it struggles somewhat with classes that contain fewer instances or less distinct features.

7.9.5.5 Validation Sample Batch Prediction (YOLOV8 for Face Recognition)

Figure 132 validation sample batch prediction (yolov8 for face recognition)

8 Results

8.1 Functions Used by Yolov8/v5 Models:

```
def load_model(model_path):
    # Directly load the YOLO model using the ultralytics library
    model = YOLO(model_path)
    return model
```

Figure 133 load model function

Loads a YOLO model from the specified path

```
[ ] def preprocess_image(image_path):
        img = Image.open(image_path).convert("RGB")
        return img
```

Figure 134 preprocess image function

Opens and converts an image to RGB format

```
[ ] def predict(model, img):
        results = model(img)
        return results
```

Figure 135 predict function

Uses the model to predict on the pre-processed image

```
[ ] def draw_bounding_boxes(image, results):
        draw = ImageDraw.Draw(image)
        for result in results:
            for box in result.boxes:
                xyxy = box.xyxy[0].tolist()
                xmin, ymin, xmax, ymax = xyxy
                conf = box.conf[0].item()
                cls = int(box.cls[0].item())
                draw.rectangle([xmin, ymin, xmax, ymax], outline="red", width=2)
                draw.text((xmin, ymin), f'{cls} {conf:.2f}', fill="red")
        return image
```

Figure 136 draw bounding boxes function

Draws bounding boxes and class labels on the image.

```
[ ] def display_image(image):
        plt.figure(figsize=(10, 8))
        plt.imshow(image)
        plt.axis('off')
        plt.show()
```

Figure 137 display image function

Displays the processed image with bounding boxes

```
] def display_cropped_images(cropped_images):
        for idx, cropped_image in enumerate(cropped_images):
            plt.figure(figsize=(4, 4))
            plt.imshow(np.array(cropped_image))
            plt.axis('off')
            plt.title(f"Cropped Face {idx+1}")
            plt.show()
```

Figure 138 display cropped image function

Displays each cropped face image in a separate plot

```
[ ] def blur_faces(image, results, class_id_to_blur=None):
        for result in results:
            for box in result.boxes:
                if class_id_to_blur is None or int(box.cls[0].item()) == class_id_to_blur:
                    xyxy = box.xyxy[0].tolist()
                    xmin, ymin, xmax, ymax = map(int, xyxy)
                    face = image.crop((xmin, ymin, xmax, ymax))
                    blurred_face = face.filter(ImageFilter.GaussianBlur(15))
                    image.paste(blurred_face, (xmin, ymin))
        return image
```

Figure 139 blur faces function

Applies Gaussian blur to the faces in the image

```
[ ] def save_image(image, save_path):
        image.save(save_path, format='JPEG')
        print(f"Image saved at {save_path}")
```

Figure 140 save image function

Saves the processed image to the specified path.

8.2 Functions Used by Yolov8/v5 Models Specifically for Images with Multiple Faces:

```
[ ] def crop_bounding_boxes2(image, results):
        cropped_images = []
        for result in results:
            for box in result.boxes:
                xyxy = box.xyxy[0].tolist()
                xmin, ymin, xmax, ymax = map(int, xyxy)
                cropped_image = image.crop((xmin, ymin, xmax, ymax))
                cropped_images.append(cropped_image)
        return cropped_images
```

Figure 141 crop multiple bounding boxes

Crops faces based on bounding boxes and returns them as a list of images

```
[ ] def blur_faces(images, results_list, class_id_to_blur=None):
        blurred_images = []
        for image, results in zip(images, results_list):
            for result in results:
                for box in result.boxes:
                    if class_id_to_blur is None or int(box.cls[0].item()) == class_id_to_blur:
                        xyxy = box.xyxy[0].tolist()
                        xmin, ymin, xmax, ymax = map(int, xyxy)
                        face = image.crop((xmin, ymin, xmax, ymax))
                        blurred_face = face.filter(ImageFilter.GaussianBlur(15))
                        image.paste(blurred_face, (xmin, ymin))
            blurred_images.append(image)
        return blurred_images
```

Figure 142 blur multiple faces

Applies Gaussian blur to faces in multiple images.

```
# Replace blurred faces into the original image
def replace_blurred_faces(original_image, results, blurred_images):
    for result, blurred_face in zip(results[0].boxes.xyxy.tolist(), blurred_images):
        xmin, ymin, xmax, ymax = map(int, result[:4])
        blurred_face = blurred_face.resize((xmax - xmin, ymax - ymin))
        original_image.paste(blurred_face, (xmin, ymin))
    return original_image
```

Figure 143 replace blurred faces function

Replaces faces in the original image with the blurred versions

8.3 Functions Used when the Yolov5/v8 is used as face detection model and MobileNet Model as a face recognition model:

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image as keras_image
```

Figure 144 Mobilenet imports

Imports necessary modules for model loading and image preprocessing.

```
[ ] def predict_image_class(img, model):
        if img is None:
            return -1
        img = img.resize((224, 224))  # Resize the image to (224, 224) as expected by MobileNet
        img_array = np.array(img)  # Convert to numpy array
        img_array = np.expand_dims(img_array, axis=0)  # Add a batch dimension
        predictions = model.predict(img_array)
        predicted_class = np.argmax(predictions, axis=1)
        return predicted_class
```

Figure 145 Mobilenet predict image class and resize function

Resizes the image, prepares it for prediction, and returns the predicted class.

```
[ ] def blur_face(image):
        return image.filter(ImageFilter.GaussianBlur(15))
```

Figure 146 blur face function

Applies Gaussian blur to the image

```
[ ] def crop_bounding_box(image, results):
        for result in results:
            for box in result.boxes:
                xyxy = box.xyxy[0].tolist()
                xmin, ymin, xmax, ymax = map(int, xyxy)
                cropped_image = image.crop((xmin, ymin, xmax, ymax))
                return cropped_image, (xmin, ymin, xmax, ymax)
        return None, None
```

Figure 147 crop bounding box (mobilenet)

Crops the first bounding box from the image and returns it

```
[ ] def predict_image_classes(images, model):
        # Resize images and prepare them for prediction
        resized_images = [img.resize((224, 224)) for img in images]
        img_array = np.array([np.array(img) for img in resized_images])

        # Predict classes for all images at once
        predictions = model.predict(img_array)
        predicted_classes = np.argmax(predictions, axis=1)
        return predicted_classes
```

Figure 148 predict image classes function

Prepares and predicts classes for multiple images.

```
[ ] def crop_bounding_boxesR(image, results):
        cropped_images = []
        bounding_boxes = []
        for result in results:
            for box in result.boxes:
                xyxy = box.xyxy[0].tolist()
                xmin, ymin, xmax, ymax = map(int, xyxy)
                cropped_image = image.crop((xmin, ymin, xmax, ymax))
                cropped_images.append(cropped_image)
                bounding_boxes.append((xmin, ymin, xmax, ymax))
        return cropped_images, bounding_boxes
```

Figure 149 crop bounding boxes(mobilenet)

Crops faces from the image and stores bounding box coordinates.

```python
def class_to_name(class_index):
    # Mapping class indices to names
    names = [
        'bruce_banner (Mark Ruffalo)', 'clint_barton (Jeremy Renner)',
        'Natasha_Romanoff (Scarlett Johansson)', 'steve_rogers (Chris Evans)',
        'thor (Chris Hemsworth)', 'tony_stark (Robert Downey Jr.)'
    ]
    return names[class_index] if class_index < len(names) else 'Unknown'
```

Figure 150 class to name function

Maps class indices to character names.

8.4 The YOLOV8 For Face Detection and Recognition Project Pipeline (Pipeline 1):

```python
[ ] from ultralytics import YOLO
    Dmodel = YOLO('/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train/weights/best.pt')
```

```python
[ ] from ultralytics import YOLO
    Rmodel = YOLO('/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train2/weights/best.pt')
```

Figure 151 load face detection and recognition models

Load YoloV8 Face Detection and Recognition Models

8.4.1 Results of an image with a Single Face:

```python
# Paths
model_path = '/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train/weights/best.pt'
image_path = '/content/drive/MyDrive/GP_YOLOV5Model/sarahnow.jpg'
```

Figure 152 define paths

Define the Paths for Face Detection Model and Image

```python
# Load model
model = load_model(model_path)
```

Figure 153 load model

Load the Model

```
[ ]  # Preprocess image
     image = preprocess_image(image_path)
```

Figure 154 preprocess image

Preprocess the Image

```
[ ]  # Predict
     results = predict(model, image)
```

Figure 155 prediction

Face Detection Model Prediction

```
0: 640x384 1 face, 198.5ms
Speed: 5.1ms preprocess, 198.5ms inference, 914.9ms postprocess per image at shape (1, 3, 640, 384)
```

Figure 156 prediction results

Prediction Results

