VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

Final Report

Face Detection

Course: CSC14006 - Pattern Recognition

Reported by:
22127230 - Duong Binh Nguyen Lan
22127320 - Bui Ta Phat
22127476 - Dang Trieu Kha
22127486 - Dao Ngoc Thien

Teacher:
Le Hoang Thai
Duong Thai Bao
Truong Tan Khoa

Ho Chi Minh city, 12th May 2025

Contents

I. Group members 3

II. Introduction 3

III. Related works 3
1. AdaBoost-based method . . . 3
2. Faster R-CNN . . . 3
3. YOLO . . . 4
   - 3.1 YOLOv1 [4] . . . 4
   - 3.2 YOLOv2 (YOLO9000) [5] . . . 5
   - 3.3 YOLOv3 [6] . . . 7
   - 3.4 YOLOv4 . . . 8
   - 3.5 YOLOv5 [8, 3] . . . 8
4. An overview of state-of-the-art method . . . 11

IV. State-of-the-art approach 12
1. Backbone . . . 12
   - 1.1 Architecture . . . 12
   - 1.2 Scale-Aware RFE Model . . . 13
2. Neck . . . 14
   - 2.1 Architecture . . . 14
3. SEAM: Occlusion-Aware Attention Network . . . 15
   - 3.1 Structure of SEAM . . . 15
4. Head . . . 16
5. Loss functions . . . 16
   - 5.1 Intersection over Union (IoU) loss . . . 17
   - 5.2 Normalized Wasserstein Distance (NWD) loss . . . 17
   - 5.3 Repulsion loss . . . 17
6. Evaluation metric - mean Average Precision (mAP) . . . 18
7. Pros and Cons of YOLO-FaceV2 . . . 19

V. Experiments in the paper 20

VI. Our experiment 21
1. Qualitative Results on Sample Images . . . 21
2. Dataset selection and rationale . . . 22
   - 2.1 mAP calculation . . . 22
3. Hardware used . . . 22
4. Observed weaknesses during experimentation . . . 22

VII. Conclusions 23

VIII. Demonstration 23
1. Technologies used . . . 23
   - 1.1 Web . . . 23
   - 1.2 Model . . . 23
2. User interface . . . 23
   - 2.1 Home . . . 23
   - 2.2 Demo . . . 24
   - 2.3 About . . . 24
3. How to conduct . . . 24
   - 3.1 Project setup overview . . . 24
   - 3.2 Running the backend (API server) . . . 25
   - 3.3 Running the frontend (Web interface) . . . 25
   - 3.4 Using the web application . . . 25
   - 3.5 Demonstration scenarios . . . 25
   - 3.6 Technical notes . . . 25
   - 3.7 Additional considerations . . . 25
4. Link to demo video . . . 25

IX. References **25**

I. Group members

Student ID	Student name
22127230	Dương Bình Nguyễn Lân
22127320	Bùi Tá Phát
22127476	Đặng Triệu Kha
22127486	Đào Ngọc Thiện

II. Introduction

Face detection is a fundamental task in computer vision, aiming to identify the location and boundaries of faces in images or videos. Given an image or a video frame as input, the output consists of bounding boxes containing the coordinates and dimensions of the detected faces. This task serves as the foundation for various applications such as identity recognition, emotion analysis, and biometric security.

As the first step in an automatic face recognition pipeline, the reliability of the face detection stage plays a pivotal role, directly affecting the performance and applicability of the entire system. In modern approaches, face detection is typically formulated as a binary classification problem, in which sub-windows scanned from the input image are classified into two categories: face and non-face.

Before 2001, several methods were proposed based on anthropometric measurements, such as the size and distance between the nose, eyes, and mouth, to locate faces. However, these approaches yielded low recognition performance, leading to slow progress in the field. Other methods used dimensionality reduction techniques, such as principal component analysis (PCA) combined with classifiers such as Support Vector Machines (SVM). Although they improved accuracy to some extent, these methods still suffered from slow processing speeds and limited applicability in real-time scenarios.

III. Related works

1. AdaBoost-based method

In machine learning, the AdaBoost algorithm [2] constructs a strong classifier by linearly combining multiple weak classifiers. Its primary goal is to select the most effective weak classifiers, assigning them optimal weights to improve overall performance. However, AdaBoost itself does not directly learn the weak classifiers; instead, it assumes the existence of a base learning algorithm capable of training these classifiers based on the current weight distribution of the training data.

To enhance the efficiency of face detection, Viola and Jones integrated AdaBoost into a specialized structure known as a cascade of classifiers [2]. This structure consists of a sequence of strong classifiers arranged in increasing order of complexity. Each stage in the cascade is a strong classifier built from a small set of weak classifiers. If an image window fails to pass any stage, meaning it is classified as a non-face, it is immediately discarded, significantly reducing computation time by quickly eliminating irrelevant regions.

The combination of the AdaBoost algorithm and the cascade structure resulted in a face detection method that is both fast and accurate, outperforming earlier techniques in terms of both speed and effectiveness. This method marked a breakthrough in the field and paved the way for practical applications in facial recognition, including surveillance systems, human–machine interfaces, and mobile devices.

2. Faster R-CNN

Traditional face detection methods, like **AdaBoost**, often depended on low-level handcrafted features to locate faces in images. These features, however, struggle to handle the diverse variations in facial appearance seen in uncontrolled settings, such as varying lighting, poses, or occlusions. Enter **Faster R-CNN** [7], a powerful approach built on convolutional neural networks (CNNs). Unlike its predecessors, **Faster R-CNN** learns rich, adaptable visual features directly from raw images, enabling it to more effectively distinguish faces from complex backgrounds and perform reliably in challenging, real-world environments.

Faster R-CNN operates as a two-stage object detection framework. It starts with a **Region Proposal Network (RPN)**, which scans feature maps from a shared CNN backbone (e.g., ResNet or VGG) to propose potential bounding boxes. These proposals are then refined and classified in the second stage by a Fast R-CNN detector, which extracts region-specific features and finalizes the detection process. This structured approach ensures high accuracy in face detection by combining precise region proposals with robust feature learning.

However, the complexity of Faster R-CNN's two-stage architecture comes at a cost. The separate RPN and detection stages demand significant computational resources, limiting its suitability for real-time applications. To overcome these challenges, single-stage models like **YOLO** have emerged as compelling alternatives. By integrating feature extraction, bounding box prediction, and classification into a single pass.

3. YOLO

YOLO (You Only Look Once) is one of the major families of real-time object detection models, representing a novel approach in the field of computer vision. Unlike traditional two-stage detectors - such as R-CNN - which involve a region proposal stage followed by classification and refinement, YOLO models adopt a single neural network architecture that performs the entire process in one forward pass: from feature extraction, bounding box prediction, to object classification. The input image is divided into a grid, and each cell simultaneously predicts object locations, dimensions, and class probabilities. This results in a simple yet effective pipeline, resembling a panoramic observer recognizing everything in its field of view in a single instant.
YOLO has evolved through multiple versions, each improving upon its predecessor by optimizing speed, accuracy, and deployment capabilities. This progression is evident in the adoption of deeper backbone networks (such as Darknet and CSPDarknet), advanced feature extraction techniques (like FPN and PANet), and enhanced loss functions (such as CIoU and GHM). The core advantages of the YOLO family lie in its exceptional processing speed (ranging from dozens to hundreds of frames per second on GPU), lightweight and deployable architecture, and competitive performance on standard datasets like COCO - making it a top choice for real-time applications such as video surveillance, autonomous driving, and robotics.

3.1 YOLOv1 [4]

YOLOv1 (You Only Look Once), introduced by Joseph Redmon and colleagues in the paper "You Only Look Once: Unified, Real-Time Object Detection" presented at CVPR 2016, marked a breakthrough in real-time object detection. Unlike two-stage approaches such as R-CNN, Fast R-CNN, and Faster R-CNN, YOLOv1 integrated object localization and classification within a single neural network, framing the task as a regression problem.

3.1.1 Idea

The YOLOv1 (You Only Look Once) model operates by dividing the input image into an evenly spaced grid of size $S \times S$, where each grid cell is responsible for detecting objects whose centers fall within that cell. Each cell predicts B bounding boxes, and each box is described by five parameters: the center coordinates (x, y) (relative to the grid cell), the width and height (w, h) (relative to the entire image), and a confidence score, which represents the product of the probability that an object exists in the box and the accuracy of the box (typically measured using IoU – Intersection over Union). Additionally, each cell predicts a class probability distribution over C object categories, which is shared across all boxes within the cell.
The final output of the model is a tensor with shape $S \times S \times (B \times 5 + C)$. During inference, boxes with a confidence score below a predefined threshold are filtered out to reduce noise. Then, the Non-Maximum Suppression (NMS) algorithm is applied to eliminate redundant boxes (those with high IoU and the same class prediction), retaining only the one with the highest confidence score to ensure clear, non-overlapping, and accurate detections.

