



**Software Engineering Department**

**Braude College of Engineering**

# **Improvements in Person Identification in Video under constraints**

**Project code: 25-1-R-20**

**Capstone Project Phase B – 61999**

**Jul 2025**

**Asad Saffouri**

**Alexandra Simonishvili**

**Git repository link:**

**<https://github.com/asadsaffouri/phase-A.git>**

**<https://github.com/AlexaSimoni/Person-Identification.git>**

**Supervisor:**

**Mr. Ilya Zeldner**

<b>1</b>	<b>Abstract</b>	<b>5</b>
<b>2</b>	<b>Introduction</b>	<b>6</b>
<b>3</b>	<b>Background and Related Work</b>	<b>7</b>
3.1	Person Identification in Videos	7
3.2	Uses of Person Detection	7
3.3	CNN	8
3.3.1	CNN architecture	8
3.4	FlowNet	9
3.4.1	Architecture of FlowNet	9
3.4.2	Advantages of FlowNet	10
3.4.3	Why is FlowNet Unique?	10
3.5	FlowNetSimple	11
3.6	FlowNetCorr	11
3.7	YOLO	11
3.7.1	YOLO architecture	12
3.7.2	Why is YOLO unique?	12
3.8	YOLO V8	13
3.9	FaceNet	13
3.9.1	FaceNet Architecture	13
3.9.2	Why is FaceNet unique?	14
3.10	CLIP	15
3.10.1	CLIP Architecture	15
3.10.2	Why is CLIP unique?	16
<b>4</b>	<b>Research / Engineering Process</b>	<b>17</b>

4.1	Research direction	17
4.2	Development methods	17
4.2.1	Python for AI and Machine Learning as server	17
4.2.2	React for the Frontend	17
4.3	Algorithms and Dataset	18
4.3.1	YOLOv8 for object detection	18
4.3.2	FaceNet for Person Recognition	18
4.3.3	Integrating FlowNet for Person tracking	18
4.3.4	COCO Dataset	18
4.3.5	FiftyOne for data visualization	19
4.3.6	CLIP for flow tracking validation	19
4.4	Methods to measure precision	19
4.4.1	Methods to measure precision of object detection	19
4.4.2	Methods to measure precision of Person Identification	21
4.4.3	Methods to measure precision of Person tracking	22
4.5	Person Detection and Tracking in videos	25
4.5.1	The need for enhanced tracking in Real-World Scenarios	25
4.6	Performance Improvement	25
4.6.1	Improving tracking reliability by crop validation	25
4.6.2	Cyclic FaceNet Database Updates	27
4.6.3	Saving FlowNet-Tracked data to Database	27
4.6.4	Final Database filtering	28
5	Work Artifacts	29
5.1	System Flow	29

5.3	Algorithm Description –	30
5.4	FR & NFR Requirements	32
5.4.1	Functional Requirements	32
5.4.2	Non-functional Requirements	33
5.5	Working Progress	35
5.6	Pseudo-Code	35
5.6.1	YOLO Pseudo-Code	35
5.6.2	FaceNet Pseudo-Code	36
5.6.3	FlowNet Pseudo-Code	37
5.6.4	CLIP Pseudo-Code	38
5.7	Use Cases	40
5.7.1	System Use Case	40
5.7.2	Person Initialization Use Case	41
5.7.3	Person Tracking Use Case	42
5.7.4	CLIP Use Case	42
5.8	System Activity Diagram	43
5.9	Package Diagram	44
5.10	System Class Diagram	45
5.11	User Interface	45
5.11.1	Page 1: Loading Data Screen	45
5.11.2	Page 2: Analyzing Screen	47
5.11.3	Page 3: Result Screen	47
6	Testing Plan	49
6.1	Test Cases	49

<b>7</b>	<b>Expected Accomplishments</b>	<b>54</b>
<b>7.1</b>	<b>Problems</b>	<b>54</b>
<b>7.2</b>	<b>Criteria for Success</b>	<b>55</b>
<b>8</b>	<b>User Guide</b>	<b>56</b>
<b>8.1</b>	<b>Getting Started</b>	<b>56</b>
<b>8.1.1</b>	<b>Step 1: Loading Data Screen</b>	<b>56</b>
<b>8.1.2</b>	<b>Step 2: Processing Videos</b>	<b>58</b>
<b>8.1.3</b>	<b>Step 3: Viewing Results</b>	<b>59</b>
<b>8.2</b>	<b>Features Overview</b>	<b>61</b>
<b>8.2.1</b>	<b>YOLO-based Object Detection</b>	<b>61</b>
<b>8.2.2</b>	<b>FaceNet-based Person Recognition</b>	<b>61</b>
<b>8.2.3</b>	<b>FlowNet-based Flow Tracking</b>	<b>61</b>
<b>8.3</b>	<b>System Errors and Fixes</b>	<b>62</b>
<b>8.3.1</b>	<b>Common Problems and Fixes</b>	<b>62</b>
<b>9</b>	<b>Maintenance Guide</b>	<b>63</b>
<b>9.1</b>	<b>System Architecture</b>	<b>63</b>
<b>9.1.1</b>	<b>Frontend Interaction</b>	<b>63</b>
<b>9.1.2</b>	<b>Backend</b>	<b>63</b>
<b>9.1.3</b>	<b>Database Management</b>	<b>63</b>
<b>9.1.4</b>	<b>Dependencies</b>	<b>64</b>
<b>9.2</b>	<b>Scheduled Operational Tasks</b>	<b>64</b>
<b>9.2.1</b>	<b>Managing Python Virtual Environment</b>	<b>64</b>
<b>9.2.2</b>	<b>Data Backup Procedures</b>	<b>64</b>
<b>9.2.3</b>	<b>Log Management</b>	<b>65</b>

9.2.4	Security Maintenance	65
9.3	Future Improvements:	66
9.3.1	Monitoring and Alerts	66
9.3.1.1	Why Monitoring Matters?	66
9.3.1.2	Implementing Alerts	66
9.3.1.3	Responding to Alerts	66
9.3.2	Multi-Person Detection and Tracking	67
9.3.2.1	Why Multi-ID Handling Matters?	67
9.3.2.2	System Behaviour with Multiple IDs	67
9.3.2.3	Optimization by Multi-Person Tracking	68
9.4	Troubleshooting and Debugging	68
9.5	Documentation and Updates	69
10	References	70

## 1 Abstract

Today cameras and videos take a significant part in many various aspects of our lives. They already have a wide usage in security, production, culture entertainment and i.e. Along with that arises the need for efficient and precise person identification systems for videos. Our project provides a new approach for person recognition and tracking in video, combining person identification technology and object tracking in video algorithm. We use person identification techniques from previous project (AI-Driven Person Recognition and Identification in Videos [\[1\]](#) by Rom Harell and Itay Benjamin) that combines YOLO (You Only Look Once) algorithm for human detection and FaceNet algorithm for person face recognition and upgrade it with FlowNet algorithm for object tracking. Our approach promises significant improvements in the video person detection ability which is very useful in different institutions. Also used CLIP image encoding for tracked pictures extra validation.

## 2 Introduction

The rapid growth in the use of video content across various fields has highlighted the need for advanced person recognition systems. These systems are critical in areas such as security, surveillance (e.g., tracking individuals in crowded environments), smart devices vision and media analysis. Existing methods often struggle with accuracy and efficiency due to the inherent challenges of dynamic video environments, including changing lighting, movement, and resolution variability, as well as situations where faces are partially or fully obscured. Enhancing person recognition and tracking in such conditions is crucial to meet the needs of real-world institutions and applications.

Our project aims to develop an innovative system that leverages state-of-the-art deep learning technologies to identify individuals in video footage accurately. The system integrates with some powerful tools: FlowNet, which analyzes motion and tracks changes between consecutive frames [3], YOLO (You only look once) algorithm for real-time object detection [1,2], and FaceNet, which specializes in face recognition by mapping facial features into a Euclidean space [1,2]. By combining motion analysis and facial recognition, the system is designed to perform well even in challenging scenarios where both precision and speed are essential but having to deal with low video quality and different changing conditions.

In our system we combine the findings of the previous project (AI-Driven Person Recognition and Identification in Videos by Rom Harell and Itay Benjamin [1]) and Optical Flow algorithm (FlowNet: Learning Optical Flow with Convolutional Networks by Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D., & Brox, T [3], Computer Vision and Deep Learning: From Image to Video Analysis by Jeremy Cohen [9]) to optimize the results and to get the benefits from both worlds. Our system focuses on optimizing computational efficiency by analyzing motion dynamics and analyzing frames for face recognition. Instead of processing and tracking all the objects in a video, our system prioritizes zones at frames where a detection match occurred. This approach minimizes redundant computations and ensures the system can track individuals even when their faces are not always fully visible. By leveraging FlowNet ability to detect movement and FaceNet reliable recognition capabilities.

## 3 Background and Related Work

### 3.1 Person Identification in Videos

Person identification in videos is a tough problem due to several real-world challenges. Lighting in videos can change because of factors like time of day, different environments, or camera angles, making it harder to see faces and objects clearly. In crowded areas, faces might be partially blocked, complicating the recognition process. On top of that, scenes can change rapidly, making it difficult to track movement, identify people, or recognize features consistently. These challenges are further intensified by the huge amount of data in video streams, requiring systems to process many frames in real-time without slowing down. In the past, traditional methods relied on manually designed features, which were not as flexible in adapting to dynamic conditions. However, deep learning, especially Convolutional Neural Networks (CNNs) [9], has made a big difference by automating feature extraction, allowing the system to learn patterns from the data directly and perform better in complex scenarios. Recent advancements have leveraged CNNs and Recurrent Neural Networks (RNNs) to improve identification across video frames, with CNNs excelling at spatial features and RNNs [10], like ConvLSTMs, capturing temporal dynamics. These models have helped systems adapt to variations in appearance, lighting, and posture. Techniques like 3D pose estimation and multi-stream networks have further enhanced performance [11], especially in cases involving multiple perspectives. But there are still considerable obstacles, particularly around real-time processing, scalability, and adapting to different datasets, which continue to limit the effectiveness of person recognition systems in real-world applications.

### 3.2 Uses of Person Detection

Person detection in video streams has a wide range of applications across various industries. It is widely used in surveillance and security systems to monitor public and private spaces, helping detect intruders and raise alarms in case of suspicious activity. In face recognition systems, person detection is the first step for identifying and verifying individuals, used in security systems like access control, smartphone unlocking, or entity verification at the airport during passport control. Retailers use person detection for retail analytics, tracking customer behavior and foot traffic to optimize store layouts and



marketing efforts. It is also crucial in autonomous vehicles, helping detect pedestrians and cyclists to improve safety. In smart cities, person detection supports crowd monitoring, traffic management, and optimizing public transportation. Additionally, sports and performance analysis use person detection to track athletes. These applications highlight the versatility and importance of person detection across many sectors.

### 3.3 CNN

Convolutional neural networks (CNNs) have changed the computer vision and image processing field. CNNs are a class of deep neural network architectures, which get hierarchical features from raw pixel data and allow to achieve high accuracy in solving different computer vision tasks, in image classification, object detection, semantic and instance segmentation [1]. They consist of convolutional layers that apply learnable filters on input images resulting in low-level features like edges, textures, patterns and etc. They capture spatially localized patterns, so the network exploit inherent stationary images, and pooling layers that reduce computational complexity by downsizing the spatial dimension retaining the most informative parts. Nonlinear activation functions, such as ReLUs, enable CNNs to learn complex data patterns. As networks deepen, they combine lower-level features to form high-level, semantic representations. Despite their effectiveness, CNNs face challenges such as the need for large, annotated datasets, vulnerability to adversarial attacks, and difficulty in interpretation. Research is exploring methods like transfer learning and explainable AI to improve their robustness and transparency.

#### 3.3.1 CNN architecture

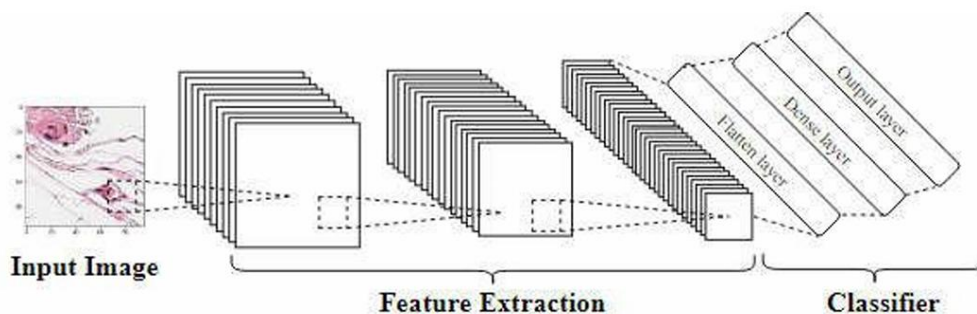


Figure 1: Convolutional neural network (CNN) diagram [1]

CNN architecture consists of convolutional layers that extract features from images, pooling layers that reduce spatial dimensions, and fully connected layers that combine features for final predictions. This structure allows CNNs to automatically learn hierarchical features and recognize complex patterns in data.

## 3.4 FlowNet

FlowNet is a deep learning framework made for optical flow estimation, which is an important task in video analysis. Optical flow is a way to measure how objects, surfaces, or edges move between two video frames. It creates a dense map of motion vectors that show the direction and speed of each pixel's movement. FlowNet is unique because it was the first convolutional neural network (CNN) that could learn optical flow directly from data in an end-to-end way [3], without needing traditional methods that used handcrafted features and slow optimization techniques.

Optical flow has many uses. For example, it helps in video stabilization by detecting and correcting unwanted camera movements. It is also used in action recognition to understand behaviors, object tracking to follow objects in a video, and motion analysis to study how things move in a scene. Older methods like Horn-Schunck [4] or Lucas-Kanade [5] were accurate but very slow and not practical for real-time applications.

FlowNet solves these problems by using a neural network that learns how motion works directly from data. It is much faster than the older methods and can be trained on different datasets to handle many types of scenarios. This makes FlowNet a great tool for tasks that need both speed and accuracy, like real-time video analysis.

### 3.4.1 Architecture of FlowNet

FlowNet takes two consecutive image frames as input and outputs a dense optical flow field. It consists of a deep convolutional network trained to predict motion vectors for each pixel. The network learns from synthetic datasets with ground truth optical flow (such as the Flying Chairs dataset used in FlowNet article [2]) ensuring robust performance.

FlowNet has two main variants: FlowNetSimple and FlowNetCorr [1,2].