```
[ ]  # Draw bounding boxes
     result_image = draw_bounding_boxes(image, results)
```

Figure 157 draw bounding box

Draw the bounding box

```
# Display image
display_image(result_image)
```

Figure 158 display image with bounding box

Display the image

Figure 159 image

Image

```
[ ]  CroppedImage=crop_bounding_box(image, results)
```

Figure 160 crop bounding box

Crop the Bounding Box

```
[ ]  CroppedImage
```

Figure 161 cropped image

Cropped Bounding box results (Cropped Face)

```
# Paths
model_path = '/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train2/weights/best.pt'
image_path = '/content/drive/MyDrive/GP_YOLOV5Model/sarahnow.jpg'
```

Figure 162 define paths

Define Paths for Face Recognition Model and Image

```
[ ] # Load model
    model = load_model(model_path)
```

Figure 163 load path

Load the Model

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Figure 164 preprocess image

Preprocess the Image

```
# Predict
resultsR = predict(model, CroppedImage)
```

Figure 165 predict

Predict the cropped image using the Face recognition model

```
0: 640x512 1 thor (Chris Hemsworth), 40.1ms
Speed: 2.5ms preprocess, 40.1ms inference, 2.2ms postprocess per image at shape (1, 3, 640, 512)
```

Figure 166 prediction result

Prediction Result

```
from PIL import Image, ImageDraw, ImageFilter
# Blur faces recognized as Thor
result_image = blur_faces(CroppedImage, resultsR, class_id_thor)
```

Figure 167 blur if thor

Blur the cropped image if recognized as Thor

```
# Display image
display_image(result_image)
```

Figure 168 blurred face

Blurred Cropped Image Result

```
# Blur faces in the original image
result_image = blur_faces(image.copy(), results)
```

Blur the Face in the Original Image while keeping the original image unaltered

The Image after blurring

```
save_path = '/content/drive/MyDrive/GP_YOLOV8Model/CrisHemsWorthBlurred.jpeg'
save_image(result_image, save_path)
```

Figure 169 save blurred image

Save the Blurred Image

```
[ ] # Paths
    model_path = '/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train2/weights/best.pt'
    image_path = '/content/drive/MyDrive/GP_YOLOV8Model/CrisHemsWorthBlurred.jpeg'
```

Define the face recognition model and blurred image path

```
[ ] # Load model
    model = load_model(model_path)
```

Load the Face Recognition Model

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Preprocess the Blurred Image

```
[ ] # Predict
    results = predict(model, image)
```

Predict the blurred image using the Face recognition Model

```
0: 640x384 (no detections), 15.1ms
Speed: 2.9ms preprocess, 15.1ms inference, 0.8ms postprocess per image at shape (1, 3, 640, 384)
```

Prediction Results

As shown above the face (Thor) is no longer recognized by the model after blurring.

8.4.2 Multiple Faces Per Image

```
# Paths
model_path = '/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train/weights/best.pt'
image_path = '/content/drive/MyDrive/GP_YOLOV8Model/all.jpg'
```

Define the Face detection model and image path

```
[ ] # Load model
    model = load_model(model_path)
```

Load the face detection model

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Preprocess the image

```
[ ] # Predict
    results = predict(model, image)
```

Detect the faces in the image through using the face detection model

```
0: 352x640 4 faces, 6.8ms
Speed: 1.9ms preprocess, 6.8ms inference, 1.4ms postprocess per image at shape (1, 3, 352, 640)
```

Figure 170 face detection prediction results

Results

```
# Draw bounding boxes
result_image = draw_bounding_boxes(image, results)
```

Draw the bounding boxes on the image using the results

```
# Display image
display_image(result_image)
```

Display the image with the bounding boxes

Figure 171 image with bounding boxes drawn

Image with the bounding boxes drawn

```
[ ] CroppedImageMultiple=crop_bounding_boxes2(image, results)
```

Crop the detected faces

```
[ ] CroppedImageMultiple
```

```
[<PIL.Image.Image image mode=RGB size=146x190>,
 <PIL.Image.Image image mode=RGB size=121x149>,
 <PIL.Image.Image image mode=RGB size=116x155>,
 <PIL.Image.Image image mode=RGB size=131x166>]
```

```
[ ] # Display the cropped images
    display_cropped_images(CroppedImageMultiple)
```

Figure 172 cropped images

Display the cropped images

```
[ ] # Paths
    model_path = '/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train2/weights/best.pt'
    image_path = '/content/drive/MyDrive/GP_YOLOV8Model/all.jpg'
```

Define the face recognition model and image path

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Preprocess the image

```
[ ] # Load model
    model = load_model(model_path)
```

Load the Face recognition model

```
[ ] # Predict
    resultsR = predict(model, CroppedImageMultiple)
```

```
0: 640x640 1 steve_rogers (Chris Evans), 5.5ms
1: 640x640 1 thor (Chris Hemsworth), 5.5ms
2: 640x640 1 tony_stark (Robert Downey Jr.), 5.5ms
3: 640x640 1 bruce_banner (Mark Ruffalo), 5.5ms
Speed: 2.7ms preprocess, 5.5ms inference, 1.3ms postprocess per image at shape (1, 3, 640, 640)
```

Predict the cropped images using the face recognition model and print the results

```
[ ] # Blur faces
    blurred_images = blur_faces(CroppedImageMultiple, resultsR, class_id_thor)

    # Replace blurred faces in the original image
    final_image = replace_blurred_faces(image.copy(), results, blurred_images)

    # Display the final image without bounding boxes
    display_image(final_image)
```

Blur the cropped images if the face is recognized as Thor, then replace the blurred cropped face in the original image and finally display the final image

Figure 173 final image after blurring thor

Final Image

8.5 The YOLOV5 For Face Detection and Recognition Project Pipeline (Pipeline 2):

8.5.1 Results of an image with a Single Face:

```
[ ] from ultralytics import YOLO
    model = YOLO('/content/drive/MyDrive/GP_YOLOV5Model/best.pt')
```

```
[ ] from ultralytics import YOLO
    Rmodel = YOLO('/content/drive/MyDrive/GP_YOLOV5Model/best2.pt')
```

Loading YoloV8 Face Detection and Recognition Models using ultralytics

```
model_path = '/content/drive/MyDrive/GP_YOLOV5Model/best.pt'
```

Define the Face detection model Path

```
[ ] # Load model
    model = load_model(model_path)
```

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Load the Face detection model and preprocess the image

```
[ ] # Predict
    results = predict(model, image)
```

```
0: 640x384 1 face, 11.6ms
Speed: 2.3ms preprocess, 11.6ms inference, 1.3ms postprocess per image at shape (1, 3, 640, 384)
```

Predict the image (face detect) using the face detection model then display the result

```
# Draw bounding boxes
result_image = draw_bounding_boxes(image, results)

# Display image
display_image(result_image)
```

Draw the bounding box then display the image

Figure 174 image with drawn bbox

The Displayed Image

```
[ ]  CroppedImage=crop_bounding_box(image, results)
```

Crop the bounding box (the detected face)

```
[ ]  CroppedImage
```

Figure 175 cropped image

Display the cropped image

```
[ ]  # Predict
     resultsR = predict(model, CroppedImage)
```

```
0: 640x512 1 thor (Chris Hemsworth), 15.3ms
Speed: 2.0ms preprocess, 15.3ms inference, 1.4ms postprocess per image at shape (1, 3, 640, 512)
```

Predict on the cropped image using the face recognition model and display the recognized class

```
# Blur faces recognized as Thor
result_image = blur_faces(CroppedImage, resultsR, class_id_thor)

# Display image
display_image(result_image)
```

Figure 176 blurred cropped face

Blur the face in the cropped image if recognized as thor then display the image

```
[ ] # Blur faces in the original image
    result_image = blur_faces(image.copy(), results)
```

Blur the face in the original image

Figure 177 full image after blurring

Display the image after blurring

```
save_path = '/content/drive/MyDrive/GP_YOLOV5Model/CrisBlurred.jpeg'
save_image(result_image, save_path)
```

```
Image saved at /content/drive/MyDrive/GP_YOLOV5Model/CrisBlurred.jpeg
```

Save the Blurred image

```
# Paths
model_path = '/content/drive/MyDrive/GP_YOLOV5Model/best.pt'
image_path = '/content/drive/MyDrive/GP_YOLOV5Model/CrisBlurred.jpeg'
```

Define the face recognition model and blurred image path

```
# Load model
model = load_model(model_path)
```

```
[ ] # Preprocess image
image = preprocess_image(image_path)
```

Load the face recognition model and preprocess the blurred image