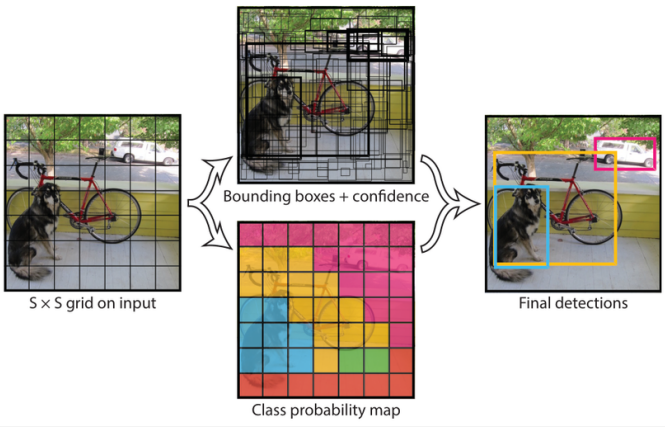

Figure 1: The YOLO's idea [4]

3.1.2 Detailed architecture

YOLOv1 uses a simple convolutional neural network (CNN) for real-time object detection. The image processing pipeline consists of:

1. Input: The image is resized to a fixed size of 448×448.
2. Feature extraction: A series of convolutional layers process the image, interleaved with four max-pooling layers that reduce the spatial dimensions from 448×448 to 7×7, generating high-level features.
3. Prediction: Two fully connected layers convert the features into a $7 \times 7 \times 30$ tensor, predicting 2 bounding boxes (coordinates, size, confidence) and class probabilities for 20 categories for each grid cell.

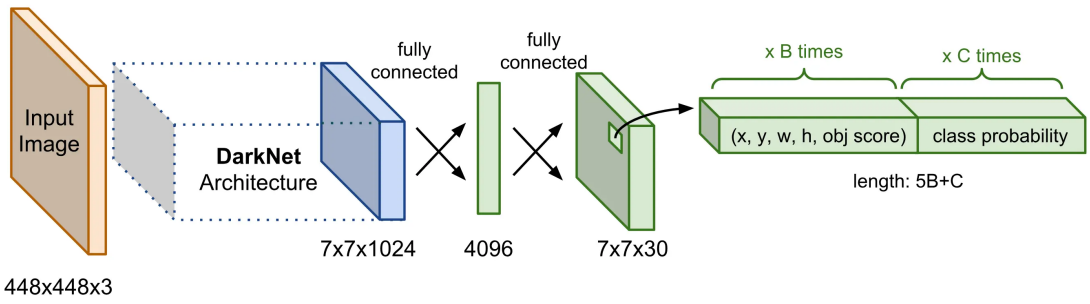

Figure 2: YOLOv1 architecture [1]

In addition, YOLOv1 employs a multi-part loss function, designed to handle three key tasks within a single model:

1. Localization Loss – measures the accuracy of the predicted bounding box coordinates and dimensions.
2. Confidence Loss – evaluates the confidence of the model regarding the presence of an object.
3. Classification Loss – quantifies the error in assigning the correct class label to the detected object.

The entire loss function is expressed as:

$$
\begin{aligned}
\mathcal{L} = \lambda_{\text{coord}} & \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
& + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
& + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
& + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbf{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
& + \sum_{i=0}^{S^2} \mathbf{1}_{i}^{\text{obj}} \sum_{c=1}^{C} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

3.2 YOLOv2 (YOLO9000) [5]

As an improved version of YOLOv1, YOLOv2 was introduced by Joseph Redmon and Ali Farhadi in 2017 through the paper "YOLO9000: Better, Faster, Stronger", released just one year after YOLOv1, bringing several significant enhancements. This model addressed YOLOv1's limitations in localization errors and multi-object detection capabilities, while also introducing techniques that later became foundational for newer versions such as YOLOv3, YOLOv4, and YOLOv5.

Notable improvements over YOLOv1: YOLOv2 introduced core changes that were preserved and further developed in subsequent versions.

3.2.1 Addressing YOLOv1's limitation in detecting multiple objects

In YOLOv1, each grid cell (7×7) predicts up to two bounding boxes. However, both boxes share a single set of class probabilities, meaning each cell can only effectively detect one object. This leads to missed detections when multiple objects from different classes appear within the same grid cell, especially in scenes with high object density or overlapping instances.

3.2.2 Anchor boxes: Easier prediction and improved object recall

YOLOv2 introduces anchor boxes for bounding box prediction. Instead of directly predicting absolute bounding box coordinates as in YOLOv1, the model predicts offsets relative to a set of predefined anchor boxes. Learning offsets from typical bounding box shapes simplifies the regression task and stabilizes training.
These anchor boxes are designed using k-means clustering on the training dataset to accurately reflect the common shapes and aspect ratios of objects, which helps reduce localization errors.

3.2.3 Direct location prediction

YOLOv1 did not enforce any constraint on the location of predicted boxes, leading to instability during early training when boxes could be predicted far from their corresponding grid cells.
YOLOv2 addresses this by applying a sigmoid function to the predicted coordinates, restricting them to the range $(0, 1)$. This forces the model to predict relative offsets around the current grid cell, resulting in more stable and faster convergence during training.
Specifically, given an anchor box of size (p_w, p_h) at a grid cell with top-left corner coordinates (c_x, c_y), the model predicts 4 values:

- t_x, t_y: Relative offsets within the cell
- t_w, t_h: Scale factors for the anchor box
- t_o: Objectness confidence

From these, the predicted bounding box coordinates are computed as follows:

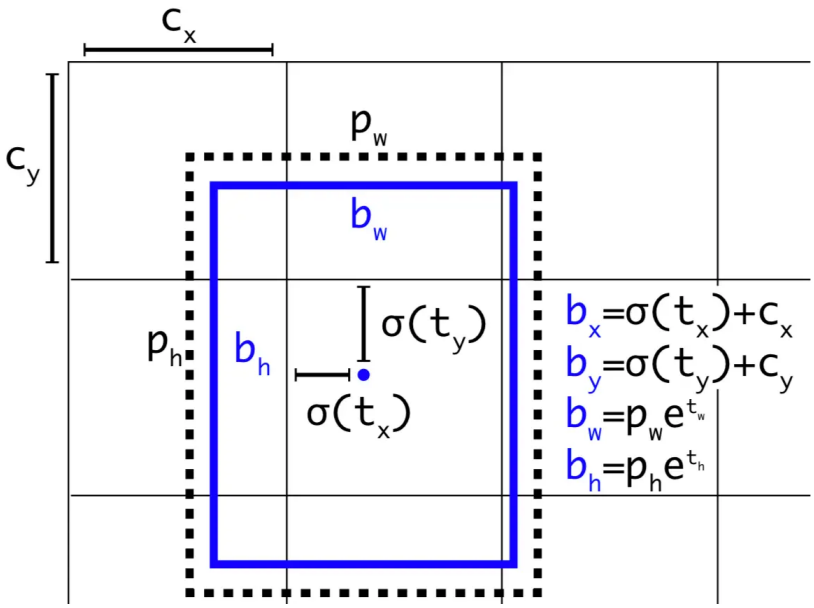

Figure 3: Prediction on YOLOv2 [5]

This formulation enables the model to effectively predict accurate bounding boxes without needing to search across the entire image.

3.2.4 Fully convolutional architecture

YOLOv1 used fully connected layers at the end of the network for prediction, which required a fixed input size (448×448) and resulted in a large number of parameters, making it less efficient for real-world deployment. YOLOv2 removes the fully connected layers, adopting a fully convolutional architecture where the entire network consists only of convolutional and pooling layers.
This design brings three main benefits: reduced parameter count, faster inference, and support for flexible input resolutions (from 320×320 to 608×608). As a result, YOLOv2 adapts better to real-world applications such as surveillance or autonomous driving, where image resolution often varies.

3.2.5 Batch normalization

YOLOv1 did not use batch normalization, making the training process more sensitive to weight initialization and prone to instability. In YOLOv2, batch normalization is applied after each convolutional layer, which normalizes outputs across the batch and keeps data distributions stable during training. This leads to faster convergence, reduced overfitting, and improved generalization.

3.3 YOLOv3 [6]

YOLOv3, introduced by Joseph Redmon and Ali Farhadi in 2018 through the paper "YOLOv3: An Incremental Improvement", was released one year after YOLOv2. This model enhanced object detection performance, especially for small objects and complex datasets, while maintaining real-time speed. YOLOv3 introduced a three-part architecture—backbone, neck, and head—and several core improvements that laid the foundation for later YOLO versions, although it still faced challenges in detecting very small objects in densely populated scenes.