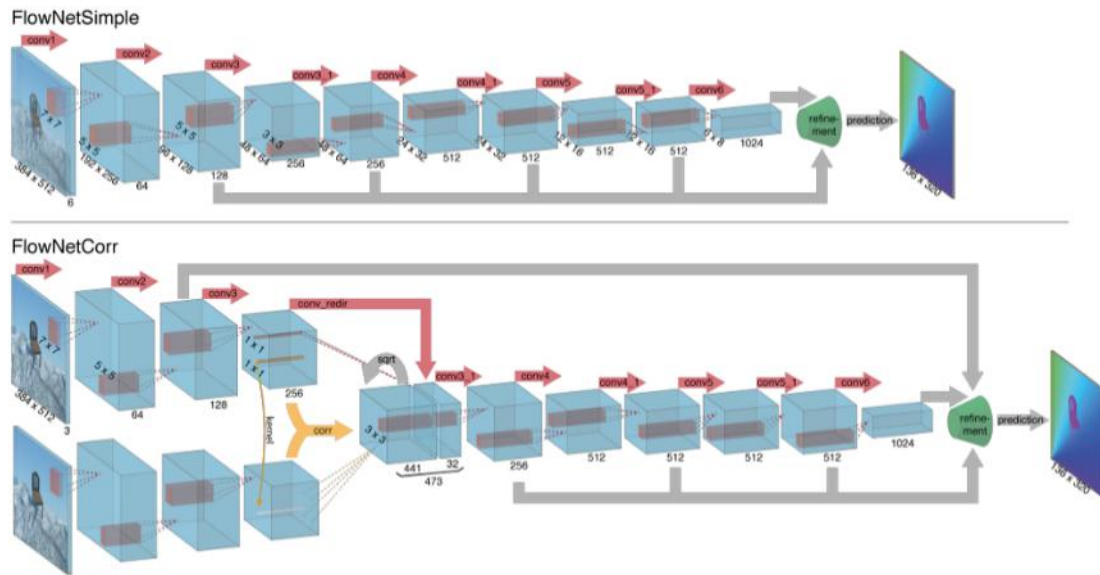


Figure 2. The two network architectures: FlowNetSimple (top) and FlowNetCorr (bottom).

### 3.4.2 Advantages of FlowNet

FlowNet offers several key advantages over traditional optical flow methods. First, it supports end-to-end learning, meaning it is fully trainable without the need for manually designed features, which simplifies the process and improves flexibility. Additionally, FlowNet is much faster than traditional optimization-based methods, making it well suited for real-time applications where speed is crucial. It also offers excellent scalability, as it can generalize across a variety of scenarios when trained on diverse datasets, making it adaptable to different tasks. Finally, FlowNet adaptability allows for fine-tuning on domain-specific data, significantly improving its performance for specialized applications.

### 3.4.3 Why is FlowNet Unique?

FlowNet is unique for several reasons that set it apart from other systems. First, it performs end-to-end learning, meaning it learns the entire process of motion estimation directly from the data without the need for manual features. Additionally, it operates at high speed, making it suitable for real-time applications where fast processing is required. Its adaptability is another key advantage, as FlowNet can be fine-tuned for

different applications by retraining it on custom datasets. Finally, its simplicity allows for quick understanding and easy implementation.

### 3.5 FlowNetSimple

FlowNetSimple is a basic version of a network designed to estimate motion in videos in a simple and effective way [3,13,15]. It works by taking two consecutive frames from a video and combining them into a single tensor with 6 channels (3 for each frame, representing RGB color). The network then uses convolutional layers to detect different patterns of motion and space. The initial layers focus on learning basic features like edges, textures, and motion, while the deeper layers capture features that are more complex. After the network learns these features, the image goes through de-convolution layers to expand it back to its original size, ensuring that the motion is predicted for every pixel in the image. The final output is a 2D optical flow field that shows the direction and strength of the motion for each pixel in the video.

### 3.6 FlowNetCorr

FlowNetCorr is an improved version of FlowNetSimple, adding a correlation layer to boost its performance [3,13,14]. Just like FlowNetSimple, it takes two consecutive frames and combines them into a tensor with 6 channels. The frames are processed separately through convolutional layers to extract features from each one. Then, the correlation layer calculates how similar the features from both frames are, helping the network had better capture long-range motion. After that, the combined features go through up sampling to create a dense optical flow field. The final output is a 2D optical flow field that shows the motion between the two frames.

### 3.7 YOLO

YOLO (You Only Look Once) is a real-time object detection system [1,4,6] developed by Joseph Redmon et al. YOLO uses deep learning to identify specific objects in images and videos. It is a single convolutional neural network that divides the input image into a grid system where each cell is responsible for detecting objects within itself. The network predicts both bounding boxes and class probabilities for these boxes simultaneously.

### 3.7.1 YOLO architecture

YOLO starts with an input image, which is divided into an  $N \times N$  grid. Each cell processes the image independently to detect objects [16]. After being trained, YOLO can recognize specific object classes (e.g., person, dog, wolf, cow). For each cell, it predicts the probability of each class, stored in an array format (e.g., 50% person, 30% dog). YOLO also predicts several bounding boxes per cell, each defined by its center coordinates, width, height, and confidence score (indicating the likelihood of overlap with a real object).

The algorithm loops through each cell, extracting class probabilities and selecting the bounding boxes with the highest confidence scores. If a class's confidence exceeds a threshold, its prediction is added to the final list. The output is a list of detected objects, each including the bounding box coordinates and predicted class, which are encoded later into a tensor with dimensions  $S \times S \times (B \times 5 + C)$  [1, 6].

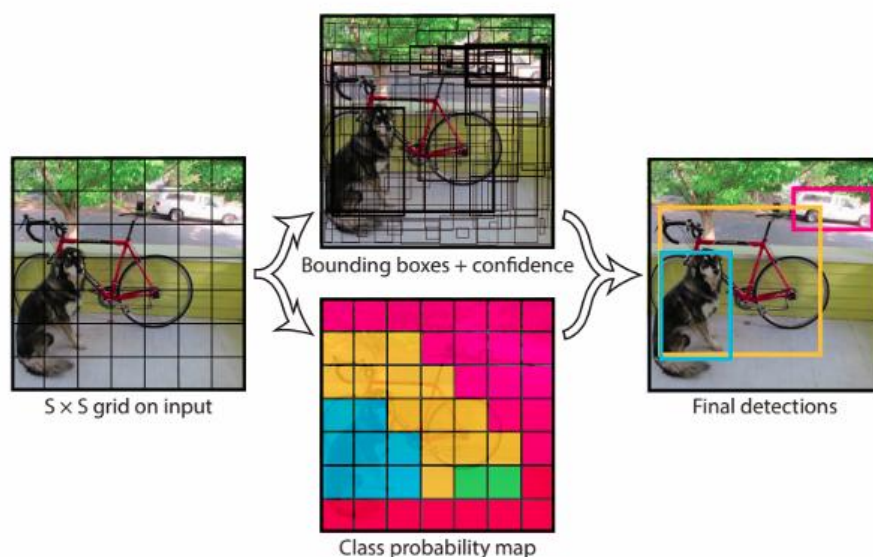


Figure 3: Yolo bounding box prediction simulation [1, 6]

### 3.7.2 Why is YOLO unique?

YOLO is unique by its ability to process information in real time. YOLO can handle up to 45 frames per second. The object detection is performed with only one evaluation of the neural network what ensures high efficiency. YOLO also can generalize and detect objects that were not specifically trained on [6]. So due to its speed, YOLO is promising for live streaming videos.

## 3.8 YOLO V8

YOLOv8 represents the latest version of the YOLO (You Only Look Once) object detection algorithm [1]. Over time YOLO architecture has evolved, with each new version introducing innovative updates that improve its performance and accuracy. When compared to YOLOv5, YOLOv8 offers faster and more precise object detection, making it the most accurate version to date and setting a new standard. The architecture of YOLO consists of three key parts: the backbone, which extracts features; the neck, which combines features from various scales, and the head, which makes the final detection predictions. YOLOv8 keeps this structure intact but introduces several changes to enhance its performance. YOLOv8 uses the CioU and DFL loss functions for bounding-box loss, along with binary cross-entropy for classification loss, which enhances object detection, especially for smaller objects.

## 3.9 FaceNet

FaceNet (developed by Schroff et al. [2]) brings a significant advancement in face recognition technology. FaceNet is a deep learning model which converts facial images into compact Euclidean space embeddings where the distances between embeddings reflect the similarity between faces. The architecture of FaceNet includes three key components: the feature extraction network, the embedding layer, and the triplet loss function.

### 3.9.1 FaceNet Architecture

FaceNet model converts facial images into Euclidean space embeddings for efficient similarity measurement. Its architecture consists of three main components [1,2]:

**Feature Extraction Network:** Using a deep Convolutional Neural Network (CNN) based on the Inception architecture, it processes face images to extract high-dimensional features that are invariant to pose, lighting, and expression, and the network learns to recognize them.

**Embedding Layer:** The extracted features are mapped to a fixed-size vector (typically 128 dimensions), where the Euclidean distances between embeddings represent face similarity. This helps cluster images of the same person closer together and those of different people farther apart. This approach fastens the search through large, labeled

faces dataset which has a various spectrum of face images from different subjects taken under many visual variations.

**Triplet Loss Function:** During training, FaceNet uses triplets consisting of an anchor image, a positive image (same person), and a negative image (different person). The goal is to minimize the distance between the anchor and positive embeddings while maximizing the distance to the negative embeddings, enabling better clustering of faces. During testing, similar images are considered to belong to the same person, while faces of different people are widely separated by the x-y axes.

FaceNet has been highly effective in face recognition tasks, outperforming humans on datasets like LFW [2], and has driven the development of improved datasets like VGGFace2. Its unified embedding approach has significantly impacted both research and practical applications in biometric identification and computer vision.



Figure 4: FaceNet model structure [1, 2]

The figure illustrates the model structure, where the network includes a batch input layer followed by a deep CNN. After processing through CNN, L2 normalization is applied to produce the face embedding, and during training, the triplet loss function is used to optimize the model.

### 3.9.2 Why is FaceNet unique?

FaceNet's unified embedding approach enables various face recognition tasks, such as verification, identification, and clustering. For example, one-to-one matching (face verification), one-to-many matching (face recognition) and clustering based on facial similarity. It has outperformed humans on benchmark datasets like LFW and influenced the creation of the VGGFace2 dataset [1, 2], which supports better face recognition across variations like pose and aging. FaceNet's focus on learning discriminative embeddings has shaped modern face recognition systems, with ongoing research addressing challenges like variations, fairness, and scalability. Overall, FaceNet has significantly advanced face recognition technology and its practical applications.



## 3.10 CLIP

CLIP (Contrastive Language–Image Pretraining) is a deep learning model developed by OpenAI that enables the comparison of visual and textual information in a shared embedding space [22,23]. Unlike traditional recognition systems that rely heavily on specific object classes or facial features, CLIP is designed to compare entire images based on their overall visual characteristics. It can match images of a person based on unique clothing patterns, color combinations, body size, or walking posture.

### 3.10.1 CLIP Architecture

CLIP consists of two core components:

**Image Encoder:** The image encoder used in popular CLIP implementations, such as openai/clip-vit-base-patch32 [23], is based on a Vision Transformer (ViT). The image is divided into small patches, which are processed through multiple layers of self-attention to extract complex visual features that represent the entire image's content.

**Text Encoder:** The text encoder processes natural language descriptions using a Transformer-based architecture. Although CLIP can compare images to text queries, in image-to-image comparison tasks, only the image encoder is typically used

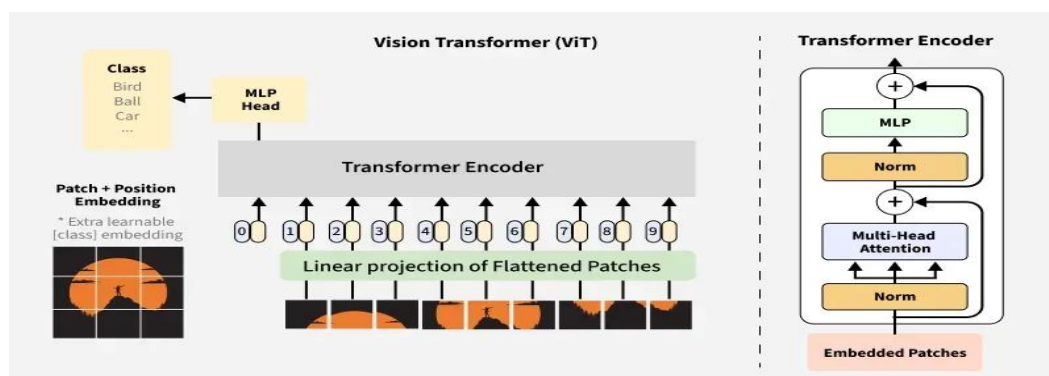


Figure 5: CLIP image encode diagram [24]

Key steps in CLIP encoding:

**Patch Embedding:** The image is split into patches, linearly embedded, and positionally encoded.



**Transformer Encoding:** The sequence of embedded patches passes through stacked transformer encoder layers, where multi-head attention captures spatial relationships across patches.

**Feature Extraction:** A special [CLS] token aggregates global context, resulting in a final 512-dimensional image embedding.

This embedding is then aligned with a text embedding in the shared CLIP space, enabling image-image and image-text comparisons. These embeddings are L2-normalized and can be directly compared using cosine similarity to determine how closely related they are based on overall appearance.

### 3.10.2 Why is CLIP unique?

CLIP is unique because it compares images based on overall appearance, not just specific features like faces. It compares images based on global visual features such as clothing, body shape, colors, and posture. Unlike models that need task-specific training, CLIP uses zero-shot learning, which means it can compare new images without additional fine-tuning. Another advantage of CLIP is its fast processing speed, making it suitable for real-time image comparison tasks.

## 4 Research / Engineering Process

### 4.1 Research direction

We build on the findings of the last research project, AI-Driven Person Recognition and Identification in Videos [\[1\]](#) by Rom Harell and Itay Benjamin and upgrade it further for better performances. The previous project combines YOLO (You Only Look Once) algorithm for human detection and FaceNet algorithm for person face recognition. In our project we upgraded it with FlowNet algorithm for object tracking to cover up the lack of tracking and identification abilities in different constraints. Our approach promises significant improvements in the video person detection ability.

### 4.2 Development methods

#### 4.2.1 Python for AI and Machine Learning as server

We consider using Python as it offers several advantages, including its simplicity and readability. Those make it ideal for rapid development and prototyping. The language is supported by a vast ecosystem of libraries and frameworks, such as OpenCV, Dlib, TensorFlow, and PyTorch, that provide ready-to-use tools for computer vision and machine learning tasks. Python benefits from strong community support and abundant online resources, facilitating problem-solving and collaboration. Python's versatility and easy integration with other technologies make it an efficient choice for developing AI and neural network applications [\[17\]](#).

#### 4.2.2 React for the Frontend

We selected React for the frontend due to its flexibility, efficiency, and scalability in creating user interfaces [\[18\]](#). Its component-based architecture enhances reusability and maintainability, while the virtual DOM optimizes performance by reducing unnecessary updates. React's unidirectional data flow makes debugging easier, and its hooks simplify state management. Additionally, React's extensive ecosystem of tools and libraries, along with a strong developer community, speeds up development and boosts productivity. Its seamless integration with various backend technologies further solidifies it as an ideal choice for modern web applications.

## 4.3 Algorithms and Dataset

### 4.3.1 YOLOv8 for object detection

In the previous project - AI-Driven Person Recognition and Identification in Videos (by Rom Harell and Itay Benjamin [1]), YOLOv8 and FaceNet were utilized for object detection and person identification. YOLOv8 was chosen for providing the best solutions among other environments as MobileNet V2 SSD or YOLO older versions (YOLOv5, YOLOv6, YOLOv7) [1] in terms of speed and accuracy.

### 4.3.2 FaceNet for Person Recognition

As we mentioned earlier, FaceNet model was chosen for face comparison and recognition process [1]. Using a triplet loss function, this model learns to distinguish faces with high accuracy, even under challenging conditions like lighting changes and partial occlusion. FaceNet is widely used in identity verification and surveillance systems due to its efficiency and robust performance.