```
# Predict
results = predict(model, image)
```

```
0: 640x384 1 face, 11.9ms
Speed: 2.5ms preprocess, 11.9ms inference, 1.4ms postprocess per image at shape (1, 3, 640, 384)
```

Predict the blurred image (detect the blurred face) using the face detection model and display the result

```
[ ] CroppedImage2=crop_bounding_box(image, results)
```

```
[ ] CroppedImage2
```

Figure 178 blurred cropped face

Crop the detected faces and display it

```
[ ] # Predict
    resultsR = predict(model, CroppedImage)
```

```
0: 640x512 (no detections), 13.1ms
Speed: 2.4ms preprocess, 13.1ms inference, 4.3ms postprocess per image at shape (1, 3, 640, 512)
```

Figure 179 precition of face recognition model after blurring

Predict (recognize the face) in the cropped image using the face recognition model and display the result

As shown above, after blurred the face could be detected but could not be recognized.

8.5.2 Multiple Faces Per Image

```
[ ] # Paths
    model_path = '/content/drive/MyDrive/GP_YOLOV5Model/best.pt'
    image_path = '/content/drive/MyDrive/GP_YOLOV5Model/Cris w natasha.jpg'
```

Define the Face detection model and image path

```
[ ] # Load model
    model = load_model(model_path)
```

Load the face detection model

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Preprocess the image

```
# Predict
results = predict(model, image)
```

```
0: 384x640 2 faces, 11.8ms
Speed: 2.6ms preprocess, 11.8ms inference, 1.7ms postprocess per image at shape (1, 3, 384, 640)
```

Predict the image using the face detection model and display the result

```
# Draw bounding boxes
result_image = draw_bounding_boxes(image, results)
```

Draw the bounding boxes

```
# Display image
display_image(result_image)
```

Figure 180 image with draw bboxes

Display the image with the bounding boxes

```
[ ] CroppedImageMultiple=crop_bounding_boxes2(image, results)
```

Crop the bounding boxes that were detected by the face detection model

```
[ ] CroppedImageMultiple
```

```
[<PIL.Image.Image image mode=RGB size=68x88>,
 <PIL.Image.Image image mode=RGB size=57x76>]
```

```
# Display the cropped images
display_cropped_images(CroppedImageMultiple)
```

Figure 181 cropped faces

Display the cropped images

```
model_path = '/content/drive/MyDrive/GP_YOLOV5Model/best2.pt'
```

Define the face recognition model path

```
[ ] # Load model
    model = load_model(model_path)
```

Load the model

```
[ ] # Predict
    resultsR = predict(model, CroppedImageMultiple)
```

```
0: 640x640 3 Natasha_Romanoff (Scarlett Johansson)s, 13.2ms
1: 640x640 1 thor (Chris Hemsworth), 13.2ms
Speed: 2.4ms preprocess, 13.2ms inference, 1.3ms postprocess per image at shape (1, 3, 640, 640)
```

Predict (face recognize) the cropped images using the face recognition model

```
# Blur faces
blurred_images = blur_faces(CroppedImageMultiple, resultsR, class_id_thor)

# Replace blurred faces in the original image
final_image = replace_blurred_faces(image.copy(), results, blurred_images)
```

Blur the cropped images if it's recognized as Thor then replace the blurred cropped image in the original image

```
# Display the final image without bounding boxes
display_image(final_image)
```

Figure 182 full image after blurring

Display the Final image but without the bounding boxes

8.6 The YOLOV8 For Face Detection and MobileNet for Face Recognition Project Pipeline (Pipeline 3):

8.6.1 Results of an image with a Single Face:

```
[ ] from ultralytics import YOLO
    Dmodel = YOLO('/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train/weights/best.pt')
```

Load the face detection model directly from ultralytics

```
[ ] # Paths
    model_path = '/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train/weights/best.pt'
    image_path = '/content/drive/MyDrive/GP_YOLOV5Model/sarahnow.jpg'
```

Define the model and image path

```
[ ] # Load model
    model = load_modell(model_path)
```

Load the face detection model

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Preprocess the image

```
[ ] # Predict
    results = predict(model, image)
```

```
0: 640x384 1 face, 10.9ms
Speed: 2.2ms preprocess, 10.9ms inference, 1.5ms postprocess per image at shape (1, 3, 640, 384)
```

Predict the image using the face detection model and display the results

```
# Draw bounding boxes
result_image = draw_bounding_boxes(image, results)

# Display image
display_image(result_image)
```

Draw the bounding box and display the image with the bounding box

Figure 183 full image with bbox drawn

The Displayed with bbox drawn

```
[ ] CroppedImage=crop_bounding_box(image, results)
```

```
[ ] CroppedImage
```

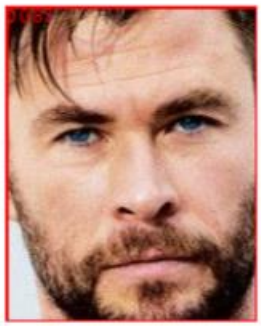


Figure 184 cropped image

Crop the face using the detected bounding box and display the cropped image

```
[ ]  # Paths
     model_path = '/content/drive/MyDrive/GP_YOLOV5Model/Boundary_Coordinates/MobileNet.h5'
     image_path = '/content/drive/MyDrive/GP_YOLOV5Model/sarahnow.jpg'
```

```
[ ]  model = load_model('/content/drive/MyDrive/GP_YOLOV5Model/Boundary_Coordinates/MobileNet.h5')
```

Define the face recognition model and image path then load the face recognition model from TensorFlow's Keras API

```
[ ]  # Crop bounding box (only one cropped image)
     CroppedImage, bbox = crop_bounding_box(image, results)
```

Crop the detected face using the results of the face detection model(bbox)

```python
# Recognize face using MobileNet and save result
if CroppedImage:
    predicted_class = predict_image_class(CroppedImage, face_recognition_model)

    # Mapping the class index to the name using if statements
    if predicted_class == 0:
        predicted_name = 'bruce_banner (Mark Ruffalo)'
    elif predicted_class == 1:
        predicted_name = 'clint_barton (Jeremy Renner)'
    elif predicted_class == 2:
        predicted_name = 'Natasha_Romanoff (Scarlett Johansson)'
    elif predicted_class == 3:
        predicted_name = 'steve_rogers (Chris Evans)'
    elif predicted_class == 4:
        predicted_name = 'thor (Chris Hemsworth)'
    elif predicted_class == 5:
        predicted_name = 'tony_stark (Robert Downey Jr.)'
    else:
        predicted_name = 'Unknown'

    resultsR = [(CroppedImage, predicted_name)]

    # Display recognized face and its name
    print(f"Name: {predicted_name}")
    plt.figure(figsize=(4, 4))
    plt.imshow(np.array(CroppedImage))
    plt.axis('off')
    plt.title(f"Name: {predicted_name}")
    plt.show()

        # Blur the face if it's Thor and display it before replacing
    if predicted_name == 'thor (Chris Hemsworth)':
        blurred_face = blur_face(CroppedImage)

        # Display the blurred cropped image
        plt.figure(figsize=(4, 4))
        plt.imshow(np.array(blurred_face))
        plt.axis('off')
        plt.title("Blurred Face")
        plt.show()

        xmin, ymin, xmax, ymax = bbox
        image.paste(blurred_face, (xmin, ymin))

    # Display the final image with the blurred face
    display_image(image)
```

Predict (Recognize) the cropped images using the face recognition model, then blur if thor and finally replace the blurred cropped image in the original image

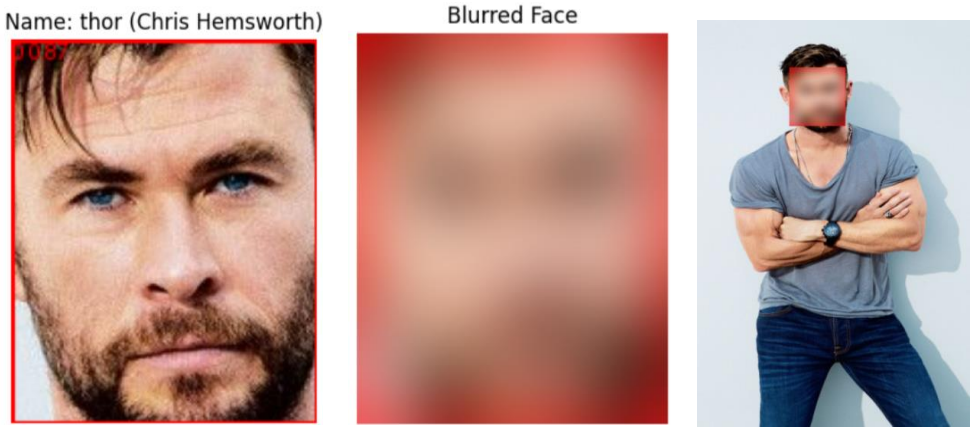


Figure 185 blurring cropped face then replacing in original image

The cropped image predicted class, blurred cropped image and original image with the blurred face that are displayed