3.3.1 Core improvements

YOLOv3 introduced four key enhancements that had a long-lasting impact and were preserved or further developed in later YOLO versions.

a. Use of Residual connections

In YOLOv2, the backbone network Darknet-19 had 19 convolutional layers but lacked mechanisms to combat the vanishing gradient problem, limiting its ability to learn complex features.
YOLOv3 replaced Darknet-19 with Darknet-53, a deeper network with 53 convolutional layers and integrated residual connections inspired by ResNet. These skip connections allow gradients to propagate more effectively during backpropagation, enabling the model to learn deeper features without performance degradation.
This technique significantly improved YOLOv3's ability to detect objects in complex scenes such as traffic surveillance or action recognition. Residual connections have since become standard practice and were further evolved in backbones like CSPDarknet (YOLOv4, YOLOv5) and ConvNeXt (YOLOv8).

b. Multi-scale prediction

In YOLOv2, predictions were made only on a single 13×13 grid, which limited performance on small object detection due to the absence of higher-resolution feature maps.
YOLOv3 addressed this by implementing a Feature Pyramid Network (FPN) architecture, making predictions at three different spatial resolutions:

- 52×52 (high resolution, for small objects)
- 26×26 (medium resolution, for medium-sized objects)
- 13×13 (low resolution, for large objects)

This multi-scale prediction strategy allowed YOLOv3 to effectively detect objects of varying sizes within the same image. It became a standard practice and was extended in later versions using structures like PANet and BiFPN.

c. Skip-layer concatenation: Combining features from different layers for improved accuracy

YOLOv3 introduced skip-layer concatenation, a method of merging features from multiple layers to retain spatial detail while leveraging deep contextual information. Specifically, during multi-scale prediction, YOLOv3 performs upsampling on deep feature maps (e.g., from 13×13) and concatenates them with shallower feature maps (e.g., from

26×26 or 52×52). This produces a rich feature tensor that contains both fine details and high-level context.
Unlike YOLOv2, which relied on a single feature map for predictions, this strategy significantly improves the detection of small objects, which are often diluted in deeper layers. The skip-layer concatenation approach later became the foundation for more complex neck designs such as PANet (YOLOv4) and BiFPN (YOLOv5/6), highlighting its long-term influence on the YOLO family.

3.3.2 Establishing the three-part architecture: Backbone – Neck – Head

A major milestone introduced by YOLOv3 was the clear formalization of the three-part architecture: Backbone – Neck – Head, which became the design foundation for most modern YOLO versions.

- Backbone: Responsible for feature extraction. YOLOv3 uses Darknet-53 with residual connections, allowing for more effective learning of features from input images.
- Neck: Formed by techniques such as multi-scale prediction and skip-layer concatenation, it facilitates the combination of information across different feature depths to enhance generalization.
- Head: Handles the final predictions, including the location, size, and class probabilities of bounding boxes at three different resolutions.

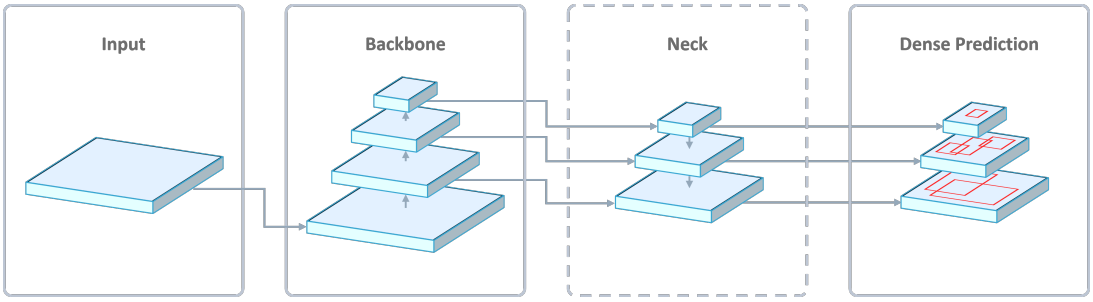

Figure 4: Architecture with 3 parts: backbone, neck, head

This architecture distinctly separates the role of each component, enabling later versions such as YOLOv4, YOLOv5, YOLOv7, and YOLOv8 to freely replace or optimize each module (e.g., swapping the backbone with CSPDarknet or ConvNeXt, or replacing the neck with PANet or BiFPN) without disrupting the overall structure.
This move toward a modular design not only increased flexibility in research but also made it easier for the community to fine-tune and improve individual components, allowing YOLO to evolve rapidly across generations.

3.4 YOLOv4

Due to the significant similarities between YOLOv4 and YOLOv5, and considering that the state-of-the-art section focuses on YOLO FaceV2, which is primarily based on the YOLOv5 architecture, our team will not cover YOLOv4 in the YOLO family discussion. Instead, we will place greater emphasis on YOLOv5.

3.5 YOLOv5 [8, 3]

YOLOv5 is an independently developed branch of YOLOv3, released by Ultralytics in 2020. Despite the absence of an official publication, YOLOv5 quickly became the most widely adopted version within the community thanks to its ease of use, flexible training pipeline, and strong performance. The model retains the backbone–neck–head structure from YOLOv3 but is comprehensively redesigned with modern components such as CSPDarknet, PANet, and the SiLU activation function. These improvements not only enhance accuracy but also significantly reduce model complexity, enabling efficient deployment on both edge devices and large-scale systems.
Within the scope of YOLO FaceV2, the model architecture is built upon YOLOv5. Therefore, in this **Related works** section, only up to YOLOv5 is discussed among the YOLO family, and later versions are not covered.

3.5.1 Core improvements

a. CSP (Cross-Stage Partial)

YOLOv5 integrates the Cross-Stage Partial (CSP) module into the backbone to overcome the limitations of Darknet53 used in YOLOv3, which is heavy, resource-intensive, and unsuitable for edge devices like IoT. Darknet53 suffers from

gradient redundancy in deep networks, increasing computational complexity and limiting deployment on low-power hardware. Inspired by DenseNet, CSP splits the feature map into two branches: one goes through dense convolutional blocks, and the other bypasses them directly, preserving original information and reducing computation. This allows YOLOv5 to reduce parameters (YOLOv5n: 1.9M vs. YOLOv3: 61M), speed up inference (6.3ms on GPU), and enhance gradient flow.

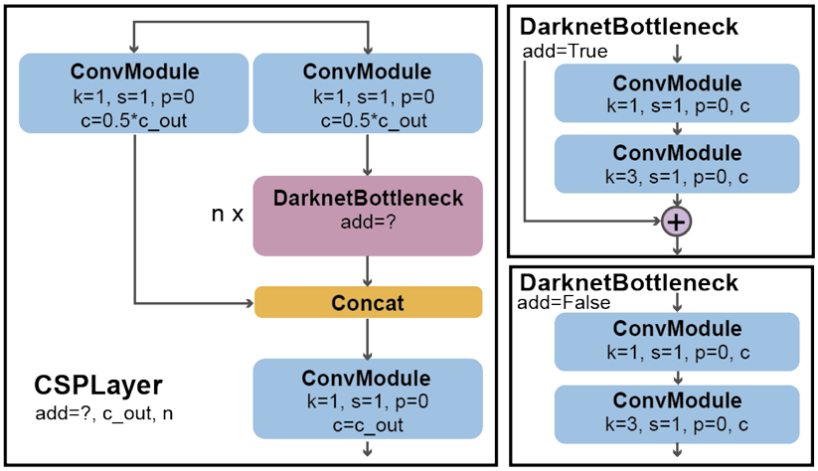

Figure 5: CSP block [8]

b. PA-Net (Path Aggregation Network) Neck

YOLOv5 replaces the FPN in YOLOv3 with PA-Net to improve multi-scale feature aggregation, addressing the limitation of FPN's top-down-only feature propagation, which hinders small object detection in datasets.

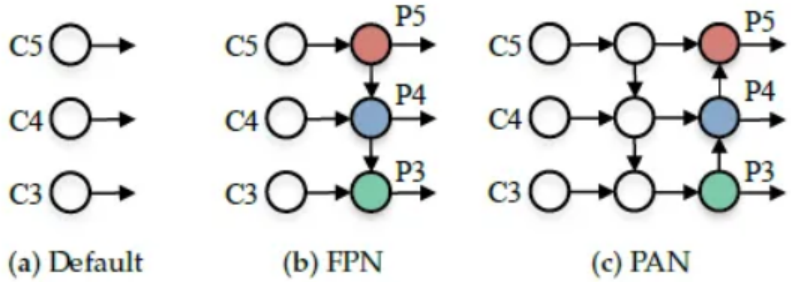

Figure 6: The structure of PAN

c. Optimized head

YOLOv5 improves the prediction head to overcome the drawbacks of YOLOv3, which has a large number of parameters, long inference time, and low efficiency on weaker hardware. The YOLOv5 head incorporates updates from PANet, reduces parameters, performs predictions at three scales, accelerates inference (YOLOv5s: 6.4ms on GPU), and improves mAP.

d. Expanded Receptive Field Module

YOLOv5 incorporates Spatial Pyramid Pooling Fast (SPPF) to address the limited receptive field in YOLOv3, which performs poorly on large objects or complex scenes due to the lack of modules like SPP. SPPF combines multi-scale pooling (5x5, 9x9, 13x13) to expand the receptive field without increasing computational cost, enhancing large object detection and mAP, making it suitable for surveillance or quality inspection tasks.

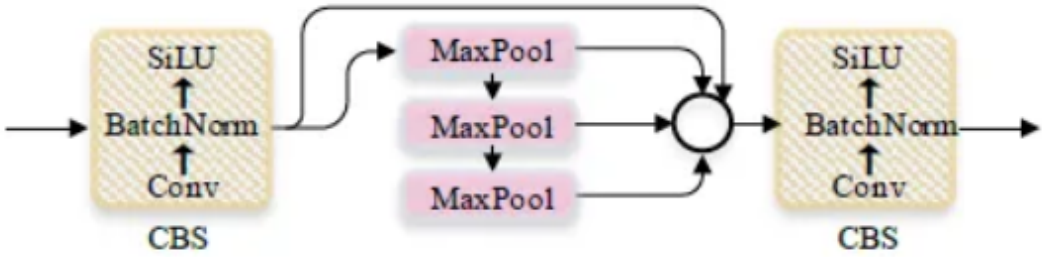

Figure 7: The structure of SPPF

3.5.2 Overall architecture of YOLOv5

After detailing the evolution from YOLOv1 to YOLOv5, we can observe a significant progression in model architecture. YOLOv5, which combines advanced improvements from YOLOv3 and modern design elements, is structured into three main components: Backbone, Neck, and Head, each playing a crucial role in optimizing the model's performance.