### 4.3.3 Integrating FlowNet for Person tracking

In our updated version, we integrate the FlowNet optical flow algorithm (from the research FlowNet: Learning Optical Flow with Convolutional Networks by Dosovitskiy et al. [3]). We use it to enhance the system's motion analysis capabilities to meet the needs for advanced person recognition, especially in dynamic environments such as security and surveillance. Our system combines YOLO's real-time object detection with FaceNet's facial recognition, and now with FlowNet's motion tracking. Due to that, it efficiently handles challenges like changing lighting, resolution, and partial face occlusions. This integration improves tracking accuracy and computational efficiency by prioritizing areas of interest in video frames, ensuring more reliable identification even in complex scenarios.

### 4.3.4 COCO Dataset

For testing and evaluation, we are using the COCO 2017 Validation Dataset [19] as the last project [1], focusing on images containing people. This dataset offers detailed annotations, including bounding boxes and segmentation masks, which are vital for training advanced models. It contains 2,693 images that include the "person" category [1,19]. Using the COCO API from pycocotools, we filter the dataset to assess the model's

performance on person detection with metrics like mean Average Precision (mAP) and Average Recall (AR).

#### 4.3.5 FiftyOne for data visualization

We also use FiftyOne [20] to visualize and analyze the dataset which enabling us to compare model predictions with ground truth annotations. FiftyOne provided tools for plotting precision-recall curves and confusion matrices, helping us refine the model and improve performance in real-world scenarios.

#### 4.3.6 CLIP for flow tracking validation

In this version, we integrate the CLIP model (Contrastive Language–Image Pretraining by OpenAI [22,23]) to improve the accuracy of person tracking when using FlowNet. While FlowNet tracks motion across frames, it may sometimes lose the target, especially in crowded scenes or when the person changes orientation. CLIP is used to validate that the tracked person remains the same by comparing full-body crops based on clothing, posture, and body shape, not just facial features. This allows the system to continue reliable tracking even when faces are occluded or not clearly visible and helps prevent tracking errors and switching to unrelated objects. Its fast visual similarity checks add an extra layer of validation without slowing down the process, making the system more robust in real-world conditions.

### 4.4 Methods to measure precision

#### 4.4.1 Methods to measure precision of object detection

**Recall** - (also known as Sensitivity) measures how many of the actual objects in the dataset are detected by the model. It is defined as the ratio of true positive detections to the total number of ground truth objects (true positives + false negatives).

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

**Precision** - a measure of how many of the objects detected by the model are actually correct (i.e., true positives). It is defined as the ratio of true positive detections to the total number of detections made (true positives + false positives).

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

**Average Precision (AP)** - a measure that combines both precision and recall to provide a single metric for model performance. It is calculated as the area under the precision-recall curve. The precision-recall curve plots precision (y-axis) against recall (x-axis) for different threshold values. AP is usually computed for different Intersection over Union (IoU) thresholds, and the mean of these AP values gives a comprehensive view of the model's performance.

$$AP = \int_0^1 Precision(Recall) d(Recall)$$

**Average Recall (AR)** - is the average of recall values computed for different IoU thresholds and object categories. It provides an overall measure of the model's ability to detect objects across all categories and IoU thresholds.

$$AR = \frac{1}{N} \sum_{i=1}^N Recall_i$$

**Average Precision (mAP)** - the average of the AP values computed for different IoU thresholds and object categories. It provides an overall measure of the model's performance across all categories and IoU thresholds.

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \mid N \text{ is the number of IoU thresholds}$$

**Intersection over Union (IoU)** - a metric used to evaluate the accuracy of an object detection model. It measures the overlap between two bounding boxes: the predicted bounding box and the ground truth bounding box. The IoU value ranges from 0 to 1 where: 0 indicates no overlap between the bounding boxes, and 1 indicates a perfect overlap between the bounding boxes. IoU is calculated as the ratio of the area of overlap between the predicted bounding box and the ground truth bounding box to the area of their union.

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

#### 4.4.2 Methods to measure precision of Person Identification

**Precision** - a measure of how many of the objects detected by the model are actually correct (i.e., true positives). It is defined as the ratio of true positive detections to the total number of detections made (true positives + false positives).

This same calculation is useful for both calculating Verification Precision and Identification Precision.

In **Verification Precision** it measures the accuracy of verifying whether two images belong to the same person (i.e., face verification). It is determined by comparing the predicted similarity score against a threshold, with a positive match being a true positive (TP) and a mismatch being a false positive (FP) or false negative (FN).

In **Identification Precision** this metric evaluates how many of the top-ranked candidates in a face recognition task are correct matches for the identified person. For each query face, FaceNet ranks potential matches from a database, and the top-ranked match is considered a true positive if it correctly identifies the person.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

**True Acceptance Rate (TAR)** - a metric measures the percentage of correct matches (true positives) from all attempted comparisons for a particular identity.

$$Tar = \frac{Number\ of\ CorrectMatches}{TotalNumber\ of\ Comparisons}$$

**False Acceptance Rate (FAR)** – a metric measures the proportion of incorrect matches, i.e., the number of times the system falsely accepts an incorrect identity as a match.

$$Far = \frac{FalsePositives}{FalsePositives + TrueNegatives}$$

**Rank-N Precision** - This method measures how many times the correct person appears within the top N ranked candidates provided by FaceNet during an identification task. Typically, N is set to 1, but it can be expanded to higher values to measure the system's performance when a larger set of candidates is considered. This provides an insight into how well FaceNet ranks faces in terms of their similarity to the query face.

$$Rank - N\ Precision = \frac{CorrectMatches\ in\ TopN}{N}$$

**Cumulative Matching Characteristic (CMC) Curve** - a plot of the recognition rate (i.e., the percentage of correct matches) versus the rank of the correct match in a list of candidates. It usually shows the percentage of correct matches as a function of the rank position.

$$CMC@Rank - N = \frac{Number\ of\ CorrectMatches\ in\ Top - N}{TotalNumber\ of\ Queries} \times 100$$

**Equal Error Rate (EER)** - a performance metric that evaluates the trade-off between the False Acceptance Rate (FAR) and the False Rejection Rate (FRR). The EER is the point where the FAR and FRR are equal and is used to measure precision by balancing the model's false positives and false negatives.

$$EER = FAR = FRR \text{ at the threshold where both are equal}$$

**False Rejection Rate (FRR)** - this method measures the proportion of actual matches (true positives) that the system incorrectly rejects. It calculates the ratio of false rejections (when the system fails to recognize a correct match) to the total number of positive cases (true positives and false negatives).

$$FRR = \frac{FalseNegatives}{False\ Negatives + TruePositives}$$

#### 4.4.3 Methods to measure precision of Person tracking

**End-Point Error (EPE)** – a method measures the difference between the predicted flow vectors and the ground truth flow vectors at the endpoints. It is one of the most used metrics to evaluate the accuracy of optical flow methods.

There  $v_i$  is the ground truth flow vector for the pixel  $i$ ,  $\hat{v}_i$  is the predicted flow vector for the pixel  $i$  and  $N$  is the total number of pixels. A lower EPE indicates that the predicted optical flow vectors are closer to the ground truth, signifying better tracking accuracy.

$$EPE = \frac{1}{N} \sum_{i=1}^N \|v_i - \hat{v}_i\|$$

**Average Optical Flow (AOF)** - this metric evaluates the average magnitude of optical flow across all pixels or regions in the frame. It helps in assessing how well FlowNet tracks the motion of objects across frames.

There  $v_i$  is the flow vector at pixel  $i$  and  $N$  is the total number of pixels in the frame. The AOF provides an overall measure of the motion in the video. A higher average optical flow typically corresponds to more significant movement, which is often desirable for tracking objects in dynamic scenes.

$$AOF = \frac{1}{N} \sum_{i=1}^N \|v_i\|$$

**Tracking Accuracy** - this metric evaluates how closely the predicted tracking paths of objects align with their ground truth trajectories. It assesses how well FlowNet tracks an object across multiple frames. The method counts the number of successful objects matches across consecutive frames, indicating how well FlowNet tracks an object's movement. A higher tracking accuracy means better tracking performance.

$$TrackingAccuracy = \frac{Number\ of\ CorrectObjectPredictions}{TotalNumber\ of\ ObjectPredictions}$$

**Motion Boundary Accuracy (MBA)** - a metric that evaluates the accuracy of object boundaries by comparing the predicted motion field with the actual motion in the frame. It is particularly useful for tracking objects that have sharp boundaries or undergo significant motion.

There  $B_i$  is the true boundary at pixel  $i$  and  $\hat{B}_i$  is the predicted flow vector for the pixel  $i$ . **IoU** stands for Intersection over Union, which measures the overlap between the true and predicted boundaries. A higher Motion Boundary Accuracy indicates that the optical flow method accurately tracks the boundaries of moving objects, which is important in scenarios like object segmentation or detection.

$$MBA = \frac{1}{N} \sum_{i=1}^N IoU(B_i, \hat{B}_i)$$

**Object Tracking Success Rate** – this metric measures the percentage of frames where the tracked object is correctly identified and followed over time. It helps evaluate how robust the object tracking is in maintaining the identity of the tracked object across frames. A higher success rate indicates that the object is consistently tracked correctly across frames. Success is typically defined by the object being correctly identified and staying within a predefined tracking region.

$$TrackingSuccess\ Rate = \frac{Number\ of\ Successful\ Frames}{TotalNumber\ of\ frames} \times 100$$



**Intersection over Union (IoU)** - a metric used to evaluate the accuracy of an object detection model. IoU for object tracking evaluates the overlap between the predicted bounding box of the tracked object and the ground truth bounding box at each frame.

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

**F-Score for Object Tracking** - is the harmonic mean of precision and recall, used to evaluate the trade-off between the accuracy of tracking and the completeness of tracking across frames. The F-Score balances precision (how many of the predicted tracks are correct) and recall (how many of the actual object tracks are captured), giving a holistic view of tracking performance.

$$F - score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

**Tracking Precision** - This evaluates the accuracy of the predicted positions of the object in the tracked frames compared to the ground truth positions.

There  $P_i$  is the true position of the object in frame  $i$  and  $\hat{P}_i$  is the predicted position of the object in frame  $i$ .  $N$  is the number of frames. These metric measures how close the predicted object positions are to the actual positions across all frames. Lower values of tracking precision indicate better tracking performance.

$$TrackingPrecision = \frac{1}{N} \sum_{i=1}^N \|P_i - \hat{P}_i\|$$

## 4.5 Person Detection and Tracking in videos

### 4.5.1 The need for enhanced tracking in Real-World Scenarios

Person detection in videos is essential across many fields, including law enforcement, security, search and rescue, transportation hubs, smart cities, and event monitoring. In these environments, tracking individuals accurately is critical for public safety, missing person searches, crowd control, and surveillance. The original project [1] relied solely on object detection and face recognition using a combination of YOLO and FaceNet. But the major challenge in these real-world scenarios is that faces are often obscured, turned away, or blocked by objects or other people. Relying solely on face detection is not enough, as losing visual contact when the face is not visible can interrupt tracking or lose it completely. To solve this, advanced tracking methods are needed that can follow people based on motion and overall appearance, and not just facial features. Continuous tracking, even when faces are not visible, is vital for maintaining reliable person identification in complex, crowded, and fast-changing video environments.





	Original solution: YOLO+FaceNet	Our solution: YOLO+FaceNet+FlowNet
Good face visibility		
Bad face visibility		

Table 1: Detection results using Yolo+FaceNet detection vs Yolo+faceNet+FlowNet tracking

## 4.6 Performance Improvement

### 4.6.1 Improving tracking reliability by crop validation

FlowNet tracking improves motion analysis, but motion alone is not sufficient to ensure identity consistency. To solve this, we implemented dual validation for FlowNet-tracked crops using both FaceNet and CLIP. Each crop generated by FlowNet is now validated using FaceNet's facial similarity check and CLIP's full-body visual similarity comparison.

FaceNet focuses on matching faces, while CLIP validates appearance based on clothing, posture, and body shape, which is useful to validate that it's still the same person even if he turns around. This two-step validation helps to prevent the system from mistakenly switching to other moving objects or people, especially in scenes where facial visibility is limited. By using both models, the system significantly reduces identity drift and increases confidence in tracking accuracy.

FaceNet & FlowNet				
FaceNet & FlowNet + FaceNet validation				
Facenet & FlowNet + CLIP validation				
FaceNet & FlowNet + FaceNet + CLIP validation				

Table 2: Detection results using FlowNet tracking with and without FaceNet and Clip validation



#### 4.6.2 Cyclic FaceNet Database Updates

One of the key improvements we added is the cyclic update of the FaceNet embedding database every 50 frames. In the original system, the reference embeddings were static and limited only to ones extracted from user provided pictures before the search. By cyclically refreshing the embeddings using both the original user-provided images and new crops detected by FaceNet during processing, the system now adapts to the person's evolving visual representation, like more expressions and face visual degrees. This dynamic update allows the system to maintain more accurate identity verification over time, even when visual conditions change. As a result, the system becomes more robust in tracking the correct person across long sequences.

DB not updated			
DB updated every 50 frames			

Table 3: Detection results with FaceNet DB being updated during run (every 50 frames) and without

#### 4.6.3 Saving FlowNet-Tracked data to Database

In addition to uploading FaceNet-detected crops, we expanded the system to upload unique FlowNet-tracked crops to vary the data. Only crops that meet the required similarity thresholds are saved to the database. By storing these FlowNet-validated crops alongside FaceNet detections, the system builds a more complete and diverse dataset for each tracked individual, even across frames where the face was not clearly visible. This approach enhances the system's ability to maintain long-term person tracking and improves future identification accuracy by providing more varied, high-quality embeddings in the database.













Without FlowNet detections in DB						
With FlowNet detections in DB						

Table 4: examples of Person detected crops saved to DB with FaceNet detections with and without FlowNet detections

#### 4.6.4 Final Database filtering

At the end of the tracking process, we added an additional filtering step before uploading FlowNet-tracked crops to the database. Each tracked crop must pass FaceNet and CLIP verification to ensure only high-quality, correctly matched embeddings are saved. This filtering prevents false positives from entering the database, keeping it clean and reliable for future reference. By combining this filtering step with cyclic updates and dual validation, the system builds a more accurate and trustworthy record of tracked individuals across frames.