```
[ ] # Recognize face from the blurred cropped image to compare
    predicted_class_whole = predict_image_class(blurred_face, face_recognition_model)

    if predicted_class_whole == 0:
      predicted_name_whole = 'bruce_banner (Mark Ruffalo)'
    elif predicted_class_whole == 1:
        predicted_name_whole = 'clint_barton (Jeremy Renner)'
    elif predicted_class_whole == 2:
        predicted_name_whole = 'Natasha_Romanoff (Scarlett Johansson)'
    elif predicted_class_whole == 3:
        predicted_name_whole = 'steve_rogers (Chris Evans)'
    elif predicted_class_whole == 4:
        predicted_name_whole = 'thor (Chris Hemsworth)'
    elif predicted_class_whole == 5:
        predicted_name_whole = 'tony_stark (Robert Downey Jr.)'
    else:
        predicted_name_whole = 'Unknown'
    # If the predicted name from the whole image is different from the cropped one, print 'Unknown'
    if predicted_name != predicted_name_whole:
        print('Name: Unknown')
    else:
        print(f"Name: {predicted_name_whole}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Name: Unknown
```

Predict (Recognize) the face in the cropped image after blurring and display the result (if the predicted class is not the same as the actual class, then print unknown for visualization)

As shown above, the face can no longer be recognized correctly after blurring.

8.6.2 Multiple Faces Per Image

```
[ ] # Paths
    model_path = '/content/drive/MyDrive/GP_YOLOV8Model/runs/detect/train/weights/best.pt'
    image_path = '/content/drive/MyDrive/GP_YOLOV5Model/hemsworth-evans.jpg'
```

Define the face detection model and the image path

```
[ ] # Load model
    model = load_modell(model_path)
```

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Load the face detection model and preprocess the image

```
results = predict(model, image)
```

```
0: 384x640 2 faces, 7.0ms
Speed: 2.1ms preprocess, 7.0ms inference, 1.3ms postprocess per image at shape (1, 3, 384, 640)
```

Predict the image (detect faces) using the face detection model

```
# Draw bounding boxes
result_image = draw_bounding_boxes(image, results)

# Display image
display_image(result_image)
```

Draw the bounding box and display the image with the bounding box

Figure 186 full image with drawn bboxes

Image with bbox drawn on it

```
[ ] CroppedImageMultiple=crop_bounding_boxes2(image, results)
```

```
CroppedImageMultiple
```

```
[<PIL.Image.Image image mode=RGB size=253x320>,
 <PIL.Image.Image image mode=RGB size=255x332>]
```

Crop the multiple bounding boxes that were detected in order to visualize

```
# Display the cropped images
display_cropped_images(CroppedImageMultiple)
```

Display the cropped images

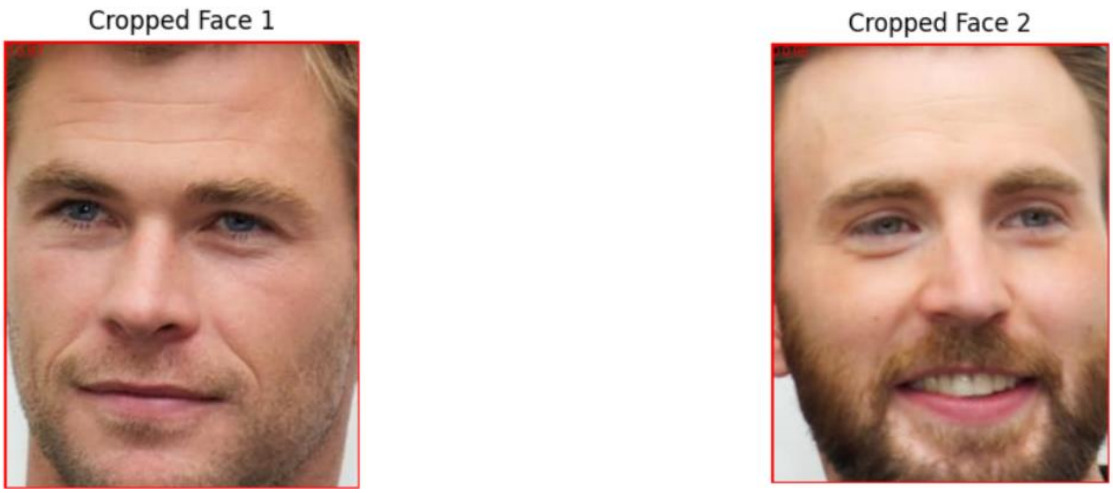


Figure 187 cropped faces image

The Displayed cropped faces

```
[ ] # Paths
    model = load_model('/content/drive/MyDrive/GP_YOLOV5Model/Boundary_Coordinates/MobileNet.h5')
    image_path = '/content/drive/MyDrive/GP_YOLOV5Model/hemsworth-evans.jpg'
```

load the face recognition model and define image path

```
[ ] # 'image' is the PIL Image object and 'results' is the output from the face detection model
    cropped_faces, face_boxes = crop_bounding_boxesR(image, results)

    # Process each face
    for idx, face in enumerate(cropped_faces):
        print(f"Face {idx + 1} bounding box: {face_boxes[idx]}")
        # Display or process each face image here
        face.show()
```

```
Face 1 bounding box: (465, 165, 718, 485)
Face 2 bounding box: (1118, 284, 1373, 616)
```

Crop the faces detected in the original image based on the bounding boxes provided by the face detection model. then process each cropped face by displaying it and printing its bounding box coordinates

```
predicted_classes = predict_image_classes(cropped_faces, model)
```

Predict the cropped faces using the face recognition model

```
[ ] predicted_classes
```

```
array([4, 3])
```

Display the classes that were predicted

```
for i, (face, bbox) in enumerate(zip(cropped_faces, face_boxes)):
    predicted_name = class_to_name(predicted_classes[i])
    print(f"Name: {predicted_name}")
    plt.imshow(np.array(face))
    plt.title(f"Name: {predicted_name}")
    plt.show()

    if predicted_name == 'thor (Chris Hemsworth)':
        blurred_face = blur_face(face)
        plt.imshow(np.array(blurred_face))
        plt.title("Blurred Face")
        plt.show()

        xmin, ymin, xmax, ymax = bbox
        image.paste(blurred_face, (xmin, ymin))

# display the final image
display_image(image)
```

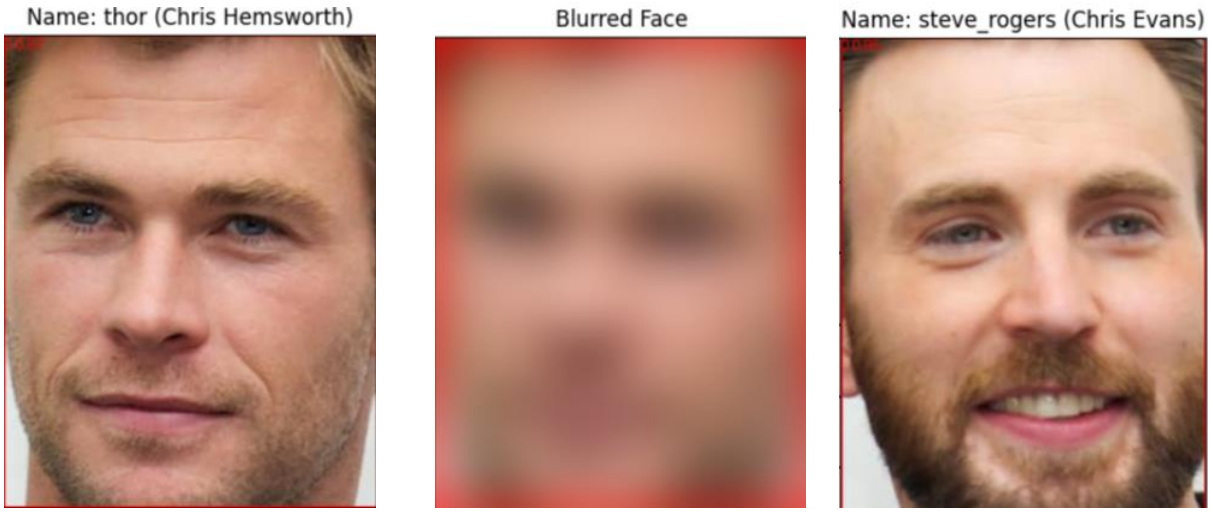


Figure 188 recognizing cropped faces and blur if thor

Display the predicted classes but as names for each cropped image, blur the cropped image if the predicted class is thor before displaying then replace the blurred image in the original image and display the final image

Figure 189 full image after blurring

Final image