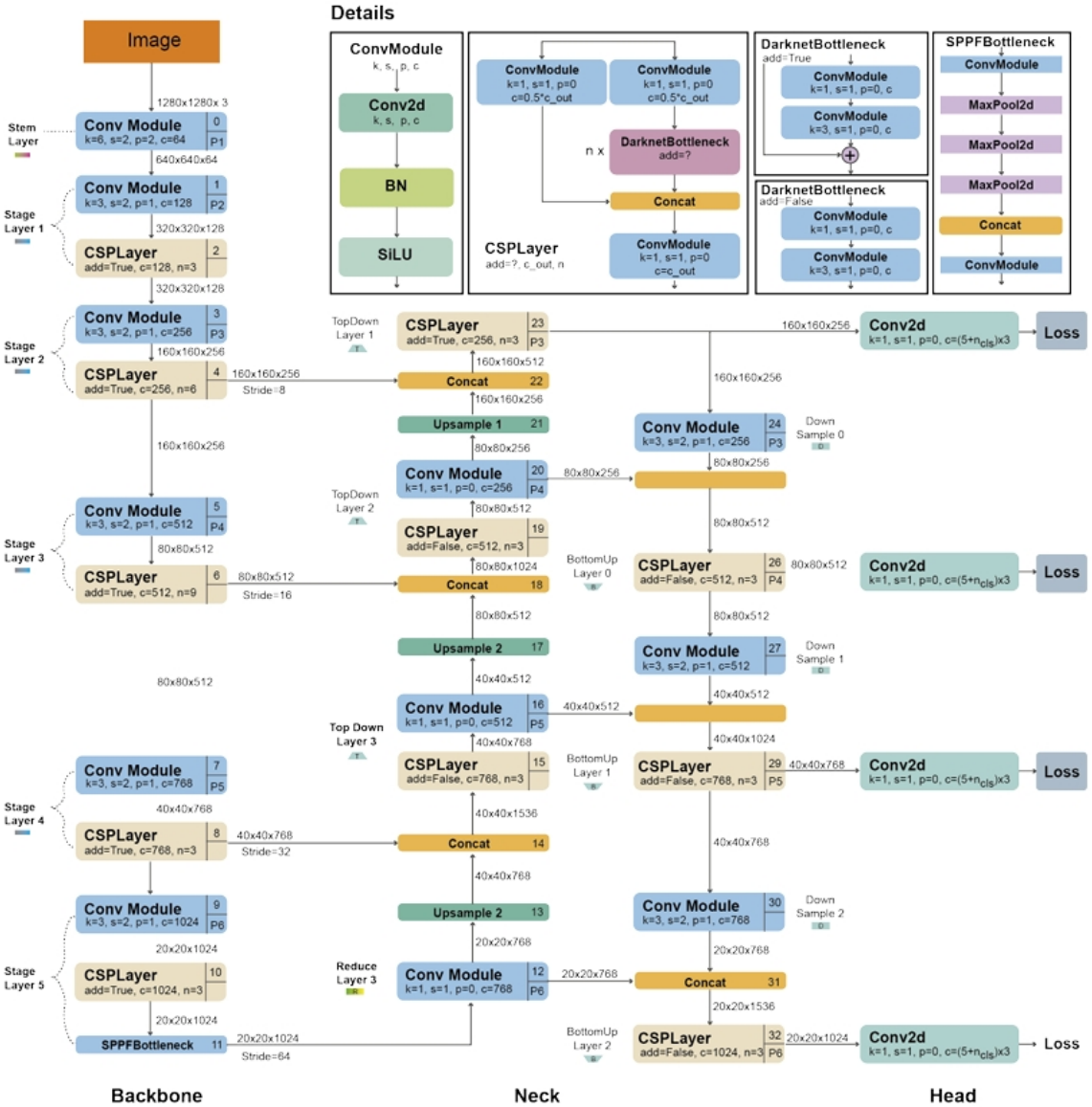

Figure 8: The architecture of YOLOv5 [8]

a. Backbone – Feature extraction

YOLOv5's backbone employs CSPDarknet53 with the integration of SPPF to enhance feature extraction. It consists of five main stages from P1 to P5, each with the following components and functions:

- Focus Layer (at P1): Slices and concatenates the image to reduce resolution while preserving important edge information.
- Conv + SiLU: Basic convolutional layers combined with the SiLU activation function, enabling the model to learn more expressive features.
- C3 Blocks (CSP Bottleneck): CSP blocks reduce parameters, improve gradient flow, and minimize computational redundancy through branching and shortcut connections.
- SPPF (at the end of P5): Applies multi-scale pooling (5×5, 9×9, 13×13) to expand the receptive field without increasing computational cost.

These improvements enable strong feature extraction, preserve original information, and extend contextual modeling capacity without significantly increasing the computation.

b. Neck – Multi-scale feature aggregation

The Neck leverages the PANet architecture (combining top-down and bottom-up paths) for feature aggregation across different levels:

- P5 → P4 → P3 (upsample + concat): Transmits features from deep layers to shallow ones, aiding in the detection of small objects.
- P3 → P4 → P5 (downsample + concat): Passes information from shallow to deep layers to enrich context and strengthen feature representations.
- C3 Blocks after each concat step: Refine and enhance the merged features.
- SPPF (at the end of P5): Applies multi-scale pooling (5×5, 9×9, 13×13) to expand the receptive field without increasing computational cost.

These enhancements improve detection across all object sizes, especially for small or partially occluded objects.

c. Head – Output prediction

The Head in YOLOv5 consists of three branches corresponding to different feature scales: P3 (small), P4 (medium), and P5 (large). Each branch predicts:

- Four bounding box coordinates (x, y, w, h) using CIoU Loss.
- One objectness score.
- N class probabilities (if multiple classes exist).

These improvements enable precise bounding box predictions and object classification at multiple resolutions, allowing the model to easily scale to different numbers of classes.

4. An overview of state-of-the-art method

YOLO Face V2 [9] represents a significant advancement compared to traditional face recognition techniques, effectively overcoming the limitations of both manual methods and intermediate deep learning models. While conventional approaches often require scanning image regions sequentially or generating separate region proposals, which can result in slow processing and failure to detect small faces, YOLO Face V2 leverages convolutional neural networks to analyze the entire image in a single forward pass. This approach allows the model to quickly locate and recognize faces in real time while greatly reducing computational costs, delivering superior performance in both speed and accuracy.
From an architectural perspective, YOLO Face V2 is structured around three core components: the backbone, the neck, and the head, along with a specialized module known as SEAM (Spatial Enhancement Attention Module). The **backbone** serves as the foundation by using deep convolutional layers to automatically extract high-quality features from the input image, capturing details such as edges, shapes, and textures of the face. These features remain robust even under challenging conditions such as varying lighting or complex viewing angles. The **neck** functions as an

intermediate aggregator, combining features from the **backbone** at multiple scales, from fine details like the eyes and nose to larger patterns like overall facial structure. This allows the model to accurately detect faces of various sizes, ranging from small faces in the background to larger ones in the foreground, before passing the features to the **head**. The **head** then performs the final task of prediction, generating precise bounding boxes to locate faces and determine whether the object is indeed a face, ensuring high reliability in the output.
One of the most important features of YOLO Face V2 is the integration of **SEAM**, a dedicated module designed to enhance attention to critical spatial features. Positioned between the neck and the head, **SEAM** acts as a smart filter that directs the model's focus to essential facial regions such as the eyes or mouth, even when parts of the face are occluded by elements like masks, hair, or shadows. By amplifying the recognition of the remaining visible features, **SEAM** enables YOLO Face V2 to maintain high accuracy under difficult conditions and significantly improves performance compared to models that lack such attention mechanisms.
Thanks to its well-optimized architectural design and its ability to efficiently utilize modern hardware such as graphics processing units, YOLO Face V2 achieves outstanding results in both processing speed and detection accuracy. The model can smoothly handle live video streams and deliver nearly instant outputs without compromising the quality of recognition. This makes YOLO Face V2 a highly suitable solution for real time applications, including security surveillance systems, facial recognition on mobile devices, and intelligent camera platforms where both speed and reliability are essential.

IV. State-of-the-art approach

1. Backbone

1.1 Architecture

The backbone of YOLO-FaceV2 [9] is built upon CSPDarknet53, a robust deep neural network architecture proven effective for object detection tasks, particularly in YOLOv5. This architecture is designed to extract multi-scale features from input images, providing a strong foundation for face detection under challenging conditions such as scale variations, occlusions, and low resolution.
The backbone structure consists of CSP (Cross Stage Partial) blocks and CBS (Convolution, Batch Normalization, SiLU activation) blocks. These blocks are organized to generate features at multiple levels, from shallow (P2) to deep (P5), to address the requirements of detecting faces of varying sizes.
Notably, in the P5 layer, the conventional Bottleneck block of CSPDarknet53 is replaced with the Receptive Field Enhancement (RFE) module to expand the effective receptive field, enhancing the capability to detect multi-scale objects. The backbone serves as the primary feature extractor, supplying input to the neck (SPP and PAN) and heads of the network.
This architecture not only maintains high performance but also optimizes the number of parameters, reducing the risk of overfitting and ensuring fast processing speeds suitable for real-time face detection applications.