## 5 Work Artifacts

### 5.1 System Flow

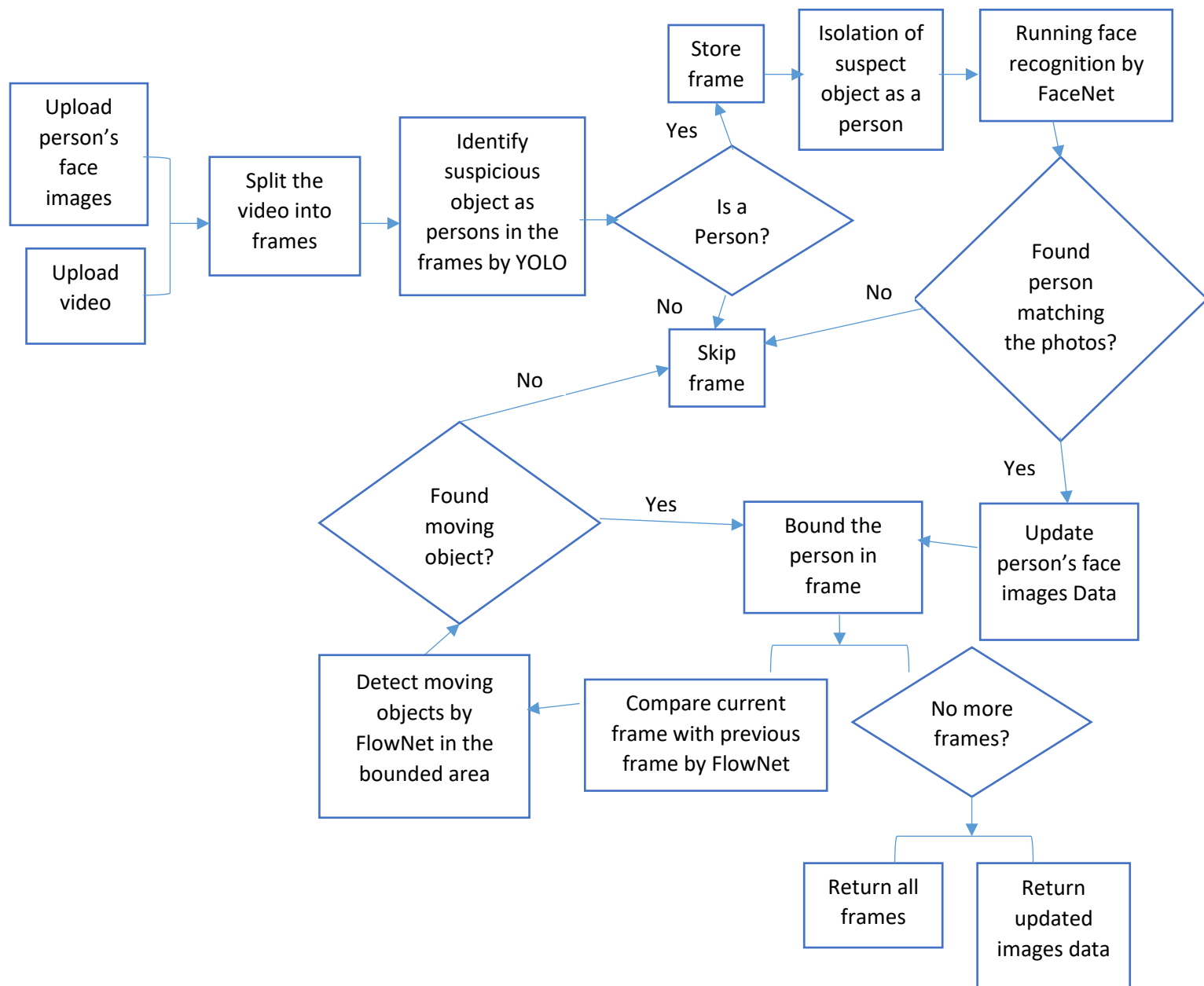


Figure 6: System flow diagram

## 5.2 Architecture

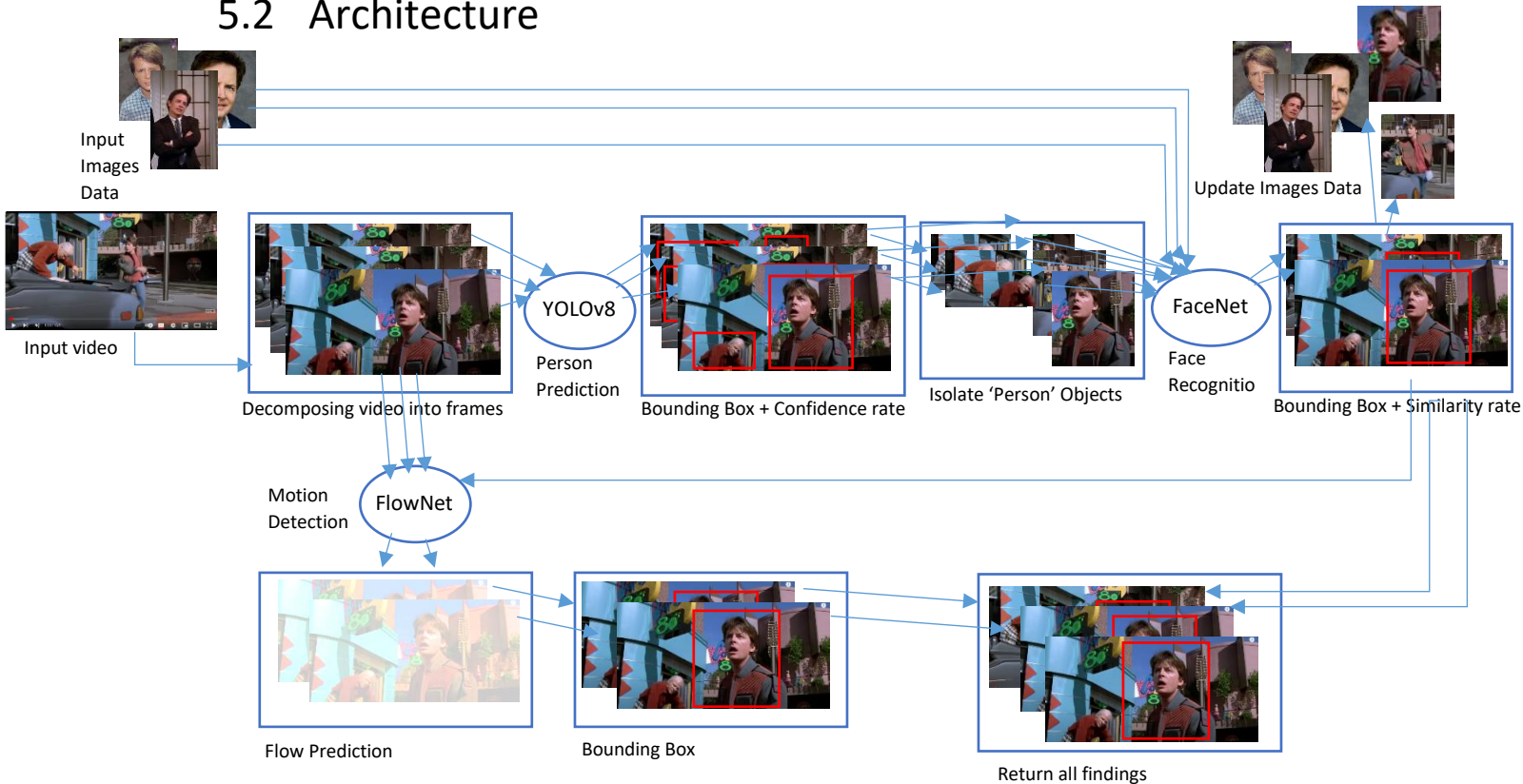


Figure 7: Architecture diagram

## 5.3 Algorithm Description –

### System Input:

The system receives two inputs: a video that needs to be analyzed and some images of a specific person to compare against the video. The goal is to prepare these inputs for further processing and analysis.

### Preprocessing:

The video is divided into individual frames (images), where each frame is saved and processed separately. This step allows the system to transform the video into a series of frames, which are easier to analyze individually. The output of this stage is a sequence of numbered frames ready for analysis.

### Detection with YOLO:

For each frame, the YOLO (You Only Look Once) model is used to detect objects, particularly people, within the image. YOLO returns bounding boxes, which are

rectangular areas that indicate the location of detected individuals. This step is crucial as it narrows down the parts of the image that need further analysis, improving efficiency. The output is a list of bounding boxes for each frame, showing where people are located.

#### **Face Detection with FaceNet:**

The bounding boxes from YOLO are passed to the FaceNet model, which is designed to detect and recognize faces. FaceNet analyzes the faces within the bounding boxes and compares them to the data of input images of the person. This comparison allows the system to determine whether the person in the video matches the one in the image. The output of this step is a list of identified people, including a verification of whether the person from the image is present in the current frame, and his matching rate.

#### **Motion Analysis with FlowNet:**

FlowNet is used to analyze motion between consecutive frames. It compares the current frame with the previous frame. FlowNet computes the motion only in regions where YOLO detected people, and where FaceNet confirmed that the person matches the target images. This step produces motion data, including direction, speed, and movement path for each identified person.

#### **Result Integration:**

The system integrates the results from the previous stages: the location data from YOLO, the identity verification from FaceNet, and the motion data from FlowNet. For each person, the system combines and shows their detected location, identity, and movement across frames. The result is a comprehensive list of individuals found to be matching the person from the images provided by the user, showing bounded characters with matching rate, their location and how they move across the video. The system stores matching individuals' images for later comparisons and update image data with high rate findings.

#### **Final Output:**

The final output provides detailed information about each identified person, including whether they appear in the video based on the target images, where they are located in each frame and how they move overtime. The system also provides updated person's faces images data.



## 5.4 FR & NFR Requirements

### 5.4.1 Functional Requirements

No.	Requirement
1	The system shall provide a user interface for users to upload videos and images.
2	The system shall provide upload functionality.
2.1	The system shall allow users to upload a video.
2.2	The system shall allow users to upload pictures.
3	The system shall decompose the uploaded video into individual frames.
3.1	The system shall perform preprocessing on extracted frames
3.2	The system shall perform preprocessing on the uploaded pictures.
4	The system shall detect person in each frame using the YOLO model.
4.1	The system shall generate bounding boxes around detected persons.
5	The system shall isolate detected faces within bounding boxes.
6	The system shall generate embeddings for detected faces using FaceNet.
6.1	The system shall classify a detected face as a match if the similarity score exceeds a threshold.
7	The system shall save the matching detected face into FaceData.
8	The system shall analyze motion between consecutive frames using FlowNet.
8.1	The system shall correlate motion data with detected persons for tracking.
9	The system shall track detected persons across multiple frames.

9.1	The system shall store frames where the detected person appears.
9.2	The system shall store timestamps for matched detections.
10	The system shall display the user the video frames, the bounding boxes, the similarity score and the timestamps where detection occurred.
10.1	The system shall allow users to download results, including detected frames and motion data.
12	The system shall display error messages for invalid file uploads.
13	The system shall notify users if no match is found.

*Table 5: Functional requirements*

## 5.4.2 Non-functional Requirements

No.	requirement	type
1	The system will avoid running unnecessary person detections in frames where no object found.	performance
1.1	The system will avoid performing unnecessary calculations in frames where no expected match for the person we are looking for.	performance
2	The system should be capable of processing video streams in real-time, ensuring that motion detection, face recognition, and person identification happen with minimal latency.	performance
2.1	The system should ensure that each video frame is processed efficiently without causing significant delays.	performance
3	The system should be able to handle videos of varying lengths without significant performance degradation.	Scalability
3.1	The system should be optimized to handle different video resolutions and qualities without compromising processing speed or accuracy.	Scalability

4	The system should be able to handle multiple images of person for search process.	Scalability
4.1	The system should be optimized to handle different picture resolution and qualities.	Scalability
5	The system must provide high accuracy in both face recognition (using FaceNet) and motion detection (using FlowNet) to minimize false positives and negatives.	Accuracy
5.1	The system should avoid unnecessary analysis by detecting and excluding frames where no people are present.	Accuracy
6	The system should be highly available, with minimal downtime.	Reliability
7	The system should provide an intuitive interface for users to upload video files and images, start processing, and view results easily.	Usability
7.1	The system should allow users to specify the person to track in a video through simple image input.	Usability
8	The system should be designed for easy maintenance and updates, with clear documentation and modular code.	Maintainability
8.1	The system should make it easy to find and fix problems quickly.	Maintainability
9	The system will use advanced machine learning algorithms for facial recognition (FaceNet), person detection (YOLO), and motion analysis (FlowNet)	Technology
10	The system should be optimized to process data quickly and efficiently.	Optimization

*Table 6: Non-functional requirements*

## 5.5 Working Progress

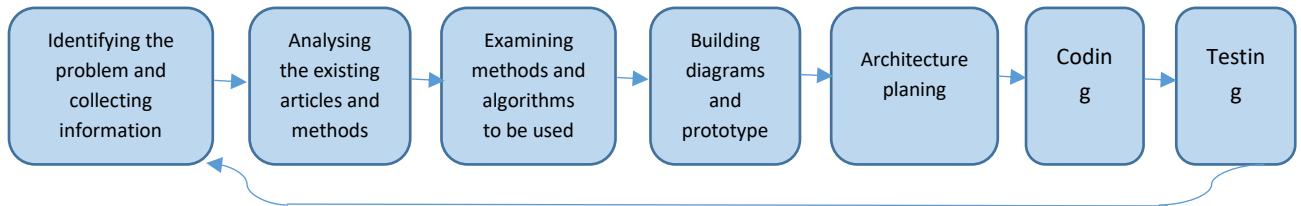


Figure 6: Work flow during semester

## 5.6 Pseudo-Code

### 5.6.1 YOLO Pseudo-Code

#### Input:

frame - A single image or video frame for object detection.

#### Output:

List of detected objects with bounding boxes, class labels, and confidence scores.

#### 1. Load YOLO Model:

- Initialize YOLO with pre-trained weights and configuration.
- Load predefined class labels (e.g., 'person').

#### 2. Preprocess Frame:

- Resize the frame to match the input size required by YOLO.
- Normalize pixel values.

#### 3. Perform Detection:

- Pass the preprocessed frame through the YOLO model.
- Extract raw detections (bounding boxes, class probabilities, confidence scores).

#### 4. Post-Process Detections:

- Filter detections based on confidence threshold.
- Apply Non-Maximum Suppression (NMS) to reduce overlapping bounding boxes:
  - Retain the box with the highest confidence.
  - Discard overlapping boxes with high IoU.

#### 5. Return Results:

- For each valid detection, return:
  - Bounding box coordinates.
  - Class label.
  - Confidence score.

### 5.6.2 FaceNet Pseudo-Code

#### Input:

- detected\_faces - List of face regions detected by YOLO.
- Input\_image – List of input face images to compare against.

#### Output:

- List of matched faces with bounding boxes and similarity scores.

#### 1. Preprocess the input face and detected faces:

- Resize the Input\_image and detected\_faces to match FaceNet input size.
- Normalize pixel values to range [-1, 1].
- Convert faces to tensor format.

#### 2. Generate embeddings:

- For each image in Input\_image:
  - Pass it through the FaceNet model to obtain its embedding.
- For each detected\_face:
  - Pass it through the FaceNet model to obtain its embedding.

#### 3. Compare embeddings:

- For each detected\_face embedding:
  - Compute similarity using cosine similarity:  $\frac{\vec{b} \cdot \vec{a}}{\|\vec{b}\| \|\vec{a}\|} = \text{Cosine Similarity}$
  - similarity = dot(embedding\_Input\_image, embedding\_detected\_face) /  
 (||embedding\_Input\_image || \* ||embedding\_detected\_face||)
  - If similarity > threshold:
    - Mark the face as a match.

#### 4. Return matched faces:

- List of detected faces with bounding boxes and similarity scores above the threshold.

### 5.6.3 FlowNet Pseudo-Code

**Input:**

- frame\_n - Current frame
- frame\_n\_minus\_1 - Previous frame
- regions\_of\_interest - Identified regions by FaceNet

**Output:**

- Motion data for detected regions (e.g., motion vectors, direction, speed)

1. Preprocess the frames:

- Resize frame\_n and frame\_n\_minus\_1 to match FlowNet input size.
- Normalize pixel values to range [0, 1].
- Convert both frames to blob format.