```
# Recognize face from the cropped image but after blurring to compare
predicted_class_whole = predict_image_class(blurred_face, face_recognition_model)

if predicted_class_whole == 0:
  predicted_name_whole = 'bruce_banner (Mark Ruffalo)'
elif predicted_class_whole == 1:
    predicted_name_whole = 'clint_barton (Jeremy Renner)'
elif predicted_class_whole == 2:
    predicted_name_whole = 'Natasha_Romanoff (Scarlett Johansson)'
elif predicted_class_whole == 3:
    predicted_name_whole = 'steve_rogers (Chris Evans)'
elif predicted_class_whole == 4:
    predicted_name_whole = 'thor (Chris Hemsworth)'
elif predicted_class_whole == 5:
    predicted_name_whole = 'tony_stark (Robert Downey Jr.)'
else:
    predicted_name_whole = 'Unknown'
# If the predicted name from the whole image is different from the cropped one, print 'Unknown'
if predicted_name != predicted_name_whole:
    print('Name: Unknown')
else:
    print(f"Name: {predicted_name_whole}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Name: Unknown
```

Predict the blurred face using the face recognition model and print the result (if the predicted class after blurring is not the same as the actual class, then print unknown for visualization)

As shown above, the face can no longer be recognized correctly after blurring.

8.7 The YOLOV5 For Face Detection and MobileNet for Face Recognition Project Pipeline (Pipeline 4):

8.7.1 Results of an image with a Single Face:

```
[ ] from ultralytics import YOLO
    Dmodel = YOLO('/content/drive/MyDrive/GP_YOLOV5Model/best.pt')
```

Load the face detection model directly from ultralytics

```
# Paths
model_path = '/content/drive/MyDrive/GP_YOLOV5Model/best.pt'
image_path = '/content/drive/MyDrive/GP_YOLOV5Model/sarahnow.jpg'
```

Define the face detection model and image path

```
[ ] # Load model
    model = load_modell(model_path)
```

Load the face detection model

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Preprocess the image

```
] # Predict
  results = predict(model, image)
```

```
0: 640x384 1 face, 52.0ms
Speed: 2.5ms preprocess, 52.0ms inference, 1.5ms postprocess per image at shape (1, 3, 640, 384)
```

Predict the image using the face detection model and display the results

```
[ ] # Draw bounding boxes
    result_image = draw_bounding_boxes(image, results)

    # Display image
    display_image(result_image)
```

Draw the bounding box and display the image with the bounding box

Figure 190 image with drawn bbox

The displayed image with the bounding box drawn

```
[ ] CroppedImage=crop_bounding_box(image, results)
```

```
[ ] CroppedImage
```

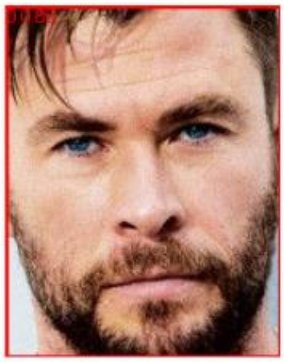


Figure 191 cropped face

Crop the face using the detected bounding box then display the cropped image

```
model_path = '/content/drive/MyDrive/GP_YOLOV5Model/Boundary_Coordinates/MobileNet.h5'
```

Define the face recognition path

```
[ ] model = load_model('/content/drive/MyDrive/GP_YOLOV5Model/Boundary_Coordinates/MobileNet.h5')
```

Load the face recognition model

```
[ ] # Crop bounding box (only one cropped image)
    CroppedImage, bbox = crop_bounding_box(image, results)
```

Crop the detected face using the results of the face detection model(bbox)

```python
# Recognize face using MobileNet and save result
if CroppedImage:
    predicted_class = predict_image_class(CroppedImage, face_recognition_model)

    # Mapping the class index to the name using if statements
    if predicted_class == 0:
        predicted_name = 'bruce_banner (Mark Ruffalo)'
    elif predicted_class == 1:
        predicted_name = 'clint_barton (Jeremy Renner)'
    elif predicted_class == 2:
        predicted_name = 'Natasha_Romanoff (Scarlett Johansson)'
    elif predicted_class == 3:
        predicted_name = 'steve_rogers (Chris Evans)'
    elif predicted_class == 4:
        predicted_name = 'thor (Chris Hemsworth)'
    elif predicted_class == 5:
        predicted_name = 'tony_stark (Robert Downey Jr.)'
    else:
        predicted_name = 'Unknown'

    resultsR = [(CroppedImage, predicted_name)]

    # Display recognized face and its name
    print(f"Name: {predicted_name}")
    plt.figure(figsize=(4, 4))
    plt.imshow(np.array(CroppedImage))
    plt.axis('off')
    plt.title(f"Name: {predicted_name}")
    plt.show()

        # Blur the face if it's Thor and display it before replacing
    if predicted_name == 'thor (Chris Hemsworth)':
        blurred_face = blur_face(CroppedImage)

        # Display the blurred cropped image
        plt.figure(figsize=(4, 4))
        plt.imshow(np.array(blurred_face))
        plt.axis('off')
        plt.title("Blurred Face")
        plt.show()

        xmin, ymin, xmax, ymax = bbox
        image.paste(blurred_face, (xmin, ymin))

    # Display the final image with the blurred face
    display_image(image)
```

Predict (Recognize) the cropped images using the face recognition model, then blur if thor and finally replace the blurred cropped image in the original image

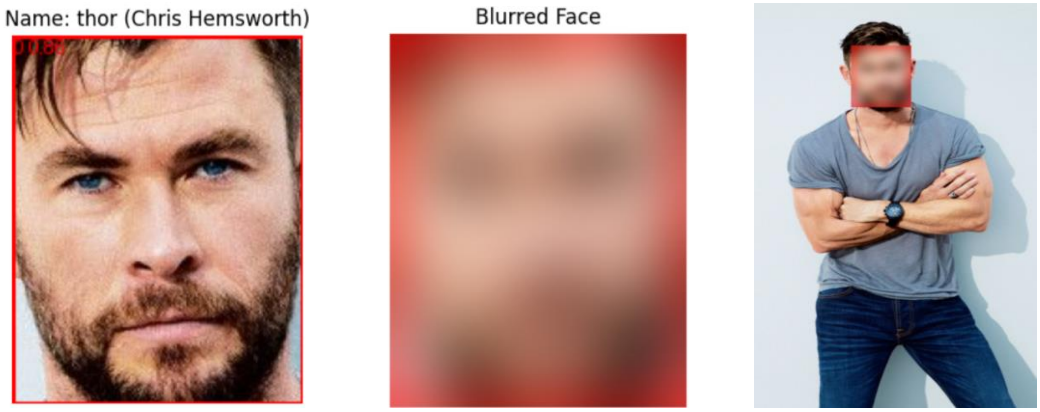


Figure 192 recognizing cropped face and if thor blur and replace in original image

The cropped image predicted class, blurred cropped image and original image with the blurred face that are displayed

```
# Recognize face from the whole blurred image to compare
predicted_class_whole = predict_image_class(image, face_recognition_model)

if predicted_class_whole == 0:
  predicted_name_whole = 'bruce_banner (Mark Ruffalo)'
elif predicted_class_whole == 1:
    predicted_name_whole = 'clint_barton (Jeremy Renner)'
elif predicted_class_whole == 2:
    predicted_name_whole = 'Natasha_Romanoff (Scarlett Johansson)'
elif predicted_class_whole == 3:
    predicted_name_whole = 'steve_rogers (Chris Evans)'
elif predicted_class_whole == 4:
    predicted_name_whole = 'thor (Chris Hemsworth)'
elif predicted_class_whole == 5:
    predicted_name_whole = 'tony_stark (Robert Downey Jr.)'
else:
    predicted_name_whole = 'Unknown'
# If the predicted name from the whole image is different from the cropped one, print 'Unknown'
if predicted_name != predicted_name_whole:
    print('Name: Unknown')
else:
    print(f"Name: {predicted_name_whole}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Name: Unknown
```

Predict (Recognize) the face in the cropped image after blurring and display the result (if the predicted class is not the same as the actual class, then print unknown for visualization)

As shown above, the face can no longer be recognized correctly after blurring.

8.7.2 Multiple Faces Per Image

```
[ ] # Paths
    model_path = '/content/drive/MyDrive/GP_YOLOV5Model/best.pt'
    image_path = '/content/drive/MyDrive/GP_YOLOV5Model/hemsworth-evans.jpg'
```

Define the face detection model and the image path

```
[ ] # Load model
    model = load_modell(model_path)
```

```
[ ] # Preprocess image
    image = preprocess_image(image_path)
```

Load the face detection model and preprocess the image