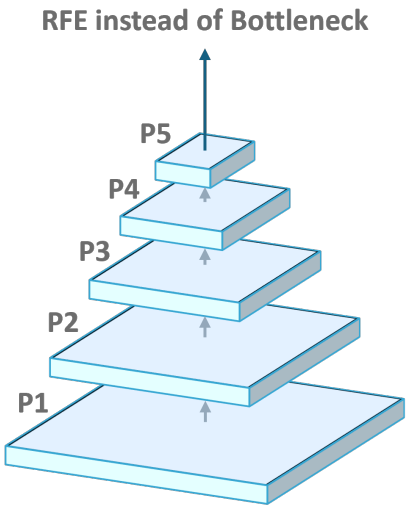

Figure 9: The backbone of YOLO-FaceV2

1.2 Scale-Aware RFE Model

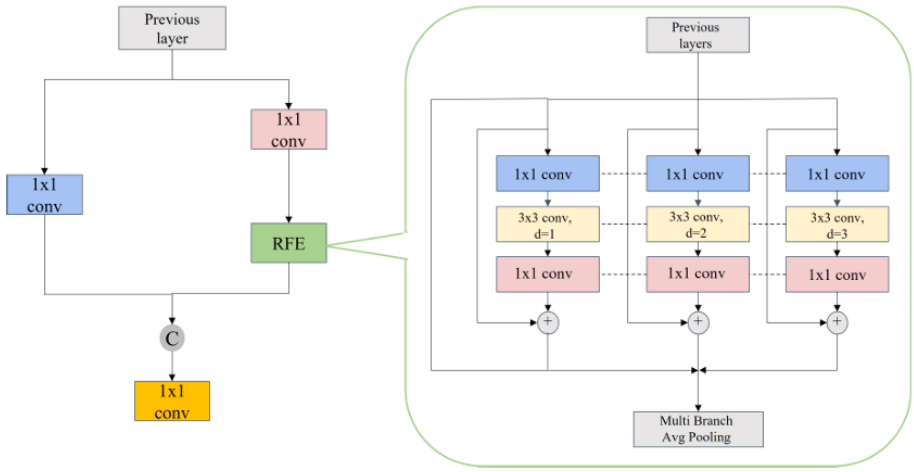

Figure 10: RFE block [9]

The **Receptive Field Enhancement (RFE)** module is designed to improve the detection of faces across different scales, particularly small faces, by expanding the receptive field of the feature map. RFE is integrated into the P5 layer of the backbone, replacing the Bottleneck block in the C3 module of YOLOv5. The module comprises two main components: The **multi-branch part** and The **gathering and weighting layer.**

- **The multi-branch part:** Consists of three branches with dilated convolutions of varying rates to capture multi-scale information, along with a residual connection to stabilize training.
- **The gathering and weighting layer:** Aggregates and balances information from the branches, incorporating an Average Pooling operation to enhance global context.

1.2.1 The Multi-branch part

The multi-branch part of RFE consists of three branches of dilated convolutions with different dilation rates (1, 2, and 3) and a fixed 3×3 kernel size, along with a residual connection. Each dilated convolution branch includes:

- A 1×1 convolution to reduce dimensionality.
- A 3×3 convolution with the specified dilation rate ((d = 1, 2, 3)).
- A 1×1 convolution to normalize the output.

These branches share weights, reducing the number of parameters and the risk of overfitting while maximizing the utilization of each data sample. The varying dilation rates enable RFE to capture diverse receptive fields, from short to long ranges, improving the detection of faces at different sizes, particularly small faces that tend to lose information after multiple convolutional layers.
Additionally, each branch incorporates an internal residual connection: the output of the branch is added to its input. This internal residual connection enhances training stability within each branch by preserving the original input features.
A residual connection is also incorporated as the fourth "branch." This residual connection directly passes the input of the RFE module (after the initial 1×1 convolution in the C3 module) to the summation step at the end of RFE, without any transformation. This connection prevents gradient explosion or vanishing during training, ensuring stability and effective learning.

1.2.2 The Gathering and Weighting Layer

The gathering and weighting layer aggregates information from the multi-branch part and applies weighting to balance the feature representations. The outputs of the three dilated convolution branches are summed together. Additionally, an Average Pooling operation is applied to the input of this layer to capture global context, and its output is also summed with the outputs of the dilated convolution branches. Finally, the residual connection adds the original input of the RFE module to this sum, producing the final output of RFE. This process ensures that features from various

receptive fields are combined efficiently, enhancing the feature map's representation without significantly increasing computational complexity.

1.2.3 Why Use RFE Instead of Bottleneck in P5?

Replacing the Bottleneck block in the P5 layer with the RFE module offers several key advantages:

- **Expanded Receptive Field**: The traditional Bottleneck block uses standard convolutions, limiting the receptive field and lacking flexibility to handle multi-scale faces, especially small ones. RFE, with dilated convolutions, expands the receptive field, allowing the network to capture broader spatial context and improving detection accuracy.
- **Enhanced Multi-Scale Capability**: The dilated convolution branches in RFE provide information from multiple receptive fields, enabling the network to handle faces of varying sizes, from small to large, whereas the Bottleneck focuses on a fixed receptive range.
- **Reduced Risk of Overfitting**: By sharing weights across branches and incorporating residual connections, RFE reduces the number of parameters compared to adding complex convolutional layers, maintaining high performance with lower computational overhead.
- **Optimized for Small Faces**: In face detection, small objects often lose information after multiple convolutional layers. RFE, with its ability to enhance the receptive field and integrate multi-scale information, helps preserve critical pixel information needed for detecting small faces.

2. Neck

2.1 Architecture

The neck of YOLO-FaceV2 is designed to aggregate multi-scale features extracted by the CSPDarknet53 backbone, enabling robust face detection across varying scales and occlusions. It comprises a Path Aggregation Network (PAN), which facilitates feature fusion across different backbone stages. Inherited from YOLOv5, the neck architecture is adapted for face detection by integrating feature maps from the P2 layer onward in the PAN to compensate for resolution loss and improve localization accuracy.

PAN: Path Aggregation Network
The Path Aggregation Network (PAN) aggregates feature maps from multiple backbone stages (P2, P3, P4, and P5) to provide multi-scale features to the detection heads. PAN employs a bottom-up and top-down pathway to fuse high-level semantic features with low-level spatial details, improving localization and detection accuracy. In YOLO-FaceV2, PAN is enhanced by incorporating the P2 layer (resolution $160 \times 160 \times 128$) to mitigate the loss of fine-grained details due to downsampling in deeper layers, which is particularly critical for detecting small or occluded faces [9].
The PAN stage consists of two pathways:

- **Top-Down Pathway**: This pathway propagates high-level semantic features from the P5 layer ($20 \times 20 \times 1024$, post-SPPF) to shallower layers. The P5 feature map is upsampled using nearest-neighbor interpolation to match the resolution of the P4 layer ($40 \times 40 \times 512$). A CBS block processes the P4 feature map, reducing its channels to 256, and it is concatenated with the upsampled P5 features. A C3 module with three Bottleneck modules refines the fused features, producing a feature map of $40 \times 40 \times 256$. This process repeats for P3 ($80 \times 80 \times 128$) and P2 ($160 \times 160 \times 64$), ensuring that high-level contextual information is integrated with finer spatial details.

$$\text{P5 to P4: } 20 \times 20 \times 1024 \rightarrow 40 \times 40 \times 256 \text{ (after upsampling and fusion)}$$
$$\text{P4 to P3: } 40 \times 40 \times 256 \rightarrow 80 \times 80 \times 128$$
$$\text{P3 to P2: } 80 \times 80 \times 128 \rightarrow 160 \times 160 \times 64$$

- **Bottom-Up Pathway**: This pathway aggregates low-level spatial features back to deeper layers to enhance localization. Starting from the P2 feature map ($160 \times 160 \times 64$), a CBS block with a 3×3 convolution and stride of 2 downsamples it to $80 \times 80 \times 128$, which is concatenated with the P3 feature map from the top-down pathway. A C3 module refines the fused features. This process continues for P4 ($40 \times 40 \times 256$), ensuring that spatial details are preserved in deeper layers.

$$\text{P2 to P3: } 160 \times 160 \times 64 \rightarrow 80 \times 80 \times 128$$
$$\text{P3 to P4: } 80 \times 80 \times 128 \rightarrow 40 \times 40 \times 256$$

The final output of the PAN stage consists of three feature maps at different scales: $160 \times 160 \times 64$ (from P2), $80 \times 80 \times 128$ (from P3), and $40 \times 40 \times 256$ (from P4). These feature maps are fed to the detection heads for predicting face bounding boxes, landmarks, and confidence scores. The inclusion of the P2 layer enhances the detection of small faces, while the bidirectional fusion ensures robust feature representation for complex scenarios.