2. Extract regions of interest:

- For each region detected by FaceNet in frame\_n:
  - Crop the corresponding region in frame\_n\_minus\_1.
  - Crop the same region in frame\_n.

3. Compute optical flow:

- For each region of interest:
  - Pass the cropped regions (from frame\_n and frame\_n\_minus\_1) through the FlowNet model.
  - Obtain motion vectors representing pixel displacements between frames.

4. Post-process motion data:

- Aggregate motion vectors for each region.
- Compute:
  - Average motion direction.
  - Average motion speed.

5. Return motion data:

- For each region:
  - Motion vector (direction and speed).

## 5.6.4 CLIP Pseudo-Code

(Used later to validate FlawNet crops)

**Function:** GET\_CLIP\_EMBEDDING(image)

**Input:**

image: A single image in BGR or RGB format.

**Output:**

normalized\_embedding: A 512-dimensional vector representing the image in CLIP space.

**Steps:**

1. Convert image to RGB:  
image\_rgb  $\leftarrow$  CONVERT\_TO\_RGB(image)
2. Divide image into fixed-size patches:  
patches  $\leftarrow$  SPLIT\_IMAGE\_INTO\_PATCHES(image\_rgb)
3. Flatten each patch:  
flattened\_patches  $\leftarrow$  FLATTEN\_PATCHES(patches)
4. Apply linear projection to each patch:  
patch\_embeddings  $\leftarrow$  LINEAR\_PROJECTION(flattened\_patches)
5. Add positional encoding:  
embedded\_patches  $\leftarrow$  ADD\_POSITIONAL\_ENCODING(patch\_embeddings)
6. Prepend learnable [CLS] token:  
sequence  $\leftarrow$  CONCAT([CLS\_TOKEN], embedded\_patches)
7. Pass through Transformer encoder layers:  
for layer in TRANSFORMER\_ENCODER\_LAYERS:  
sequence  $\leftarrow$  layer(sequence)
8. Extract global representation from [CLS]:  
image\_embedding  $\leftarrow$  sequence[0]
9. Normalize to unit length (L2 norm):  
normalized\_embedding  $\leftarrow$  image\_embedding / L2\_NORM(image\_embedding)
10. Return normalized\_embedding

**Function:** COMPARE\_IMAGES(image1, image2)

**Input:** image1, image2: Two images to compare

**Output:** similarity: Cosine similarity score between embeddings

**Steps:**

1. Get embedding of image1:  
 $\text{emb1} \leftarrow \text{GET\_CLIP\_EMBEDDING}(\text{image1})$
2. Get embedding of image2:  
 $\text{emb2} \leftarrow \text{GET\_CLIP\_EMBEDDING}(\text{image2})$
3. Compute cosine similarity:  
 $\text{similarity} \leftarrow \text{COSINE\_SIMILARITY}(\text{emb1}, \text{emb2})$
4. Return similarity



## 5.7 Use Cases

### 5.7.1 System Use Case

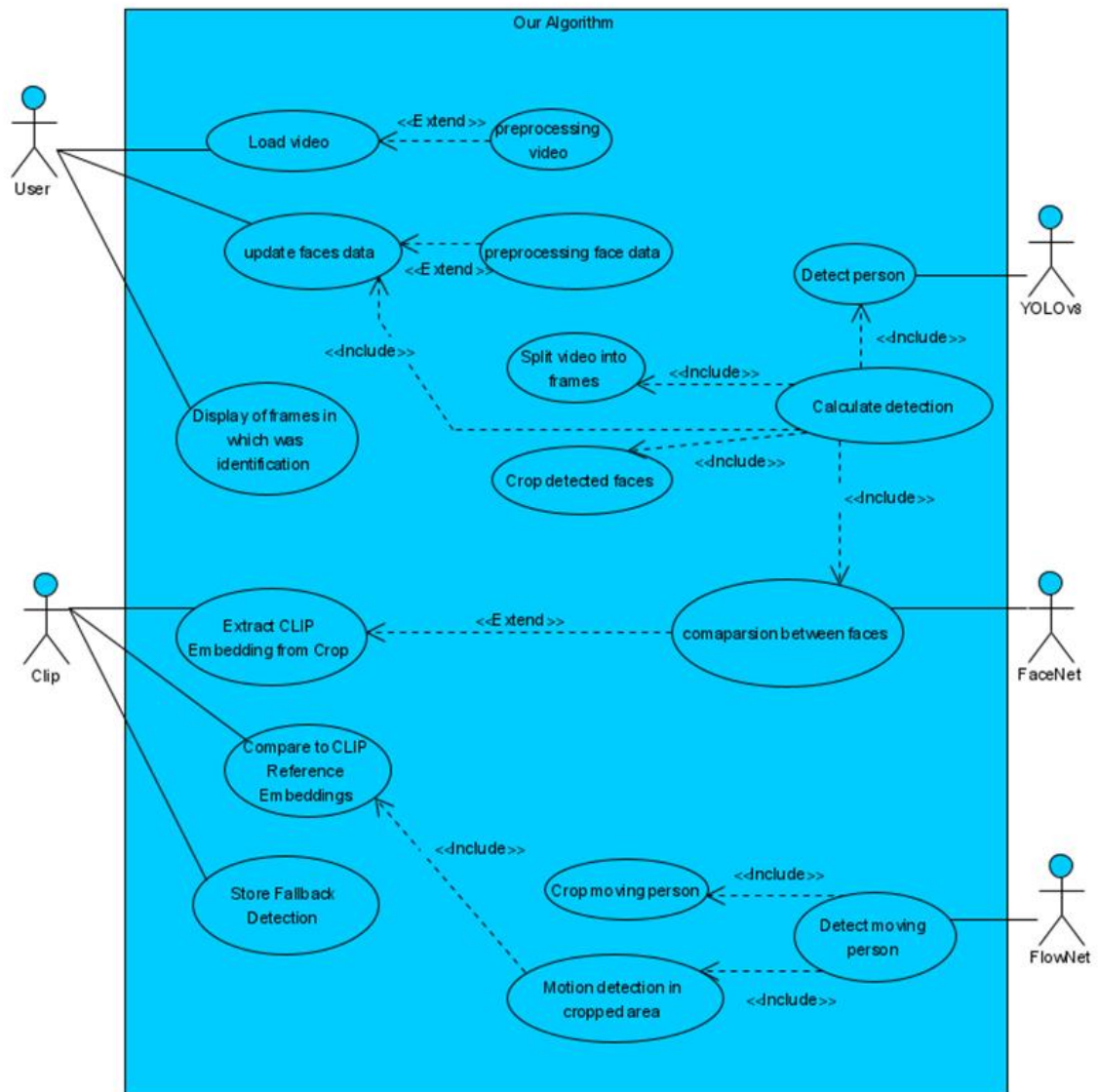


Figure 7: System use case diagram

## 5.7.2 Person Initialization Use Case

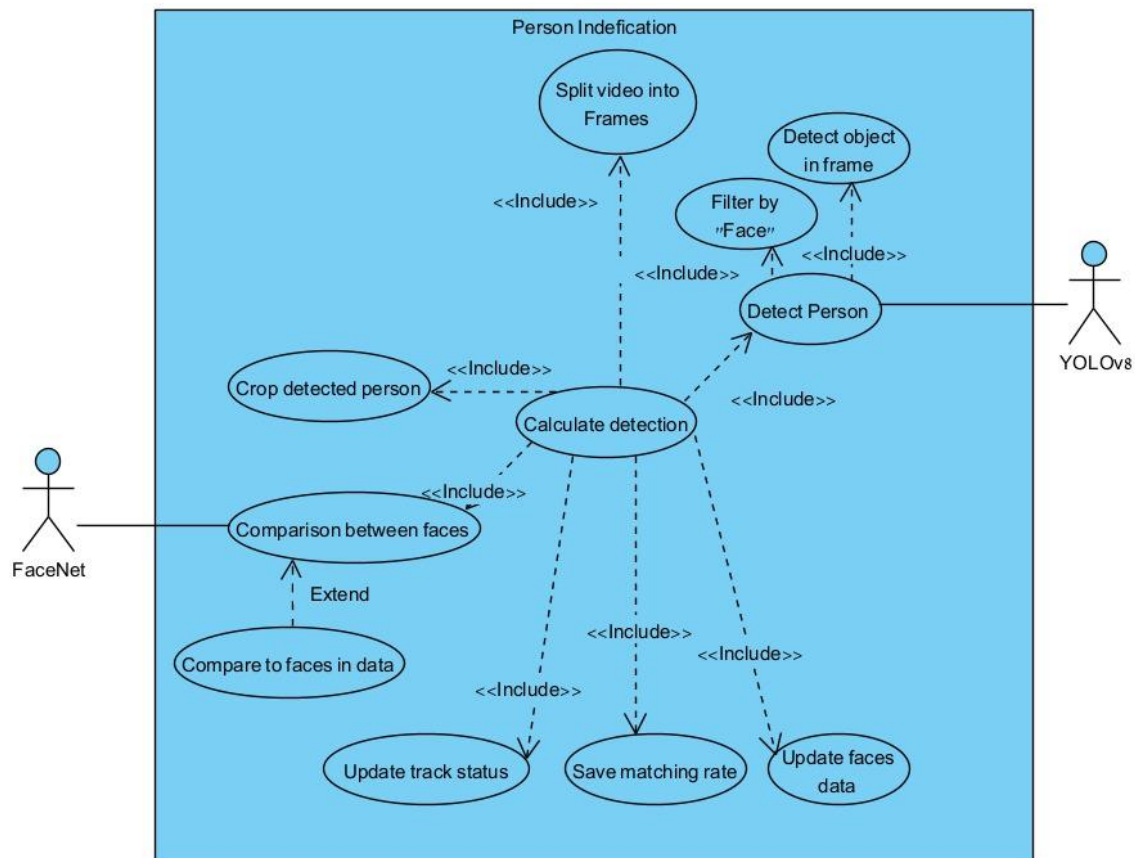


Figure 8: Person identification use case diagram

### 5.7.3 Person Tracking Use Case

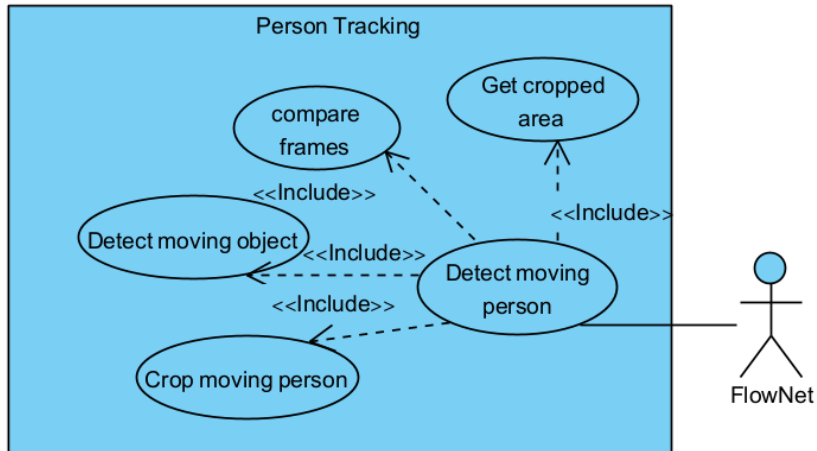


Figure 9: Person tracking use case diagram

### 5.7.4 CLIP Use Case

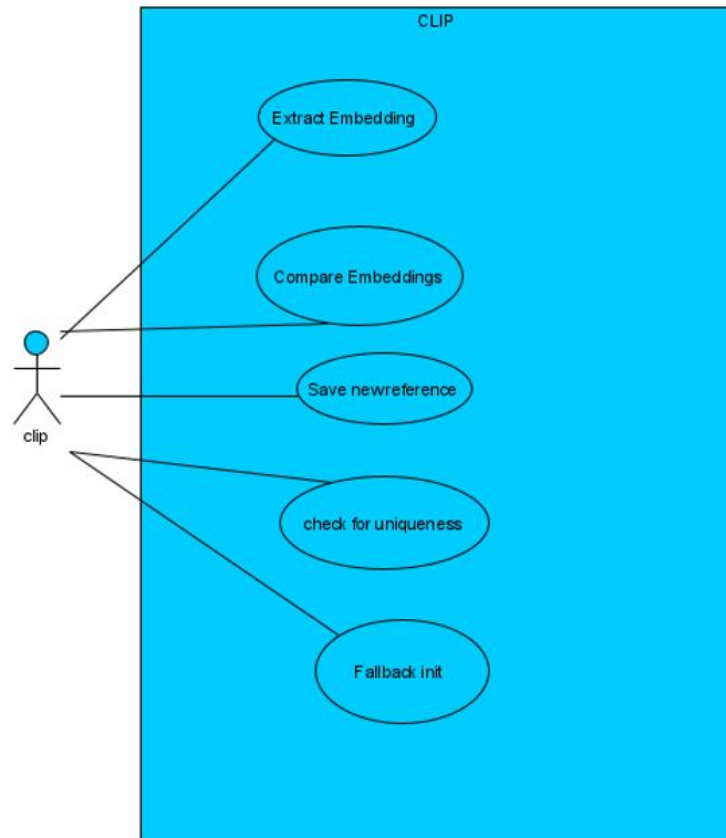


Figure 10: CLIP use case diagram

## 5.8 System Activity Diagram

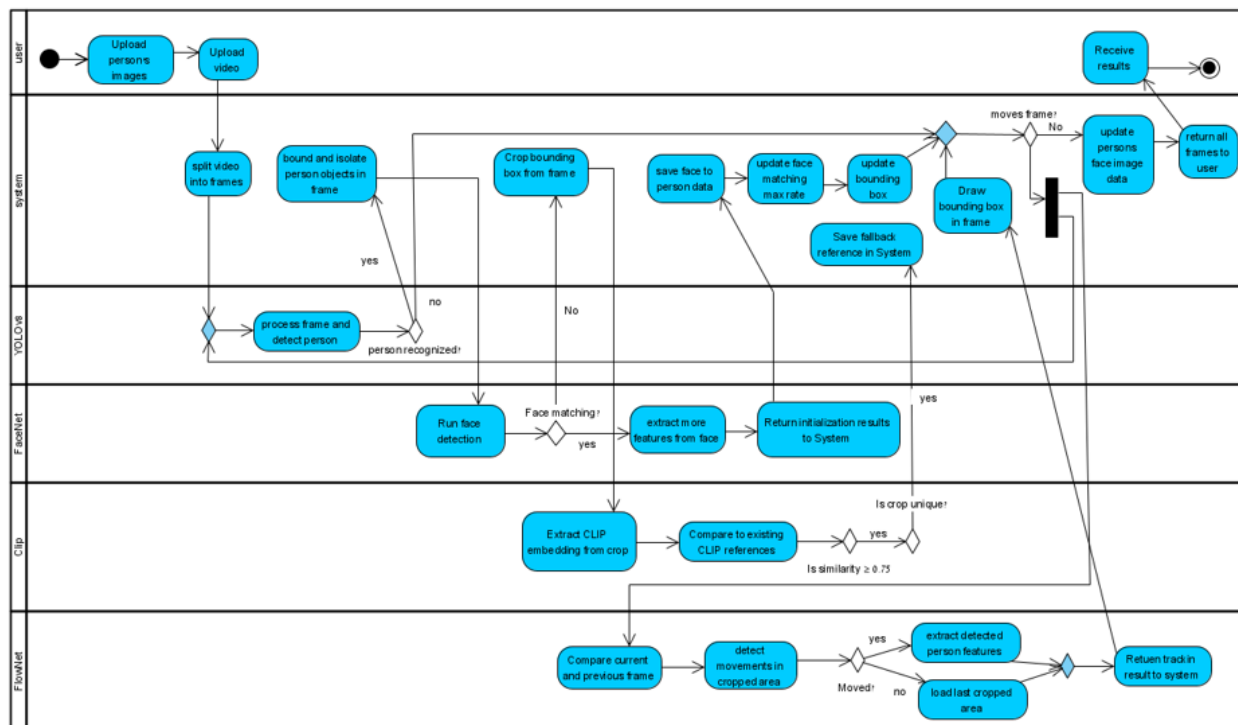


Figure 11: System activity diagram

The diagram illustrates a multi-stage system for video-based person recognition and tracking. The process starts with the user uploading face reference images and a video. The system splits the video into frames and uses YOLOv8 to detect "Person" objects. If a person is recognized, FaceNet checks if the face matches the reference. On a match, the system saves the cropped face, updates the best similarity score, and adjusts the bounding box.