```
[ ] # Predict
    results = predict(model, image)
```

```
0: 384x640 2 faces, 13.0ms
Speed: 2.4ms preprocess, 13.0ms inference, 1.6ms postprocess per image at shape (1, 3, 384, 640)
```

Predict the image (detect faces) using the face detection model

```
# Draw bounding boxes
result_image = draw_bounding_boxes(image, results)

# Display image
display_image(result_image)
```

Draw the bounding box on the image and display it

Figure 193 full image with bboxes drawn

The Image with bounding boxes drawn

```
[ ] CroppedImageMultiple=crop_bounding_boxes2(image, results)
```

```
[ ] CroppedImageMultiple
```

```
[<PIL.Image.Image image mode=RGB size=258x327>,
 <PIL.Image.Image image mode=RGB size=253x344>]
```

Crop the multiple bounding boxes that were detected in order to visualize

```
# Display the cropped images
display_cropped_images(CroppedImageMultiple)
```

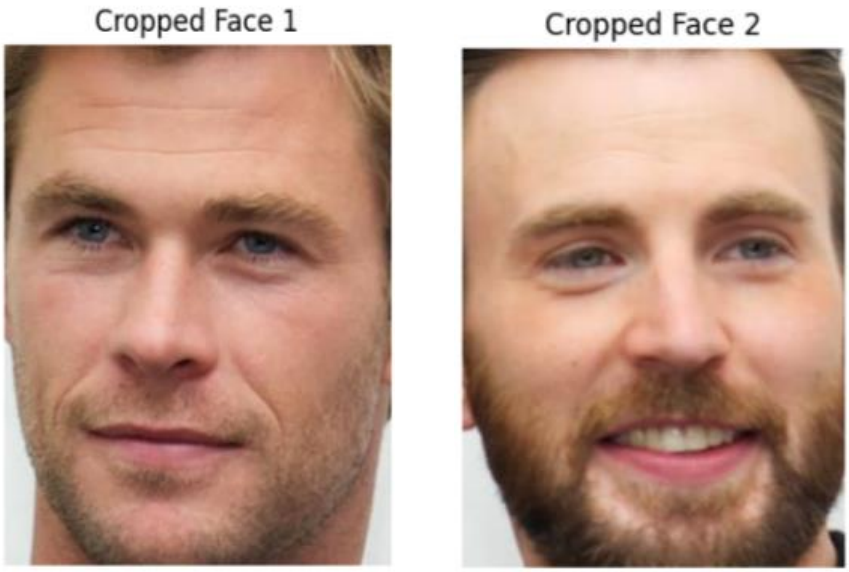


Figure 194 cropped faces

Display the cropped images

```
model = load_model('/content/drive/MyDrive/GP_YOLOV5Model/Boundary_Coordinates/MobileNet.h5')
```

load the face recognition model and define image path

```
cropped_faces, face_boxes = crop_bounding_boxesR(image, results)

# Process each face
for idx, face in enumerate(cropped_faces):
    print(f"Face {idx + 1} bounding box: {face_boxes[idx]}")
    # Display each face boundary box coordinates in the image here
    face.show()
```

```
Face 1 bounding box: (462, 170, 720, 497)
Face 2 bounding box: (1123, 283, 1376, 627)
```

Crop the faces detected in the original image based on the bounding boxes provided by the face detection model. then process each cropped face by displaying it and printing its bounding box coordinates

```
[ ] predicted_classes = predict_image_classes(cropped_faces, model)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```

```
[ ] predicted_classes
```

```
array([4, 3])
```

Predict the cropped faces using the face recognition model and display the predictions

```
for i, (face, bbox) in enumerate(zip(cropped_faces, face_boxes)):
    predicted_name = class_to_name(predicted_classes[i])
    print(f"Name: {predicted_name}")
    plt.imshow(np.array(face))
    plt.title(f"Name: {predicted_name}")
    plt.show()

    if predicted_name == 'thor (Chris Hemsworth)':
        blurred_face = blur_face(face)
        plt.imshow(np.array(blurred_face))
        plt.title("Blurred Face")
        plt.show()

        xmin, ymin, xmax, ymax = bbox
        image.paste(blurred_face, (xmin, ymin))

# displays the final image
display_image(image)
```

Figure 195 recognize cropped faces and if thor blur

Display the predicted classes but as names for each cropped image, blur the cropped image if the predicted class is thor before displaying then replace the blurred image in the original image and display the final image.

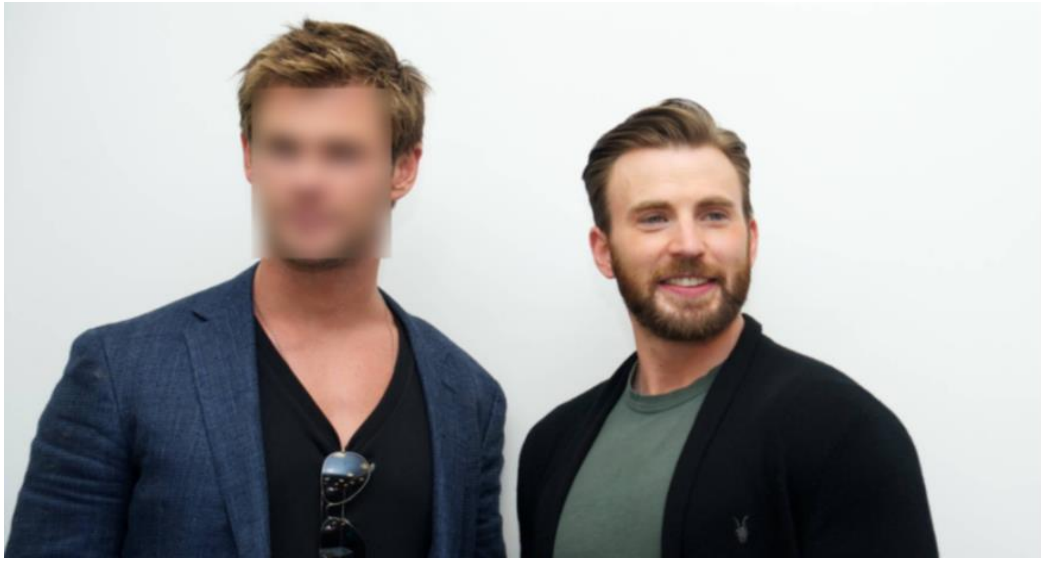


Figure 196 final image after blurring

Final image After Blurring Thor

```
# Recognize face from the cropped image but after blurring to compare
predicted_class_whole = predict_image_class(blurred_face, face_recognition_model)

if predicted_class_whole == 0:
  predicted_name_whole = 'bruce_banner (Mark Ruffalo)'
elif predicted_class_whole == 1:
    predicted_name_whole = 'clint_barton (Jeremy Renner)'
elif predicted_class_whole == 2:
    predicted_name_whole = 'Natasha_Romanoff (Scarlett Johansson)'
elif predicted_class_whole == 3:
    predicted_name_whole = 'steve_rogers (Chris Evans)'
elif predicted_class_whole == 4:
    predicted_name_whole = 'thor (Chris Hemsworth)'
elif predicted_class_whole == 5:
    predicted_name_whole = 'tony_stark (Robert Downey Jr.)'
else:
    predicted_name_whole = 'Unknown'
# If the predicted name from the whole image is different from the cropped one, print 'Unknown'
if predicted_name != predicted_name_whole:
    print('Name: Unknown')
else:
    print(f"Name: {predicted_name_whole}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
Name: Unknown
```

Predict the blurred face using the face recognition model and print the result (if the predicted class after blurring is not the same as the actual class, then print unknown for visualization)

As shown above, the face can no longer be recognized correctly after blurring.

9 Discussion

9.1 Face Detection Models Results:

Table 1 Face Detection Models Results

Model	Precision	Recall	mAP50	mAP50-95
YoloV5	0.982	0.965	0.991	0.849
Yolov8	0.983	0.961	0.99	0.839

Analysis of Face Detection Models Results:

- **Precision:**
  - **YOLOv5**: Achieved a precision of 0.982, indicating a high accuracy in correctly identifying positive detections with very few false positives.
  - **YOLOv8**: Slightly higher precision at 0.983, showcasing a marginal improvement in accuracy over YOLOv5.
- **Recall**:
  - **YOLOv5**: Demonstrated a recall of 0.965, which means it successfully identified most of the true positive cases with minimal missed detections.
  - **YOLOv8**: Slightly lower recall at 0.961 compared to YOLOv5, indicating it missed slightly more true positives.
- **mAP50 (Mean Average Precision at IoU threshold 0.50)**:
  - **YOLOv5**: Achieved an mAP50 of 0.991, indicating excellent performance in precision and recall across different IoU thresholds.
  - **YOLOv8**: Very close performance with an mAP50 of 0.99, suggesting similarly high effectiveness in detecting faces.
- **mAP50-95 (Mean Average Precision at varying IoU thresholds)**:
  - **YOLOv5**: Scored 0.849, showing strong performance across a range of stricter IoU thresholds.
  - **YOLOv8**: Slightly lower at 0.839, indicating a marginally lesser performance in handling more stringent detection requirements.