3. SEAM: Occlusion-Aware Attention Network

In real-world face detection tasks, especially in unconstrained environments, faces are often partially or fully occluded by objects such as masks, sunglasses, hair, hands, or shadows. These phenomena lead to a series of challenges in feature extraction, specifically:

- **Alignment Errors:** When key facial points are occluded, the face alignment model may inaccurately localize critical parts like the eyes, nose, or mouth.
- **Local Aliasing:** Occluded regions cause spatial disruptions, leading the model to mislearn information in important areas.
- **Feature Loss:** When facial regions are hidden, important features may disappear or be replaced by the background, reducing the discriminative power between individuals.
- **Background Confusion:** The model may be misled by background regions that are irrelevant but have similar textures or colors to facial areas.

To tackle these challenges, SEAM is designed as an occlusion-aware attention module. It adjusts feature weights in a way that enhances unoccluded facial regions and suppresses occluded or background areas, while also learning the correlation between visible and occluded regions to recover lost information.

3.1 Structure of SEAM

SEAM comprises three main components:

- **(1) CSMM modules for multi-scale feature learning**
- **(2) a bottleneck-style MLP network for refining important features**

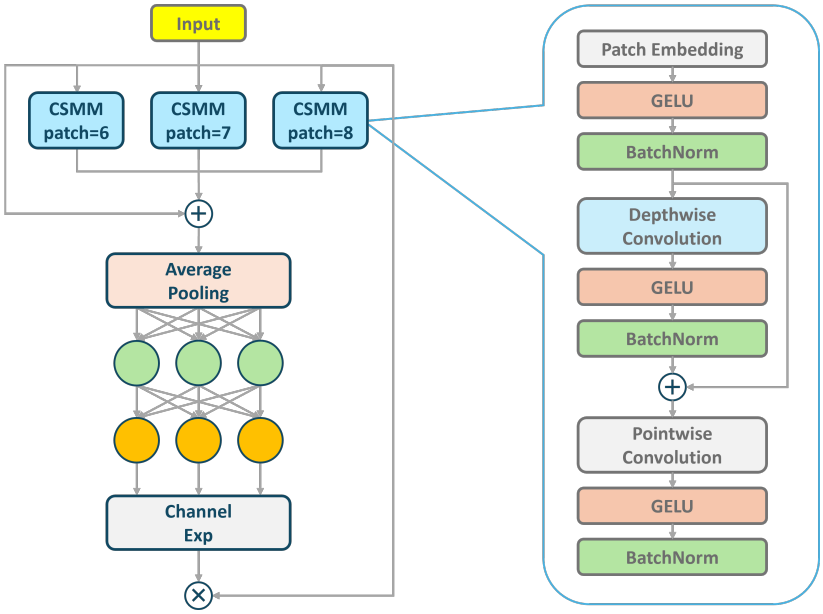

Figure 11: SEAM block [9]

3.1.1 CSMM (Channel and Spatial Mixing Module)

SEAM uses two CSMMs with patch sizes of 6×6, 7×7, and 8×8 to capture information at different scales—from fine details to global structure. Each CSMM performs four key steps:

- **Patch Embedding:** Encodes the image into patches.
- **Depthwise Convolution:** Applies convolution independently on each channel, enabling spatial pattern learning within channels without intermixing them.
- **Pointwise Convolution (1×1):** Connects inter-channel information by learning channel-wise correlations.
- **GELU + BatchNorm:** Adds non-linearity and stabilizes training.

The outputs of the three CSMMs are summed element-wise to produce a unified feature representation. This addition mechanism helps reduce noise and retain stable features across different scales, thereby helping the model resist local aliasing and feature loss due to downsampling or partial occlusion.

3.1.2 Bottleneck-Style MLP

After aggregation from CSMMs, the features pass through an average pooling layer to condense information and highlight strong regions. Next, the features are processed by a two-layer MLP with a structure of n → n/k → n (where k is the dimensionality reduction factor). This bottleneck architecture forces the model to retain only the most important information while eliminating redundant or noisy features. This component directly addresses the problems of feature loss and background confusion.
The output of the MLP undergoes an exponential transformation, mapping values from [0, 1] to [1, e]. This mapping allows stronger differentiation between important and unimportant features and reduces sensitivity to small positional errors, enabling the model to be more robust to imperfect alignment.
The final output of SEAM is a channel-wise attention weight map, which is directly multiplied with the original features. This multiplication enables the model to:

- Enhance unoccluded facial regions, highlighting truly important features.
- Reduce the influence of occluded or background areas, thereby decreasing noise and improving discriminability.

Thanks to this feature adjustment mechanism, SEAM effectively fulfills the role of an occlusion-aware attention network, providing cleaner and clearer information for the subsequent layers in the face detection pipeline.

4. Head

In the YOLOFacev2 model, the head component is designed with an architecture similar to that of YOLOv5. Both models adopt a multi-scale detection mechanism, enabling the detection of objects — in this case, faces — at various sizes. The head structure consists of three branches corresponding to three output layers from the neck, allowing the model to simultaneously predict bounding box coordinates and objectness scores at each grid point on the feature maps. By inheriting this design from YOLOv5, YOLOFacev2 maintains high performance in face detection, even under challenging conditions such as small face sizes, pose variations, or poor lighting.

5. Loss functions

In YOLO (You Only Look Once) models, the loss function is generally structured into three main components, each aiming to optimize different aspects of the object detection task simultaneously.
The first component, L_{cls} (Classification Loss), is responsible for penalizing errors related to the misclassification of detected objects. Specifically, when the model predicts the wrong label compared to the ground truth, L_{cls} measures this difference and guides the model to improve its classification accuracy.
The second component, L_{obj} (Objectness Loss), focuses on assessing the confidence of whether a predicted bounding box actually contains an object. This is crucial for controlling the number of false positives (predicting objects where there are none) and false negatives (missing real objects), as YOLO not only needs to classify objects correctly but also determine whether any object exists within a given grid cell.
Finally, L_{loc} (Localization Loss) penalizes errors in predicting the precise position and size of the bounding boxes. This component ensures that the model learns to draw bounding boxes as close as possible to the ground-truth boxes, including accurately predicting the center coordinates, width, and height. Depending on the specific YOLO version (such as YOLOv3 or YOLOv5), the localization loss can involve different variations like IoU Loss, GIoU Loss, or CIoU Loss to further enhance the accuracy not only in terms of overlapping area but also in terms of the alignment between boxes.
The tight integration of these three loss components enables YOLO models to achieve a strong balance between high detection accuracy and fast inference speed - two core requirements for real-time object detection tasks.
In the YOLO-FaceV2 architecture, the loss functions are specifically designed to address the unique challenges of face

detection, including scale variation, occlusion, and sample imbalance. The authors proposed and integrated multiple loss functions, each serving a distinct role - from optimizing the detection of small faces and handling occluded cases to balancing easy and hard samples. The seamless coordination among these loss functions not only improves detection accuracy but also ensures real-time performance, enabling YOLO-FaceV2 to achieve outstanding results on the WiderFace dataset.
IoU

5.1 Intersection over Union (IoU) loss

One of the primary loss functions employed in YOLO-FaceV2 is the IoU Loss, directly inherited from the YOLOv5 architecture. This loss measures the difference between the predicted bounding box and the ground-truth bounding box using the Intersection over Union (IoU) metric. Specifically, IoU is calculated as:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

representing the ratio between the overlapping area and the union area of the two bounding boxes. Consequently, the IoU loss function is formulated as:

$$\mathcal{L}_{IoU} = 1 - \text{IoU}$$

The IoU Loss offers notable advantages, particularly in accurately localizing objects of large and medium sizes, where the overlap between bounding boxes can easily reach high levels. However, a significant limitation of IoU Loss is its sensitivity to small positional deviations, especially when detecting small objects like tiny faces. Even minor shifts can substantially reduce the IoU value, resulting in decreased detection accuracy.
Within the YOLO-FaceV2 framework, the authors chose to retain IoU Loss to maintain strong performance on large and medium-sized faces. Nevertheless, to compensate for its shortcomings when dealing with small faces and challenging cases such as occlusions, additional loss functions were introduced and integrated alongside IoU Loss to comprehensively optimize the model's overall performance.
NWD

5.2 Normalized Wasserstein Distance (NWD) loss

A new loss function called NWD Loss (Normalized Wasserstein Distance Loss) is introduced to improve the detection of small faces. This loss is specifically designed to overcome the limitations of IoU Loss when dealing with small-scale objects.
Unlike traditional approaches that rely on the overlap between bounding boxes, NWD Loss models bounding boxes as two-dimensional Gaussian distributions and measures the similarity between the predicted box and the ground-truth box using the second-order Wasserstein distance (W_2^2), normalized by a dataset-related constant C. The loss function is defined as:

$$NWD(\mathcal{N}_a, \mathcal{N}_b) = \exp\left(-\frac{\sqrt{W_2^2(\mathcal{N}_a, \mathcal{N}_b)}}{C}\right),$$

$$\text{with: } W_2^2(\mathcal{N}_a, \mathcal{N}_b) = \left\| \left(\left[cx_a, cy_a, \frac{w_a}{2}, \frac{h_a}{2}\right]^{\mathrm{T}}, \left[cx_b, cy_b, \frac{w_b}{2}, \frac{h_b}{2}\right]^{\mathrm{T}} \right) \right\|_2^2$$

Where C is a constant closely related to the data set, $W_2^2(\mathcal{N}_a, \mathcal{N}_b)$ is a distance measure, and $\mathcal{N}_a$ and Nb are $\mathcal{N}_b$ distributions modeled by A $= (cx_a, cy_a, w_a, h_a)$ and B $= (cx_b, cy_b, w_b, h_b)$.
One of the major advantages of NWD Loss is its reduced sensitivity to object scale, enabling it to measure similarity even when the bounding boxes do not overlap. This property is particularly beneficial for detecting small or occluded faces.
In the YOLO-FaceV2 architecture, NWD Loss is combined with IoU Loss to balance performance between large and small face detection. Specifically, the total loss is computed as a weighted sum of IoU Loss and NWD Loss, with the sum of the weights equal to 1. The authors experimented with different weight combinations between IoU and NWD, including (1, 0), (0, 1), (0.5, 0.5), (0.4, 0.6), and (0.6, 0.4). Experimental results showed that the (0.5, 0.5) combination - meaning an equal contribution from both losses - achieved the best overall performance, and thus was selected for the final model.
Repulsion

5.3 Repulsion loss

Repulsion Loss is introduced by the authors to address the problem of intra-class occlusion, a common situation where multiple faces appear close to each other and may be misidentified due to overlapping. Repulsion Loss consists of two main components: **RepGT Loss** and **RepBox Loss**.