If FaceNet fails, the system extracts a crop and passes it to CLIP, which compares visual similarity to stored references. If the similarity is high and the appearance is unique, it updates the CLIP reference list and stores a fallback.

FlowNet ensures continuous tracking by comparing motion between frames. If movement is detected, it adjusts the bounding box and extracts updated features. When all frames are processed, the system returns the annotated video and recognition data to the user.

## 5.9 Package Diagram

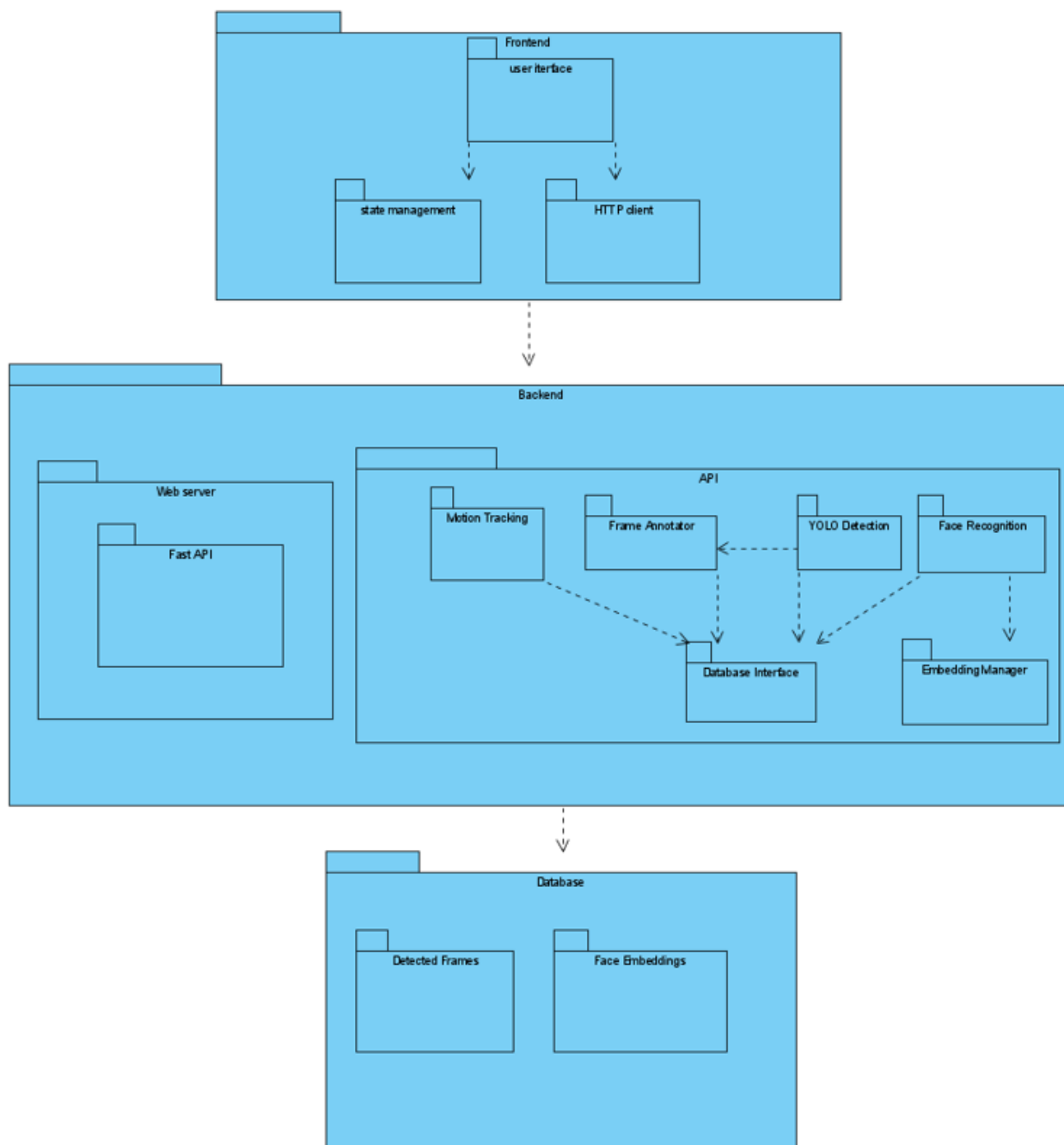


Figure 12: Package diagram

## 5.10 System Class Diagram

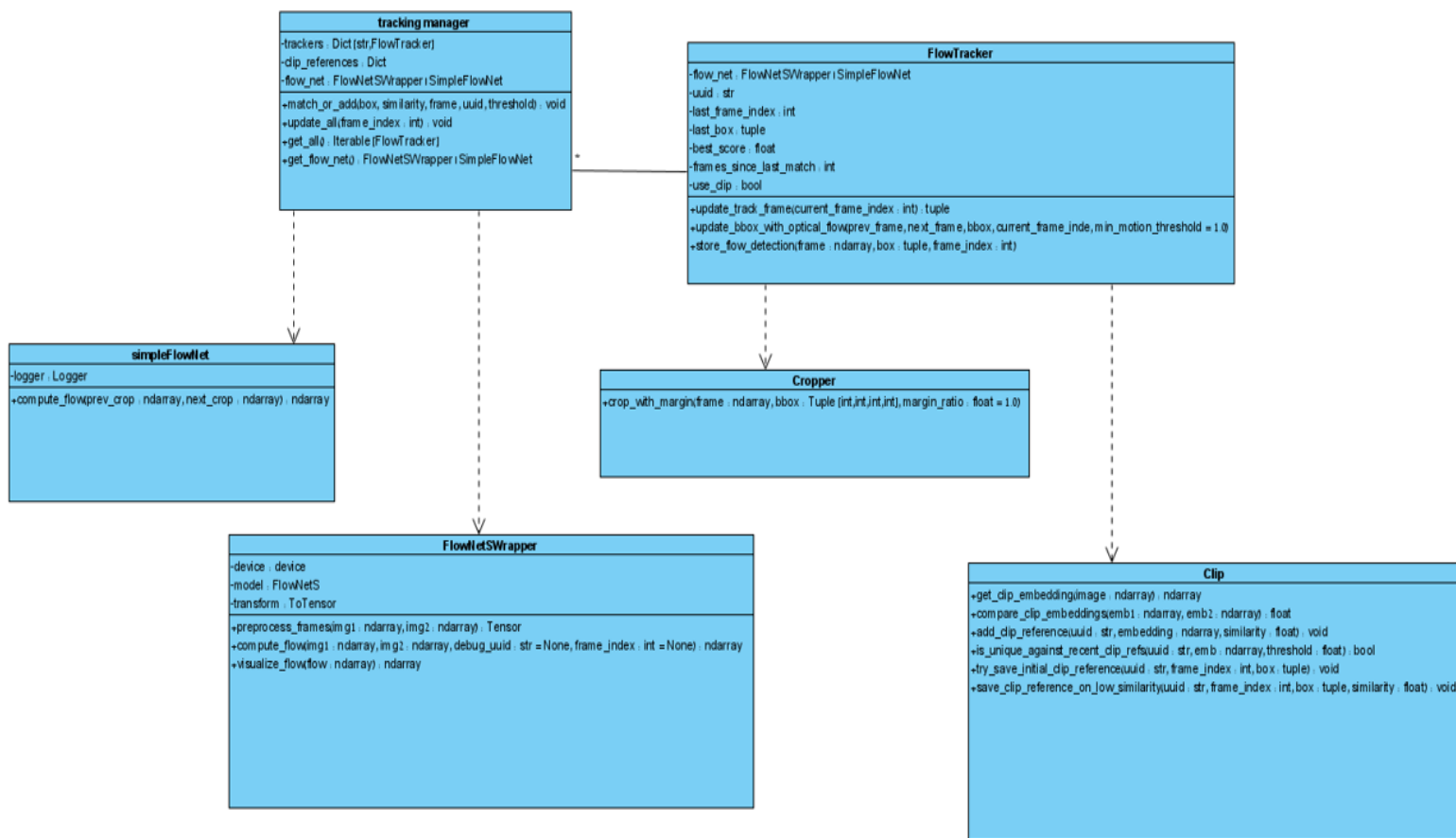


Figure 13: System class diagram

## 5.11 User Interface

### 5.11.1 Page 1: Loading Data Screen

- Fill in the person's id and personal data (optional).
- Upload a reference face images for identification.
- Upload a video file for analysis.
- Start the processing (YOLO + FaceNet + FlowNet).



FindPerson

HOMEABOUTCONTACT

1

2

3

?

Upload Photos and Video

Processing

Result

Who are we looking for?

ID \*

123000000

Name (Optional)

Leonardo Dicaprio

Age (Optional)

50

Height (Optional)

183

UPLOAD PHOTO

X

X

X

X

Where should we look?

UPLOAD VIDEO

X

FIND PERSON

Figure 14: Loading data screen

### 5.11.2 Page 2: Analyzing Screen

This page displays a processing status for a video analysis.

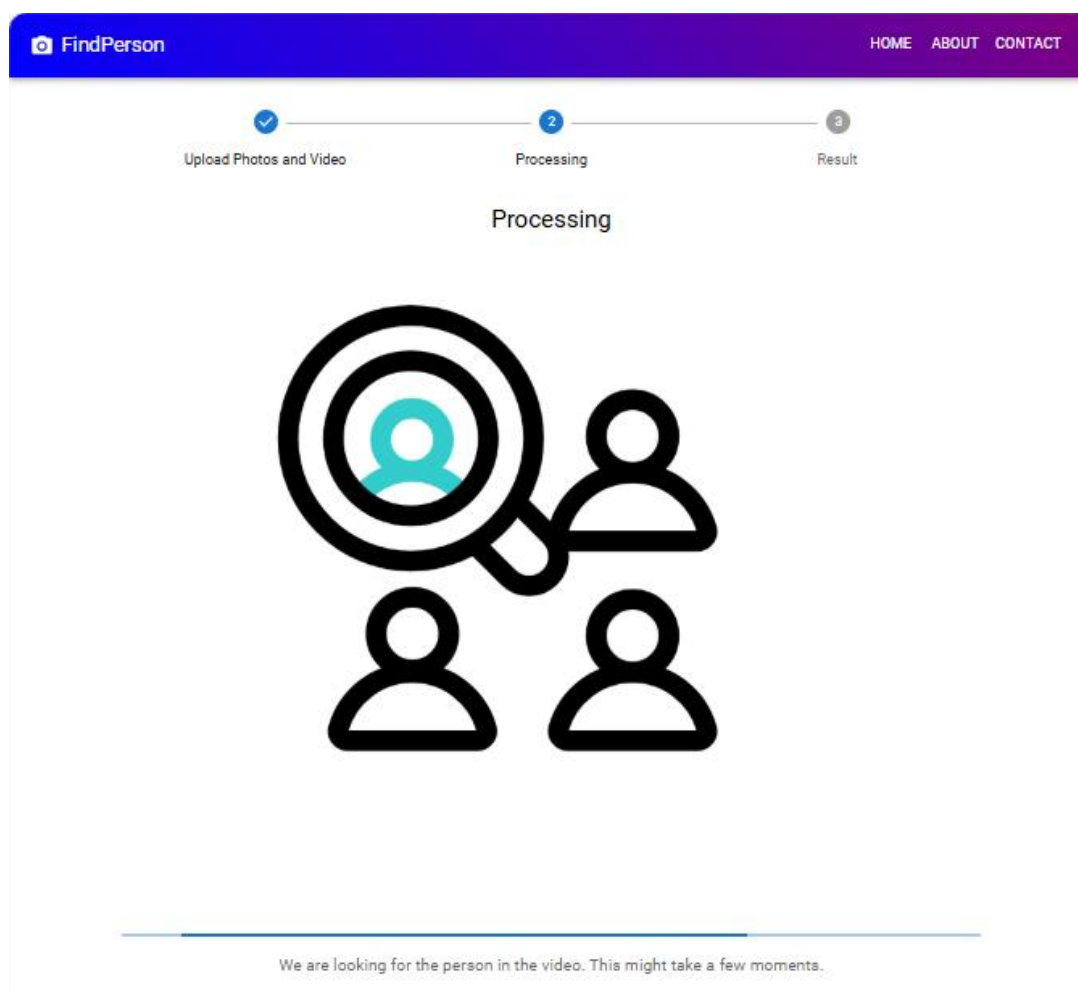


Figure 15: Analysing data screen

### 5.11.3 Page 3: Result Screen

This page displays the result of a search for a person, Showing the bounding box around the matching person in the video allowing the user to examine the character moves, and showing images of the search and the detection, along with the processing time and similarity score.





FindPerson

HOMEABOUTCONTACT

Upload Photos and Video

Processing

Result

Processed Video

FlowNet | Best: 78.23%

0:02 / 0:07

Person Details

ID: 123000000 Name: Leonardo Dicaprio Age: 50 Height: 183

Detections

Figure 16: Result screen

## 6 Testing Plan

### 6.1 Test Cases

No.	Test Case	Test Description	Preconditions	Test Steps	Expected Result	Result Pass/Fail
1	Upload Video and Image	Verify that the user can successfully upload a video and an image.	The system should be up and running.	1.Upload a video file 2.Upload an image of a person.	Both files should be successfully uploaded.	Passed - files are uploaded without errors.
2	Decompose Video into Frames	Verify that the system correctly decomposes a video into individual frames.	A valid video has been uploaded.	1. Click the "Process Video" button. 2. Check the output to verify that the video is broken into frames.	Each second of the video should result in a series of frames.	Passed - frames are generated correctly.
3	Detect Persons Using YOLO	Verify that the system detects persons in each frame using YOLO.	A valid video and image have been uploaded.	1. Process a frame through the YOLO model. 2. Check the output for bounding boxes around detected persons.	Persons should be detected with bounding boxes surrounding them.	Passed - bounding boxes are correctly generated.