Summary:

Both YOLOv5 and YOLOv8 exhibit excellent performance in face detection tasks, with YOLOv8 showing a slight edge in precision but a minor decrease in recall and mAP50-95. YOLOv5 maintains

a more balanced performance across all metrics, making it slightly better in terms of overall detection quality.

9.2 Face Recognition Models Results:

Table 2 Face Recognition Models Results

Class	Model	Precision	Recall	F1-Score
Tony Stark (Robert Downey Jr.)	Yolov5	0.943	0.725	0.819
	Yolov8	0.889	0.944	0.915
	MobileNet	1.00	0.94	0.97
Steve Rogers (Chris Evans)	Yolov5	0.933	0.819	0.872
	Yolov8	0.864	0.78	0.787
	MobileNet	0.95	0.95	0.95
Natasha Romanoff (Scarlett Johansson)	Yolov5	0.774	0.865	0.817
	Yolov8	0.968	0.941	0.853
	MobileNet	0.95	1.00	0.98
Bruce Banner (Mark Ruffalo)	Yolov5	0.881	0.939	0.909
	Yolov8	0.76	0.831	0.701
	MobileNet	0.95	0.95	0.96
Clint Barton (Jeremy Renner)	Yolov5	0.927	0.965	0.945
	Yolov8	0.924	0.854	0.678
	MobileNet	0.95	0.97	0.96
Thor (Chris Hemsworth)	Yolov5	0.881	0.939	0.909
	Yolov8	0.96	0.948	0.787
	MobileNet	0.96	0.98	0.97

Analysis of Face Recognition Models Results:

- **Tony Stark (Robert Downey Jr.):**
- **YOLOv5**: High precision but lower recall indicates fewer false positives but some missed detections.
- **YOLOv8**: Better recall than YOLOv5, indicating more consistent detections.

- **MobileNet**: Perfect precision and very high recall, indicating excellent performance with few false positives.

- **Steve Rogers (Chris Evans):**

- **YOLOv5:** Balanced performance with higher precision than recall.
- **YOLOv8:** Lower recall indicates some missed detections.
- **MobileNet:** Balanced high performance, indicating robust detection and recognition.

- **Natasha Romanoff (Scarlett Johansson):**

- **YOLOv5:** Lower precision, indicating more false positives.
- **YOLOv8:** High precision and recall, indicating fewer false positives and consistent detections.
- **MobileNet:** Perfect recall and high precision, indicating excellent performance.

- **Bruce Banner (Mark Ruffalo):**

- **YOLOv5**: High recall, indicating consistent detections.
- **YOLOv8:** Lower precision and recall compared to YOLOv5 and MobileNet.
- **MobileNet:** Balanced high performance with high precision and recall.

- **Clint Barton (Jeremy Renner):**

- **YOLOv5:** High precision and recall, indicating strong performance.
- **YOLOv8:** Lower recall, indicating more missed detections.
- **MobileNet:** High precision and recall, indicating robust performance.

- **Thor (Chris Hemsworth):**

- **YOLOv5:** Balanced high performance with high precision and recall.
- **YOLOv8:** High precision but lower recall compared to YOLOv5 and MobileNet.
- **MobileNet:** Excellent performance with high precision and recall.

Summary

- **Best Overall Model:** MobileNet, due to its high and balanced precision, recall, and F1-scores across all classes.
- **YOLOv5:** Good performance, especially in precision, but generally lower recall compared to MobileNet.

- **YOLOv8**: Mixed performance with some strengths in precision but lower recall and F1-scores compared to both MobileNet and YOLOv5.

9.3 Blurring Techniques Results:

Table 3 Blurring Techniques Results

Blurring Tech / Class	Tony Correct	Tony Incorrect	Thor Correct	Thor Incorrect
Before any Blurring	289	6	184	2
Radial Blur	10	285	29	157
Median Blur	41	254	46	140
Motion Blur	257	38	169	17
GrabCut (Gaussian)	100	195	51	135
Mean shift Filtering	209	86	91	95
Box Filter	48	247	80	106
Gaussian Blur	1	294	7	179
Motion & Mean shift Filtering	159	136	66	120
Radial & Gaussian Blur	0	295	4	182
Motion, Mean Shift, Radial & Gaussian	0	295	4	182
Motion, Mean Shift, Radial & Gaussian (5 iterations each)	0	295	0	186

Summary:

- **Most Effective Blurs for Anonymization:**
- **Gaussian Blur**: Very high incorrect recognitions for both classes.
- **Radial & Gaussian Blur**: Extremely high incorrect recognitions, indicating effective anonymization.
- **Motion, Mean Shift, Radial & Gaussian (5 iterations each):** No correct recognitions, indicating the highest level of anonymization.
- **Least Effective Blurs:**
- **Motion Blur**: Maintains high correct recognitions, indicating low anonymization.

10 Challenges and Limitations

One of the main challenges I faced was finding a dataset suitable for my entire pipeline. Most datasets were either suitable for face detection with bounding box coordinates or for face recognition with labeled images, but not both. Additionally, the face recognition datasets I found were highly imbalanced, with some classes having as few as 15 images, which was insufficient for training. To address this, I used the LFW dataset for face detection due to its adequate size and the Avengers dataset for face recognition, which contains six classes. I supplemented the Avengers dataset with additional images from celebrity datasets to increase the number of images per class, but these did not include bounding box coordinates. To overcome this, I retrained a pre-trained YOLO model for face detection on the LFW dataset, then used this model to predict and save the bounding box coordinates for the Avengers dataset.

Another limitation of the Avengers dataset was that all images were tightly cropped to the face, meaning the model could only recognize faces from very close, cropped images. To test the model on external images and ensure it worked in real-life scenarios, I had to first use the face detection model to identify the bounding box coordinates of faces in the image. I then cropped the faces using these coordinates, preprocessed the cropped faces, and passed them to the face recognition model. If the recognized face was identified as Thor, I replaced the blurred cropped face back into the original image. This multi-step process was necessary to adapt the model to recognize faces from images where the faces were not pre-cropped.

11 Trials

During the project, different models and approaches were tested to attain effective recognition and detection of faces. However, not all models produced satisfactory results—some are described below:

MTCNN (Multi-task Cascaded Convolution:

I tried MTCNN due to its reputation in face detection and alignment. While this was promising from the outset, MTCNN demonstrated some significant shortcomings as it attempted to handle the subtleties of the Avengers dataset. Because heroes' faces are so heavily cropped, bounding boxes will generally be noisy, particularly with less discriminative features. An imbalance in data, however, caused this inconsistency in detection across different classes.

CNN (Convolutional Neural Networks):

We consider this traditional CNN for face detection and recognition due to its power in image classification. But the problem of the small and imbalanced dataset appeared; the CNNs were overfitting and thus not remarkable at generalization. Preprocessing was extensive with bounding box handling, and CNNs were not well adapted to variations in distances and angles.

YOLOv7:

More exploration was done on YOLOv7, which is fast and very accurate in detecting face objects. Unfortunately, with the highly preempted images in the Avengers dataset, it kept running into some problems. Misclassification between faces and background elements was the order of the day, and the problem concerning class imbalance hindered the model from reaching high precision and recall. This does not help it generalize well, even with the ramping up of hard negatives from extra augmentation in attempts to remove backgrounds.

Haar Cascades:

I tried some initial efforts with Haar Cascades, as they are easy to use and can be implemented in real-time. By all means, they failed, breaking down at complex backgrounds or changing illumination settings. This led to many false positives and missed detections by the cropping and imbalanced dataset in the detection system.

Facenet:

Facenet was one of the methods used for creating embeddings for classification. Even though Facenet was efficient for face recognition, it required a balanced dataset with a substantial number of instances to work well. The main limitations of the Avengers dataset lay in the small number of images per class and the lack of bounding box coordinates, making Facenet less powerful. It also failed in the identification of close look-alike faces, thereby yielding low precision and recall rate.

Data Augmentation:

Different data augmentation techniques were tested for the extension of the training set and counterbalancing the small size: rotation, flipping, and zooming out to solve the cropped faces issue. Still, the models did not generalize well with data augmentation. Augmented images were sometimes noisy, and the models still were not performing very well on highly cropped or less distinctive faces. More so, these trials brought out the limitations of the vast dataset: the small size, imbalance, and absence of bounding box coordinates in the Avengers dataset had been a significant setback to the

whole process. From these experiences, it was clear there was a need for much more robust and diverse datasets and the choice of models that could handle these peculiarities. Consequently, YOLOv8 was chosen since it strikes such a good balance in speed and accuracy, addressing face detection and recognition despite the challenges presented by the dataset.