RepGT Loss (Repulsion from Ground Truth) aims to encourage predicted boxes to stay away from non-target ground-truth boxes. Specifically, for each predicted box P, the loss measures the overlap between P and the ground-truth box (not its main target) with the highest IoU. The loss is computed as follows:

$$L_{\text{RepGT}} = \frac{\sum_{P \in \mathcal{P}_+} \text{Smooth}_{\ln}\left(\text{IoG}(P, G^P_{\text{Rep}})\right)}{|\mathcal{P}_+|}$$

where $\text{IoG}(P, G)$ denotes the ratio between the intersection area and the area of the ground-truth box, and $\text{Smooth}_{\ln}$ is a smoothing function designed to reduce the influence of outlier values. The smoothing parameter $\sigma \in [0, 1)$ controls the sensitivity of this adjustment.
The $\text{Smooth}_{\ln}$ function is defined as:

$$\text{Smooth}_{\ln} = \begin{cases} -\ln(1-x) & \text{if } x \leq \sigma \\ \frac{x-\sigma}{1-\sigma} - \ln(1-\sigma) & \text{if } x > \sigma \end{cases}$$

RepBox Loss, on the other hand, ensures that predicted boxes belonging to different faces do not overlap excessively. This is particularly important because Non-Maximum Suppression (NMS) removes predictions with overlapping levels above a certain threshold, treating them as duplicate detections for the same object. Excessive overlap can therefore cause missed detections for closely positioned faces. RepBox Loss is formulated as:

$$L_{\text{RepBox}} = \frac{\sum_{i \neq j} \text{Smooth}_{\ln}\left(\text{IoU}(B^{p_i}, B^{p_j})\right)}{\sum_{i \neq j} 1\left[\text{IoU}(B^{p_i}, B^{p_j}) > 0\right] + \epsilon}$$

where B^{p_i} and B^{p_j} are predicted boxes associated with different faces.
In terms of benefits, Repulsion Loss helps reduce the missed detection rate in crowded scenes and decreases reliance on the NMS threshold, significantly enhancing the ability to detect closely packed faces. Within the YOLO-FaceV2 architecture, incorporating Repulsion Loss plays a crucial role in improving model accuracy, especially on hard subsets of data where face occlusions occur frequently.

6. Evaluation metric - mean Average Precision (mAP)

As a standard and widely used metric in object detection tasks, mean Average Precision (mAP) is also the primary evaluation metric employed in the YOLO-FaceV2 model. mAP measures the overall effectiveness of a model by averaging its precision across all levels of recall, based on the Precision-Recall (PR) curve. The calculation of mAP is typically performed at a specific Intersection over Union (IoU) threshold, which is set to 0.5 in the model, meaning a prediction is considered correct if the area of overlap between the predicted bounding box and the ground-truth box is at least 50% of their union.
The value of mAP is calculated as follows:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i$$

$$\text{AP} = \sum_k (\text{Recall}_k - \text{Recall}_{k-1}) \cdot \text{Precision}_k$$

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

Where, N and AP_i represent the number of classes or IoU thresholds and the Average Precision for the i^{th} class or at the i^{th} IoU threshold, respectively. AP is computed by averaging the precision values at various recall levels. This is achieved by plotting the Precision-Recall (PR) curve and calculating the area under the curve (AUC). The terms TP (True Positives), FP (False Positives), and FN (False Negatives) refer to the number of correctly predicted positive instances, the number of incorrectly predicted positive instances, and the number of missed detections (i.e., actual positive instances that were incorrectly classified as negative), respectively.
To better reflect the model's performance across different scenarios, the authors divide the WiderFace dataset into three subsets with increasing levels of difficulty: Easy, Medium, and Hard. The Easy subset consists of images with clear, large, and mostly unoccluded faces, making them relatively easy to detect. The Medium subset represents a moderate level of difficulty, containing smaller faces or faces in more complex contexts. Finally, the Hard subset includes the most challenging cases, they are faces that are small, heavily occluded, or captured in cluttered or low-visibility environments.

According to the results reported in the paper, YOLO-FaceV2 achieves impressive performance across all three subsets: 98.7% mAP on the Easy set, 97.2% mAP on the Medium set, and 87.7% mAP on the Hard set. These results indicate that the model performs well not only in simple scenarios but also demonstrates robustness in complex ones. Compared to other methods such as YOLO-FaceV1, YOLO5Face, RetinaFace, and RefineFace, YOLO-FaceV2 shows a clear advantage in overall accuracy, especially on the Hard subset, which is often considered the most important benchmark for evaluating a model's applicability in real-world conditions.
mAP not only reflects the model's ability to accurately detect objects but also its capacity to maintain high recall, even in difficult settings. Therefore, achieving high mAP scores, particularly on challenging subsets like Hard set, is strong evidence of the reliability and practical utility of a face detection model like YOLO-FaceV2.

7. Pros and Cons of YOLO-FaceV2

YOLO-FaceV2, built on YOLOv5, is optimized for real-time face detection, particularly for small and occluded faces. This section outlines its key advantages and limitations based on its performance on the WiderFace dataset [9].

Pros

- **Effective multi-scale detection**: The Receptive Field Enhancement (RFE) module expands the receptive field, improving detection across face sizes with mAP gains of 0.63, 0.60, and 0.57 on WiderFace's Easy, Medium, and Hard subsets [9].
- **Enhanced small face detection**: Including the P2 layer ($160 \times 160 \times 64$) in the Path Aggregation Network (PAN) boosts localization accuracy for small faces.
- **Computational efficiency**: Reduced PAN channel dimensions (e.g., $40 \times 40 \times 256$, $80 \times 80 \times 128$) and efficient SPPF and RFE designs enable real-time performance.
- **Robustness to occlusions**: RFE's long-range contextual capture enhances performance on occluded faces, notably in WiderFace's Hard subset.

Cons

- **Face-specific design**: Optimized for faces, YOLO-FaceV2 may require retraining for other objects, limiting general applicability.
- **High-Resolution Dependency**: Small face detection relies on high-resolution inputs, potentially underperforming with low-resolution images.
- **Memory demands**: The P2 layer increases memory usage, challenging for low-end devices.
- **Configuration opacity**: Limited documentation of the exact configuration (e.g., 'yolov5s_v2_RFEM_MultiSEAM.yaml' from Yolo-FaceV2 GitHub repository) may hinder replication [9].

YOLO-FaceV2 excels in real-time face detection, with strong multi-scale and occlusion handling capabilities, driven by the RFE module and P2 layer. However, its face-specific design, high-resolution dependency, memory demands, and limited configuration transparency pose constraints. It is ideal for face detection in challenging scenarios but may require adaptation for broader or resource-limited applications.

V. Experiments in the paper

Table 1: Comparison of YOLO-FaceV2 and existing face detectors on the WiderFace validation dataset. [9]

Method	**Detector**	**Easy**	**Medium**	**Hard**
Faster R-CNN				
	HR	0.925	0.91	0.806
	Face R-CNN	0.937	0.921	0.831
	FDNet	0.959	0.945	0.879
SSD				
	SFD	0.937	0.925	0.859
	SSH	0.931	0.921	0.845
	PyramidBox	0.961	0.95	0.889
	SFDet	0.954	0.945	0.888
RetinaNet				
	FAN	0.952	0.94	0.9
	SRN	0.964	0.952	0.901
	RetinaFace	0.969	0.961	0.918
	RetinaFace	0.971	0.962	0.92
YOLO				
	YOLO-FaceV1	0.989	0.872	0.693
	YOLOv5l-Face	0.963	0.954	0.908
	YOLOv7	0.969	0.955	0.880
	YOLO-FaceV2	**0.986**	**0.979**	**0.919**

According to the author: "Our YOLO-FaceV2 achieves 98.6%, 97.9%, and 91.9% on the three subsets, surpassing the previous state-of-the-art (SOTA) by 2.3%, 2.5%, and 1.1%, respectively. This improvement highlights our success in further enhancing model performance within the YOLO framework. However, our detector performed slightly worse than RetinaFace on the Hard subset. The reason is that RetinaNet-based face detectors adopt a two-stage classification and regression approach, which results in higher accuracy but at the cost of greater computational resource consumption, whereas our method is more efficient" [9].