4	Generate Embedding's Using FaceNet	Verify that the system generates embedding is for detected faces.	A valid frame with a detected face is processed.	1. Pass a detected face through the FaceNet model.  2. Verify the generated embedding for the face.	The system should generate a unique embedding for each detected face.	Passed - embeddings are generated.
5	Analyze Motion Using FlowNet	Verify that the system analyzes motion between consecutive frames.	The video is processed, and persons are detected.	1. Process consecutive frames using FlowNet.  2. Check if motion data) is correctly calculated.	Motion data should be generated for detected persons between frames.	Passed - motion data is generated.
6	Track Detected Persons Across Frames	Verify that the system tracks detected persons across multiple frames.	The system has detected persons in the video.	1. Track persons from one frame to another.  2. Check if the tracking is consistent across frames.	Persons should be consistently tracked across frames.	Passed - tracking is correct.
7	Performance Test for Real-Time Processing	Verify that the system processes video streams in real-time without significant delay.	A valid video and reference image are uploaded.	1. Start processing the video.  2. Measure the processing time for each frame.	The system should process each frame in real-time, with minimal latency.	Passed - the frame processing time is within acceptable limits.
8	Scalability Test	Verify that the system can handle	The system is ready to process video.	1.Upload videos of	The system should be able to handle	Passed - the performance is consistent

		videos of varying lengths without significant performance worsening.		different lengths  2. Measure the system's performance as the video length increases.	longer videos without performance worsening.	across video lengths.
9	Reliability Test for System Availability	Verify that the system is available and performs tasks with minimal downtime.	The system is running.	1. Start processing a video.  2. Check if the system remains available and functional throughout the process.	The system should remain operational with minimal downtime during processing.	Passed - the system runs without interruption.
10	Handling Occlusion	Verify that the system can handle scenarios where a person is partially occluded or blocked and can detect them once they become visible again.	The system is ready to process video frames.	1. Ensure the video contains a person who becomes partially occluded.  2. Play the video and wait for the person to reappear after the occlusion.  3. Verify if the system detects and resumes tracking the person once	The system should detect and track the person after they become visible.	Passed -the person is tracked correctly.

				fully visible again.		
11	Handling Quick Movements	Verify that the system can track a person even if they move quickly across the frame.	The system is ready, and the video is already processed.	1. Ensure the video contains a person moving quickly.  2. Play the video and check if the system can detect the person during their quick movement.  3. Verify that the system continues tracking the person without losing them.	The system should detect and track the person even during quick movements.	Passed - the person is detected and tracked.
12	Handling Low Lighting Conditions	Verify that the system can still detect and track a person even in low lighting conditions.	The system is ready, and the video is already processed.  The video contains low-light scenarios.	1. Ensure the video contains a person in low lighting.  2. Play the video and check if the system detects the person under low-light conditions.  3. Verify that the system continues	The system should detect and track the person in low lighting.	Passed - the person is detected and tracked in medium and low lighting, but not tracked well in darkness.

				tracking the person despite the low lighting.		
13	Handling Partial Occlusion by a Moving Object	Verify that the system can resume tracking a person after partial occlusion by a moving object.	<p>The system is ready, and the video is already processed.</p> <p>The video contains a person being partially occluded by a moving object (another person walking in front of them).</p>	<p>1. Ensure partial occlusion by a moving object.</p> <p>2. Play the video and check tracking after occlusion.</p>	Tracking resumes once the person is fully visible again.	Passed - tracking resumes.

Table 7: Test cases

## 7 Expected Accomplishments

### **Accurate Person Detection and Tracking:**

The system will detect people in video frames using YOLO, and even if someone is partially hidden or behind an object, it will pick them up again once they are fully visible.

### **Facial Recognition Efficiency:**

The system will match faces using FaceNet, and even if there are some challenges like lighting changes or partial obstructions, it will still identify the person correctly.

### **Motion Tracking:**

With FlowNet, the system will track the movement of people between frames. Even if their movement is blocked for a moment or they move intermittently, it will still capture their motion data accurately.

### **Real-Time Processing:**

The system will process video streams in real-time, ensuring that everything happens smoothly without any noticeable delay. Motion detection, face recognition, and person identification will all work together seamlessly as the video plays.

## 7.1 Problems

**Achieving real time performance:** Such systems require significant computational power, yet they must also maintain high speeds. Therefore, optimization must always strike a balance between efficiency and accuracy.

**Low video quality:** A major challenge in person identification is dealing with low video quality, where factors like low resolution, noise, or blurriness can make it difficult to accurately identify individuals, especially when they are partially obscured or the video lacks clarity.

**Cross domain generalization:** It has been challenging to generalize models trained on one dataset to perform well on another or in real-world scenarios, yet this is critical for achieving reliable performance across all domains.

## 7.2 Criteria for Success

**Identification Accuracy:** The software should maintain a precision rate of at least 90% when verifying individuals through videos captured in any environment, as demonstrated by routine testing and validation.

**Tracking Success:** The system should consistently track the same identified person with an accuracy rate of at least 70% across consecutive frames, as validated by ongoing testing and evaluation.

**Operational Impact:** The system should lead to a 30% decrease in response time and enhance decision-making efficiency in security and emergency situations.

**User Feedback and Satisfaction:** Structured feedback from specialized teams should reflect a satisfaction rate of at least 80%, demonstrating the system's effectiveness and value to them.

**System Reliability & Performance:** The system must achieve at least 99% uptime, ensuring stable performance even during high load conditions or when exposed to different environments.

**Adoption & Engagement:** The system will be considered successful if it gains adoption from 50% of the target user groups within six months of its application.



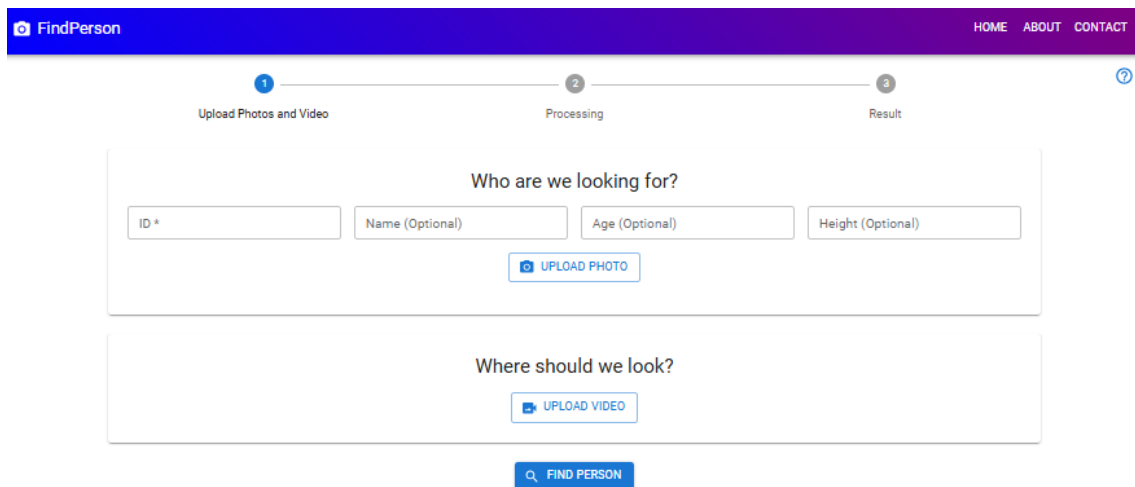
## 8 User Guide

### 8.1 Getting Started

The system allows users to locate individuals within a video using reference images. To initiate the process, users are guided through three main steps: submitting visual data (photos and video), letting the system analyze the video content, and reviewing the visual results.

#### 8.1.1 Step 1: Loading Data Screen

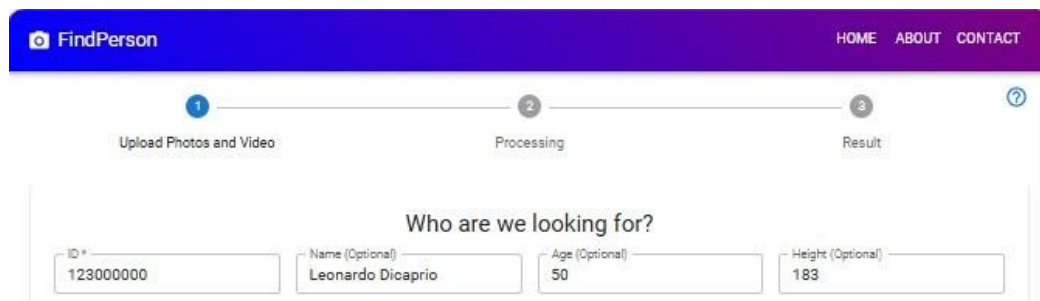
This screen represents the initial step in the person identification process.



The screenshot shows the 'FindPerson' web application interface. At the top, there is a purple header bar with the 'FindPerson' logo on the left and 'HOME ABOUT CONTACT' links on the right. Below the header, a progress bar indicates three steps: 1. Upload Photos and Video (active), 2. Processing, and 3. Result. The main content area is titled 'Who are we looking for?' and contains four input fields: 'ID \*', 'Name (Optional)', 'Age (Optional)', and 'Height (Optional)'. Below these fields is a blue button labeled 'UPLOAD PHOTO'. The second section is titled 'Where should we look?' and contains a blue button labeled 'UPLOAD VIDEO'. At the bottom of the form is a blue button labeled 'FIND PERSON'.

Figure 17: Loading data screen

1. The user begins by entering a unique ID and optional personal information such as name, age, and height. This information will be stored for the specific id along with the detections.



This screenshot shows the same 'FindPerson' web application interface as Figure 17, but with sample data entered into the input fields. The 'ID \*' field contains '123000000', the 'Name (Optional)' field contains 'Leonardo Dicaprio', the 'Age (Optional)' field contains '50', and the 'Height (Optional)' field contains '183'. The 'FIND PERSON' button is now highlighted in blue, indicating it has been clicked.

Figure 18: Loading person information into the system

- Next, the user uploads one or more reference photos of the individual he wishes to identify.



Figure 19: Loading pictures of the person to be searched in the video

- The user uploads a video in which the system will attempt to locate that person.

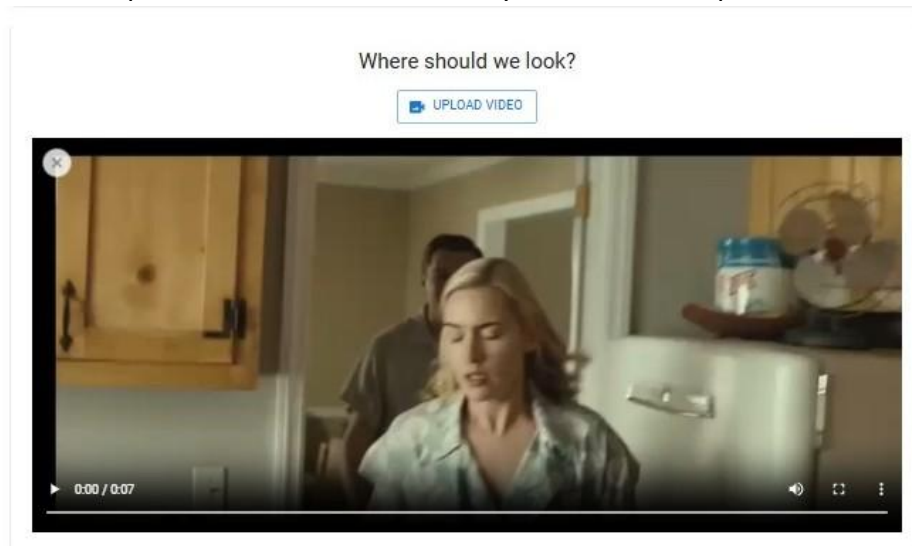


Figure 20: Loading the video to be analysed

- Once all necessary media files are uploaded, the user initiates the process by clicking the "Find Person" button. This action triggers the system's analysis phase using deep learning models for detection and recognition.



Figure 21: Starting the search process

### 8.1.2 Step 2: Processing Videos

Once the photos and video have been uploaded, and the “Find Person” button pressed, the system automatically begins analyzing the video frame by frame in order to identify the target person throughout its duration.

The process starts with YOLOv8, which detects all person objects in each frame. Each detected person is then passed to FaceNet, which performs facial recognition by comparing the detected face with the reference images provided by the user. If a match is found, the system registers the detection and saves the relevant data.

Alongside this, the system uses FlowNet, a motion tracking module that continuously analyzes frame-to-frame movement in order to refine and update the position of detected individuals. FlowNet enhances tracking performance by following the person's motion over time.

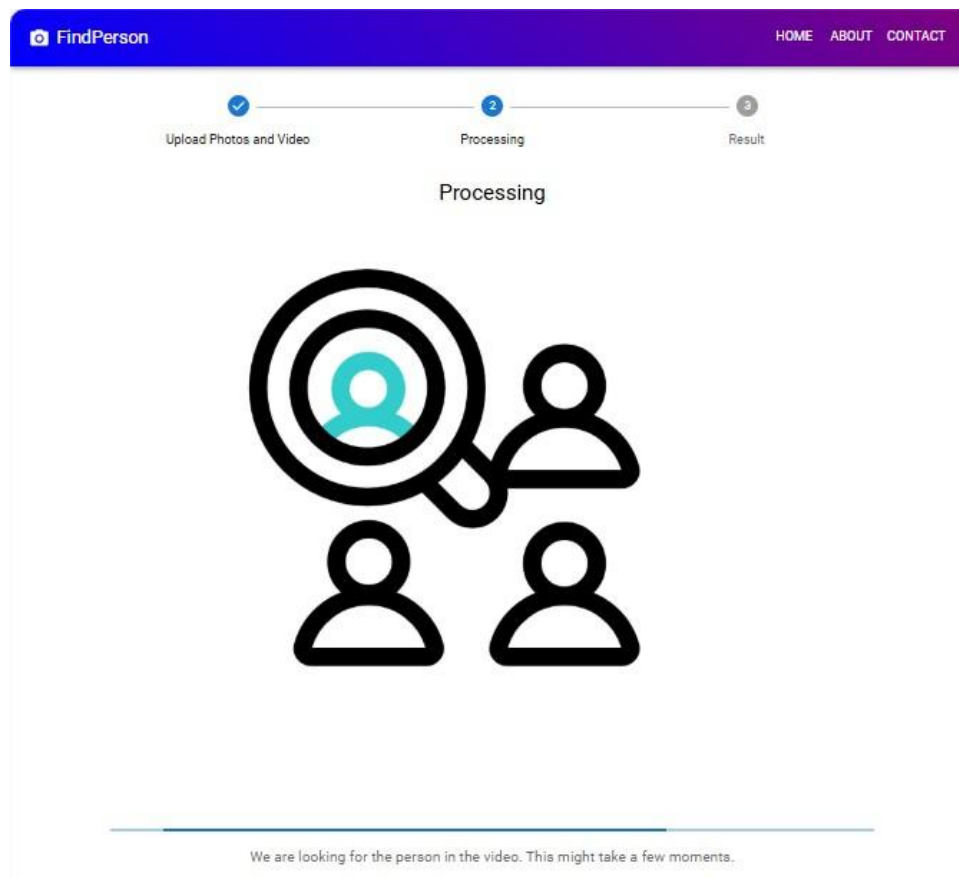


Figure 22: Analysing the data screen

### 8.1.3 Step 3: Viewing Results

After the video has been fully processed, the system presents the results on a dedicated screen. The top of the screen displays the processed video, where the identified person is highlighted with a bounding box and a confidence score. The label also indicates which module detected the person such as FaceNet or FlowNet—to provide transparency regarding the tracking source.