12 Conclusions and Future Work

12.1 Summary

Using this graduation project, I have come a long way in developing an AI-based system for face detection from images, recognizing the faces and blurring them to increase anonymity. I thus started running experiments with the YOLOv5 and YOLOv8 models for face detection on top of the YOLOv5, YOLOv8, and MobileNet models for face recognition. In addition, I conducted some more experiments with each single blurring technique: Gaussian blur, Median blur, and Motion blur. Additionally, I applied these techniques in pairs and quartets, iterating each five times to achieve optimal face obscurity.

However, datasets had evident challenges; I used the LFW and Avenger datasets. The LFW dataset was for face detection, while the Avengers dataset with some celebrity datasets to augment was used for face recognition. In the Avengers dataset, bounding box coordinates were not present; thus, YOLO models lacked face detection. That the LFW dataset was retrained on to predict and save these coordinates. This work was somewhat limited because the Avenger dataset had only cropped images of faces, shortening the recognition of distant faces by the model.

The performance of the models was varying, and precision was 0.982, recall 0.965, and mAP50 and mAP50-95 were 0.991 and 0.849 in detecting faces by YOLOv5. On the other hand, the older version of YOLO, YOLOv8, obtained similar performance with a precision of 0.983, recall of 0.961, mAP50 0.99, and mAP50-95 0.839. The character-wise recognition by these models showed mixed results, and MobileNet often performs well.

In future work, I will aim to collect a much larger and more diverse dataset, including a wide variety of classes. This increased dataset will increase the performance of face detection and recognition and will be sensitive for object findings that may breach an individual's privacy: ATM details, plate numbers, street numbers, name tags, sensitive private information, etc. This will make the model even more robust for private protection under many other cases while at the same time boosting diversity and volume, which will enhance precision for a broader application of the model to situations. The improvement in dataset diversity and volume will ensure that the model is broad in the variety of objects it can recognize, thereby making it versatile and practical under real-world application circumstances, leading to comprehensive and reliable protection of privacy for deserving individuals.

Conclusion: This project has clearly illustrated how AI can enhance privacy because of automated face blurs. By using deep learning techniques and resolving some issues found in previous datasets, the present research sets a foundation for a safer and more secure online environment.

12.2 Future Work

For future work, the collection will be of a large and very diverse dataset covering a wide range of classes that will take face detection and recognition to the next level, where the model can incorporate sensitive objects. Whose detection could breach privacy concerning individuals? For example, the model can be trained to identify ATM details, plate numbers, street numbers, and name tags, among others. This will increase model capability to protect privacy in diverse scenarios. In addition, diversity and volume of the dataset will enhance the model's robustness and accuracy at hand under different contexts. Broadening the objects that can be detected increases the versatility and effectiveness of the model in real life, and thus comprehensive and reliable privacy protection for individuals' privacy is given to them in real life.

In addition, it is equally important to state the limitations faced in this project work. One very vital challenge came when finding one single dataset that could support functionalities such as face detection (i.e., bounding box coordinates) and capabilities for face recognition (i.e., labeled data). The face recognition datasets were often imbalanced classes, with fewer images for every class. Thus, to find a solution, I used the LFW dataset to detect faces and further complemented the same information in the Avengers dataset with more images from celebrity datasets; these datasets do not contain bounding boxes. After that, I re-trained the YOLO model for face detection for the LFW dataset and used the same to predict and save bounding box coordinates for the Avengers dataset. However, the approach was itself limited as the dataset of Avengers contained only cropped face images, which reduced the capability to recognize faces at a distance.
These issues will be resolved in the future by balancing the count of the number of images on a per-class basis, as it will ensure that face detection data and recognition data have sufficient annotations to deal with the requirements. This would mean the collection of varied and good-quality images to ensure the representation of classes well. All these limitations could be tackled by expanding the dataset to prepare the model for diverse real-life scenarios better. This betterment will ensure accuracy and reliability, ultimately improving the safeguard of individual privacy—something significant in our digital world.

References

[1] S. S. Mohamed, W. A. Mohamed, A. T. Khalil, and A. S. Mohra, "Deep Learning Face Detection and Recognition," *Intl Journal of Electronics and Telecommunications*, pp. 1-8, 2024.

[2] D. Garg, P. Goel, S. Pandya, A. Ganatra, and K. Kotecha, "A Deep Learning Approach for Face Detection using YOLO," *2018 IEEE Punecon*, Pune, India, 2018, pp. 1-8. DOI: 10.1109/PUNECON.2018.8745376.

[3] Y. Sun, X. Wang, and X. Tang, "Deep Learning Face Representation from Predicting 10,000 Classes," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 1891-1898. DOI: 10.1109/CVPR.2014.244.

[4] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, Columbus, OH, USA, 2014, pp. 1701-1708. doi: 10.1109/CVPR.2014.220.

[5] R. A. Vyas, "Feature Extraction Technique of PCA for Face Recognition With Accuracy Enhancement," *International Journal on Recent and Innovation Trends in Computing and Communication (IJRITCC)*, vol. 4, no. 11, pp. 288-291, Nov. 2016.

[6] M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition," presented at the British Machine Vision Conference (BMVC), Swansea, UK, September 2015.

[7] Y. Nakashima, T. Koyama, N. Yokoya, and N. Babaguchi, "Facial Expression Preserving Privacy Protection Using Image Melding," presented at the IEEE International Conference on Image Processing (ICIP), 2015.

[8] T. Li and L. Lin, "AnonymousNet: Natural Face De-Identification with Measurable Privacy," presented at the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

[9] Hashmi, S.A. (2022) Face detection in extreme conditions: A machine-learning approach, arXiv.org. Available at: https://arxiv.org/abs/2201.06220.

[10] Face recognition and identification using deep learning approach. Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1755/1/012006.

[11] Face recognition systems use either offline data or real-time input (2020) Design and evaluation of a real-time face recognition system using convolutional neural networks, Procedia Computer Science. Available at: https://www.sciencedirect.com/science/article/pii/S1877050920311583

[12] D. V. Thosar, R. Narahare, A. Kapse, M. Sananse, and H. Khan, "Face Recognition with Machine Learning," *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, vol. 2, no. 2, pp. 435-442, May 2022. DOI: 10.48175/IJARSCT-3681.

[13] B. S. Bindushree and R. A. N. Rakshitha, "Face Recognition Using Deep Learning," *International Journal of Advanced Scientific Innovation*, vol. 01, no. 01, pp. 1-8, Nov. 2020.

[14] K. H. Rahouma and A. Z. Mahfouz, "Design and Implementation of a Face Recognition System Based on API Mobile Vision and Normalized Features of Still Images," *Procedia Computer Science*, vol. 194, pp. 32-44, 2021, doi: 10.1016/j.procs.2021.10.057.

[15] Zhu C, Huang S, Evans R, Zhang W. Cyberbullying Among Adolescents and Children: A Comprehensive Review of the Global Situation, Risk Factors, and Preventive Measures. Front Public Health. 2021 Mar 11;9:634909. doi: 10.3389/fpubh.2021.634909. PMID: 33791270; PMCID: PMC8006937.

[16] "Reputational Harm," Founder Shield. [Online]. Available: https://foundershield.com/insurance-terms/definition/reputational-harm/.

[17] "Identity Theft," Investopedia. [Online]. Available: https://www.investopedia.com/terms/i/identitytheft.asp.

[18] "GrabCut Foreground Extraction," OpenCV Documentation. [Online]. Available: https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html.

[19] "Gaussian Blurring," Data Carpentry. [Online]. Available: https://datacarpentry.org/image-processing/06-blurring.html.

[20] "Median Blur Filter," GIMP Documentation. [Online]. Available: https://docs.gimp.org/2.10/en/gimp-filter-median-blur.html.

[21] "Different Approaches to Blurring Digital Images and Their Effect on Facial Detection," ScholarWorks at the University of Arkansas. [Online]. Available: https://scholarworks.uark.edu/cgi/viewcontent.cgi?article=1067&context=csceuht.

[22] "How to Find Clusters in Image Using Mean Shift in Python OpenCV," Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/62575894/how-to-find-clusters-in-image-using-mean-shift-in-python-opencv.

[23] "Make Image More Blurry When Farther From Center," Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/75961394/make-image-more-blurry-when-farther-from-center.

[24] "Implement Radial Blur with OpenCV," Stack Overflow. [Online]. Available: https://stackoverflow.com/questions/7607464/implement-radial-blur-with-opencv.

[25] "OpenCV Motion Blur in Python," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/opencv-motion-blur-in-python/.