VI. Our experiment

1. Qualitative Results on Sample Images

To visually assess the model's performance, we tested it on several sample images collected by our group. Each original image is shown alongside its corresponding detection result to highlight the model's effectiveness in identifying faces under varying conditions such as low light, occlusion, and pose variation.

(a) Original Image 1

(b) Detected Faces - Image 1

(c) Original Image 2

(d) Detected Faces - Image 2

(e) Original Image 3

(f) Detected Faces - Image 3

(g) Original Image 4

(h) Detected Faces - Image 4

Figure 12: Sample qualitative results showing original images and detection outputs.

2. Dataset selection and rationale

In our experiments, we used two datasets: **WIDER FACE** and **DARK FACE**, each serving a different purpose.

- **WIDER FACE** was used solely for comparison with the original research. Since the model was pre-trained on this dataset, our evaluation on WIDER FACE serves as a sanity check to confirm consistency with published results. Covers a wide range of conditions including variations in scale, pose, and occlusion.
- **DARK FACE** is the main dataset we selected for evaluation. It is specifically designed to test face detection performance under extremely low light conditions, one of the most challenging environments for current detectors. DARK FACE contains over 6000 images and 50396 annotated faces, offering diverse nighttime scenes including crowded areas and motion blur. We chose this dataset to stress-test the model's robustness and practical applicability in real-world scenarios where lighting is limited or inconsistent.

Our focus on DARK FACE allows us to better understand the limitations of existing models in non-ideal conditions, which are often overlooked in benchmark-driven research.

2.1 mAP calculation

Result

	WIDER FACE	**DARK FACE**
Confidence threshold	0.02	0.02
IoU threshold	0.5	0.5
Calculation method	Pascal VOC	Pascal VOC
Total ground truth boxes	39112	50396
Total predicted boxes	64885	55850
mAP	0.9290	0.2463

Table 2: mAP results on WIDER FACE and DARK FACE datasets

Evaluation: The model performs very well in the WIDER FACE dataset with a high mAP of 0.9290, indicating strong performance under standard lighting conditions and clear visibility. However, the performance drops significantly on the DARK FACE dataset, achieving only 0.2463 mAP. This suggests that the detector struggles with low-light environments and may require further enhancements such as image preprocessing, specialized training on dark images, or data augmentation techniques. The significant gap highlights the importance of addressing the robustness of the lighting in real-world deployment scenarios.

3. Hardware used

The experiments were carried out using the following hardware configuration:

- OS: Debian GNU/Linux 12 (bookworm) x86_64
- CPU: AMD Ryzen 5 6600H with Radeon Graphics (12) @ 3.300GHz
- GPU: NVIDIA GeForce RTX 3050 Mobile
- RAM: 16 GB

4. Observed weaknesses during experimentation

During testing, we identified several limitations of the face detection system:

- High false positive rate in nonface regions (e.g., hands, backs, masks). This issue is common even in modern detectors such as *RetinaFace*, which often require additional techniques such as segmentation masks or identity verification to mitigate false positives.
- The model struggles to detect faces that are upside down or heavily rotated (i.e., detect successfully with a low confidence score)

VII. Conclusions

In this project, our team successfully fulfilled all the requirements outlined in the assignment. We thoroughly explored various face detection methods presented in the related work, gaining a broad understanding of the field. In particular, we conducted an in-depth study of the YOLO-FaceV2 model and performed experiments on a dataset that was not originally evaluated in the paper, offering additional insights into its generalization performance.
However, due to hardware limitations, we were not able to fine-tune the model on our custom dataset. Instead, we relied on the best pre-trained weights provided by the original authors. Despite this constraint, our implementation and analysis demonstrate the practical applicability of YOLO-FaceV2 in real-world scenarios.

VIII. Demonstration

1. Technologies used

1.1 Web

The demo system is built on a modern technological foundation, focusing on optimizing performance, maintainability, and enhancing user experience. TypeScript is chosen as the primary language instead of JavaScript due to its strict type-checking capabilities, which help reduce runtime errors and improve overall development stability. On the frontend, ReactJS plays a central role with its component-based architecture, enabling flexible UI organization, efficient code reuse, and easy feature expansion.
The development workflow is accelerated through the integration of Vite—a high-speed bundler and development server that supports instant hot reloads and significantly shortens build times. Additionally, Tailwind CSS is used to design the interface following a utility-first approach, allowing styles to be applied directly in HTML while minimizing the need for custom CSS, all without compromising consistency or visual appeal.
The synergy between these technologies forms a well-balanced architecture that ensures both performance and maintainability, meeting modern web development standards: robust, minimalist, and user-friendly.

1.2 Model

You need to clone the GitHub repository from here and install the required libraries and the file `yolo-facev2_last.pt`, which are noted in `README.md`.

2. User interface

2.1 Home

Figure 13: Home page

Once accessing the demo page, users encounter the Onboard screen, which is the first interface. Here, they can choose between experiencing the product demo or viewing detailed information about the product and the development team.

2.2 Demo

Figure 14: Demo page

After selecting the Demo option from the Onboard screen, users will be redirected to the product demo interface. Here, they can upload an image or video file, which the system will process using the YOLO FaceV2 model. The model performs face detection on the uploaded content, and the results are displayed directly on the web interface with identified faces and their corresponding confidence scores. In addition to on-screen visualization, the system allows users to download the processed image or video, enabling viewing in higher resolution or use for other purposes.

Note: Users should ensure that the uploaded files are in one of the supported formats: ".jpg", ".png", ".bmp", ".jpeg", ".gif", or ".mp4". The downloadable output files are available in ".jpg" format for images and ".mp4" for videos.

2.3 About

Figure 15: About page

This page provides detailed information about the model used in the demo, specifically YOLO FaceV2, an advanced facial recognition algorithm based on YOLO (You Only Look Once). This model is designed to detect faces quickly and accurately in real-time.
Additionally, users can learn more about the development team, including a list of members involved in researching and implementing the system. Transparency in information enhances trust and offers a clearer perspective on the people behind this project.

3. How to conduct

This section describes the steps to demonstrate the functionality of the YOLO-FaceV2 web-based face detection system.

3.1 Project setup overview

- **Frontend**: Built with **TypeScript**, **ReactJS**, and **Tailwind CSS** for styling.

- **Backend**: A **FastAPI** server handles file uploads (image/video), processes them using **YOLO-FaceV2**, and returns the detected output.
- **Model**: The YOLO-FaceV2 model is cloned from the official GitHub repository: https://github.com/Krasjet-Yu/YOLO-FaceV2/tree/d9c8f24d5dba392ef9d6b350a7c50b850051b32b

3.2 Running the backend (API server)

1. Ensure `Python`, `torch`, `opencv-python`, `FastAPI`, and all YOLO-FaceV2 dependencies are installed.
2. Place the trained model weights `best.pt` in the same directory as the backend script.
3. Start the FastAPI server using
4. The API will be hosted at: `http://127.0.0.1:8000/detect`

3.3 Running the frontend (Web interface)

1. Install Node.js and required dependencies:

   ```
   npm install
   ```

2. Start the development server:

   ```
   npm run dev
   ```

3. The frontend will be accessible at: `http://localhost:5173`

3.4 Using the web application

1. Open the web interface in your browser.
2. Click the **Upload** button to select an image (`.png`, `.jpg`, etc.) or a video file (`.mp4`).
3. The frontend sends the file to the FastAPI server via a POST request.
4. The server detects faces using YOLO-FaceV2 and returns the processed result.
5. The result is displayed directly on the web interface.

3.5 Demonstration scenarios

- **Scenario 1: Image Detection** — Upload a portrait image and show the bounding boxes and confidence scores.
- **Scenario 2: Multiple Faces** — Upload a group photo and show how the model detects multiple faces accurately.
- **Scenario 3: Video Detection** — Upload a short `.mp4` file and demonstrate the frame-by-frame detection.

3.6 Technical notes

- The API supports **CORS**, enabling seamless communication with the frontend.
- The model runs on **GPU** if available, or falls back to **CPU**.
- Unified endpoint `/detect` supports both images and videos.

3.7 Additional considerations

- Prepare sample image/video files in advance to ensure a smooth demo.
- Monitor the backend terminal for detection logs and potential errors.
- Ensure the model file `best.pt` is properly loaded before starting.

4. Link to demo video

https://youtu.be/mx134hOXO14

IX. References

[1] Muhammad Hussain. Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11(7):677, 2023.

[2] Anil K. Jain and Stan Z. Li. *Handbook of Face Recognition*. Springer, New York, 2011.

[3] Rahima Khanam and Muhammad Hussain. What is yolov5: A deep look into the internal features of the popular object detector. *arXiv preprint arXiv:2407.20892*, 2024.

[4] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[5] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271, 2017.

[6] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. In *arXiv preprint arXiv:1804.02767*, 2018. https://arxiv.org/abs/1804.02767.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[8] Juan Terven and Diana Cordova-Esparza. A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. *Machine Learning and Knowledge Extraction*, 5(4):1680–1716, 2023.

[9] Ziping Yu et al. Yolo-facev2: A scale and occlusion aware face detector. *Pattern Recognition*, 155:110714, 2024.