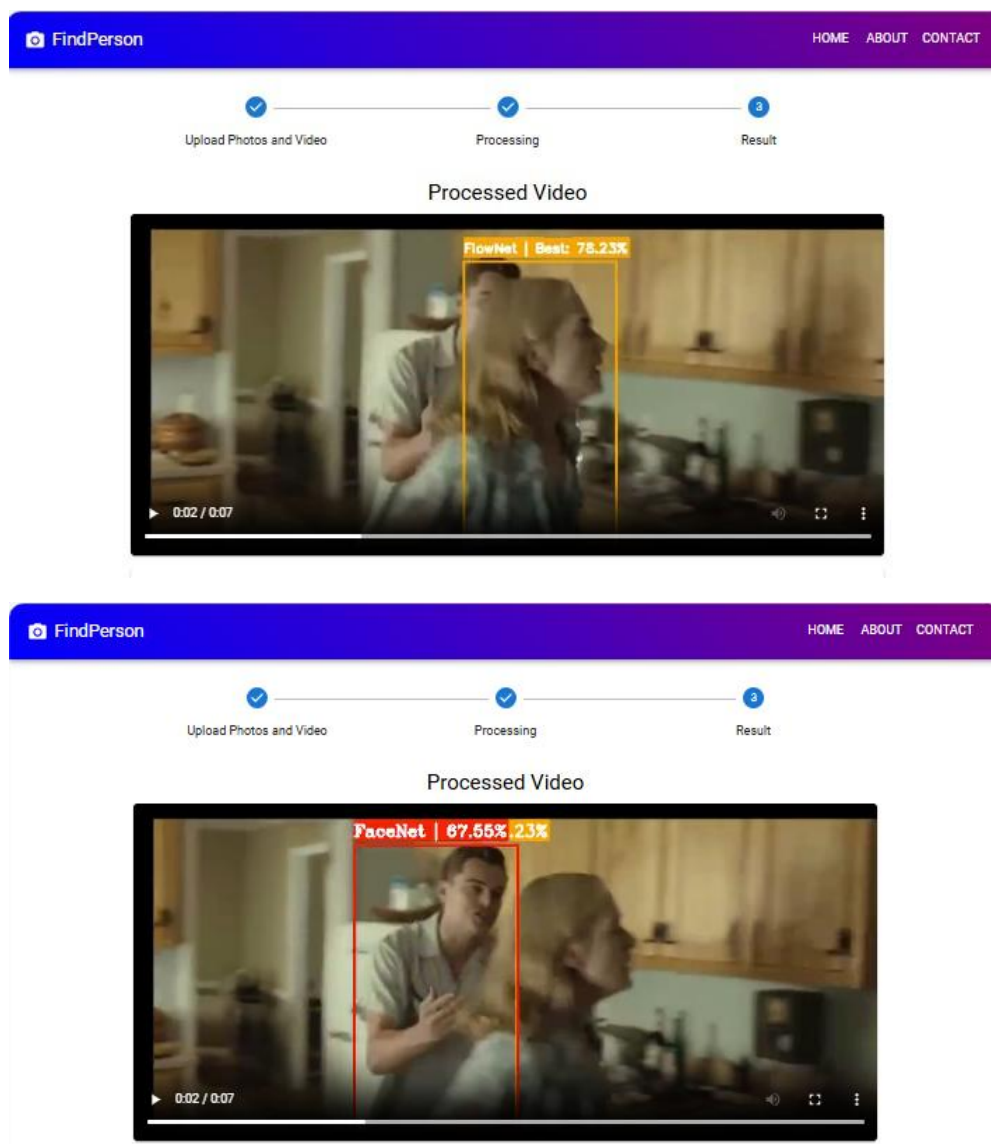


Figure 23: The processed video view with FaceNet detection and FlowNet tracking bounding boxes

Below the video, the system shows the person's details, including their ID, name, age, and height as provided during upload.



Figure 24: The searched person information

At the bottom, the Detections section displays a horizontal carousel of all frames in which the person was successfully identified. The large image at the center shows the currently focused detection, while smaller preview frames on either side allow smooth navigation through earlier and later appearances. Selecting any of these previews shifts the main view to the corresponding frame in the video. Users can scroll through the previews to examine detections more closely.



Figure 25: Display of frames crops with the detected person

This interactive results screen makes it easy to visually verify the system's performance and review where and how often the target person appeared throughout the video.

## 8.2 Features Overview

The system integrates several advanced modules that work together to enable accurate, efficient, and continuous tracking of individuals in video footage. These modules include YOLO-based object detection, FaceNet facial recognition, and FlowNet optical flow tracking. Together, they contribute to a strong and highly adaptable identification framework, which allows not only to find a person within a video but to track and follow him through the video.

### 8.2.1 YOLO-based Object Detection

YOLO (You Only Look Once) is an advanced object detection algorithm that enables real-time identification of individuals in video streams. Its architecture processes images rapidly by dividing them into grids and simultaneously predicting bounding boxes and class probabilities. This parallel approach significantly reduces detection latency compared to traditional methods. The result is a highly efficient and accurate system capable of identifying multiple objects in dynamic environments, making it well suited for security, surveillance, and tracking applications.

### 8.2.2 FaceNet-based Person Recognition

After a person is detected by YOLO, the FaceNet module compares the detected face against the uploaded reference photos. It generates a face embedding and calculates similarity scores to identify if the face matches the target person. This step ensures precision in identity verification at early stages.

### 8.2.3 FlowNet-based Flow Tracking

FlowNet is a key module for maintaining continuous tracking across frames, even when YOLO or FaceNet are unable to detect the person. It estimates motion vectors between frames and shifts the bounding box accordingly, keeping the person in view over time. This approach ensures smooth tracking throughout the video sequence.

## 8.3 System Errors and Fixes

### 8.3.1 Common Problems and Fixes

While the system usually works smoothly, sometimes users may face problems when uploading photos or videos, processing the video, or viewing the results. This section explains common issues and gives easy steps to fix them, so you can keep using the system without trouble.

#### **Frame Processing Problems**

If some video frames are skipped or if detection boxes are missing, it might be due to limited system resources or an unsupported video format. Try closing other programs or using a shorter video clip.

#### **Low Detection Rates**

When the person is not detected in some parts of the video, check that the reference images are sharp and the person's face is fully visible. Try to match lighting and face angles between the images and the video for better accuracy. Also make sure there are no other people in the pictures uploaded as it will cause search mistakes.

#### **Web Browser Problems**

If uploads or playback fails, make sure you are using a modern browser like Chrome or Edge. You can also refresh the page, clear the cache, or try private mode to resolve common browser related issues.

## 9 Maintenance Guide

### 9.1 System Architecture

The system is built with a modular architecture that supports smooth communication between the frontend and backend. It is designed to be scalable, reliable, and secure. The system is organized into three main layers: the frontend, the backend, and the database.

#### 9.1.1 Frontend Interaction

The frontend remains a lightweight web-based interface built using React, HTML, CSS and JavaScript. Users can upload photos and videos, initiate analysis, and view results through an intuitive dashboard on the web. The frontend communicates with the backend via RESTful APIs and handles responses asynchronously using AJAX, allowing for real-time feedback without page reloads.

#### 9.1.2 Backend

The backend is implemented using FastAPI and coordinates the entire recognition pipeline. It handles requests from the frontend, manages user sessions, and connects with the database to store and retrieve detection results, user profiles, and tracking metadata. It also runs key components such as:

- YOLOv8 for detecting "person" objects in frames.
- FaceNet for verifying identities based on facial embeddings.
- FlowNet for continuous person tracking when direct detection is unavailable or insufficient.

All processing is performed asynchronously to maximize responsiveness and system efficiency.

#### 9.1.3 Database Management

The system relies on a local MongoDB Compass database to handle data securely and effectively. It stores essential information such as user profiles, identification records, detected frames, and related metadata, and maintains data integrity while supporting efficient querying. To sustain high performance and rapid access to data, the system performs routine maintenance tasks including indexing, optimization, and scheduled backups.



#### 9.1.4 Dependencies

The system depends on several essential components to operate efficiently, including:

- FastAPI: For building the backend API.
- React: For developing the frontend.
- TensorFlow and Keras: For integrating machine learning models for object detection and facial recognition.
- MongoDB Compass: For managing the local database securely.
- Gunicorn and Uvicorn: For serving the FastAPI application in production.
- Torch: For running FlowNet models.
- OpenCV: For image manipulation and optical flow.
- Transformers: For running the CLIP model for person validation by visual features beyond facial recognition.

### 9.2 Scheduled Operational Tasks

To keep the system operating efficiently and securely, administrators should perform routine maintenance tasks. These procedures are essential for preventing performance issues, ensuring data reliability, and maintaining overall system stability.

#### 9.2.1 Managing Python Virtual Environment

##### 1. Creating the Virtual Environment:

- Use Python's venv module to create a virtual environment within the server directory. This approach isolates dependencies from the global Python installation, avoiding conflicts between projects.
- Activate the environment and install dependencies using the "requirements.txt" file.

##### 2. Updating the Environment:

- Periodically run environment update to ensure everything within it is up to date.
- periodically run "pip install -upgrade -r requirements.txt" to ensure all libraries remain up to date and secure.

#### 9.2.2 Data Backup Procedures

Regular backups are critical to protect against unexpected data loss caused by hardware failures, software bugs, or cyberattacks, so the following steps are recommended:

- **Automated Backups:**  
Use Mongo DB's backup utilities to schedule backups daily, weekly, or monthly.
- **Offsite Storage:**  
Store backup copies in a secure offsite location or a cloud-based solution to ensure disaster and bugs recovery.
- **Integrity Testing:**  
Periodically verify backup files by restoring a sample to confirm that data is recoverable and consistent.

### 9.2.3 Log Management

System logs play a vital role in monitoring activity, identifying issues, and ensuring accountability. To ensure visibility and support troubleshooting, some log management recommended:

- **Centralized Logging:**  
Aggregate logs from all modules (e.g., FlowNet, FaceNet, YOLO) into a unified logging system for easy analysis.
- **Scheduled Review:**  
Regularly examine logs for errors, anomalies, or security warnings, to enhance system performance.
- **Retention Policy:**  
Define how long logs should be stored, balancing storage efficiency with organizational or legal requirements.

### 9.2.4 Security Maintenance

Protecting the system from unauthorized access or malicious activity is a top priority. So, it's recommended to maintain the following:

- **Update Software Frequently:**  
Apply patches and upgrades to the operating system, backend services (FastAPI, MongoDB), and machine learning libraries (Torch, TensorFlow) to close security gaps.

- **Manage Access Controls:**  
Review user accounts regularly and limit access to sensitive data and administrative functions.
- **Run Security Audits:**  
Perform periodic audits to evaluate system defenses, including vulnerability scans and simulated attacks (penetration testing).

## 9.3 Future Improvements:

### 9.3.1 Monitoring and Alerts

As the system scales, incorporating live monitoring and alert mechanisms becomes essential. These additions can significantly improve system stability, help detect failures early, and reduce downtime.

#### 9.3.1.1 Why Monitoring Matters?

Monitoring tools allow developers and system administrators to track metrics such as memory usage, processing time per frame, and error logs. This is especially important when dealing with resource-heavy modules like YOLO and FlowNet, which rely on real-time performance.

By continuously tracking these metrics, the system can:

- Detect slowdowns or performance drops.
- Warn about failing components.
- Trigger actions automatically when thresholds are crossed.

#### 9.3.1.2 Implementing Alerts

Setting up alert mechanisms enables quick response to critical issues. For example:

- If FlowNet fails to update bounding boxes for several frames, an alert can notify the team.
- If CPU usage exceeds a safe level, an automated message can be sent.

#### 9.3.1.3 Responding to Alerts

Efficient alert handling minimizes system disruptions. The following guidelines are recommended:

- **Prioritize Alerts:** Not all alerts are critical. High-severity alerts should be addressed immediately; others can be scheduled for review.

- **Document Responses:** Maintain a log of alerts and how they were resolved to help improve future response strategies.
- **Review and Optimize:** Periodically review the alerting system to adjust thresholds, reduce noise, and ensure relevance based on system usage patterns.

By implementing real-time monitoring and structured alert workflows, the system can maintain high reliability and performance, even under demanding workloads.

### 9.3.2 Multi-Person Detection and Tracking

As the system evolves, enabling robust handling of multiple individuals in a single video becomes increasingly important. Supporting simultaneous detection, identification, and tracking of several people enhances system scalability, real-world applicability, and analytical capabilities.

#### 9.3.2.1 Why Multi-ID Handling Matters?

Real-world scenarios often involve crowded environments—such as events, protests, or public transport—where multiple people of interest appear in the same video. Properly managing multiple UUIDs per video session enables the system to:

- Track different individuals in parallel without confusion.
- Maintain separate identity logs, embeddings, and crops for each detected person.
- Compare and analyze interactions or movement patterns across multiple subjects.

This functionality is especially critical in use cases such as surveillance, missing person search, or forensic video analysis.

#### 9.3.2.2 System Behaviour with Multiple IDs

To handle several identities effectively, the system must:

- Assign a unique UUID for each detected person, even within the same frame.
- Manage concurrent FlowNet trackers, each maintaining separate optical flow logic and bounding box histories.

- Validate each identity independently using FaceNet and CLIP, allowing confidence-based filtering and correction when overlaps or occlusions occur.

For example:

- If two people enter the frame simultaneously, the system initializes two FlowTrackers and keeps their crop regions, similarity scores, and embeddings isolated.
- If a person temporarily disappears and reappears, the system re-identifies them.

### 9.3.2.3 Optimization by Multi-Person Tracking

Managing multiple tracked identities increases system complexity, so it is recommended to:

- **Prioritize Resource Allocation:** Dynamically manage GPU/memory usage by allocating model inference intelligently across IDs, especially for FlowNetS and FaceNet.
- **Frame Synchronization:** Ensure consistent frame indexing and cropping logic so that tracked identities remain stable across time.
- **Conflict Resolution:** Implement similarity-based matching thresholds to prevent switching or merging IDs when two people have similar appearance or movement.

By supporting multiple simultaneous identities in the same video stream, the system becomes significantly more powerful and versatile—capable of operating effectively in real-world, multi-person scenarios without losing track accuracy or database integrity.

## 9.4 Troubleshooting and Debugging

Maintaining the reliability and performance of the person identification system requires effective troubleshooting and debugging. When issues arise, a structured approach helps identify root causes, resolve them efficiently, and recover from failures.

### Diagnosing Issues:

Start by gathering information and replicating the problem. Review system logs for error messages and consult documentation for known issues and solutions.

### Using Debugging Tools:

Leverage development tools like IDEs (e.g., PyCharm, VS Code) for step-by-step code

analysis, use profilers to detect performance bottlenecks, and network analyzers like Wireshark for communication issues.

#### **System Recovery:**

Have a solid recovery plan that includes regular data backups, rollback procedures for faulty updates, and comprehensive disaster recovery steps to minimize downtime.

## 9.5 Documentation and Updates

Maintaining up-to-date documentation is essential for effective system maintenance, team collaboration, and smooth operation of the person identification system. As the system evolves, so must its documentation to ensure accuracy, usability, and compliance.

#### **Why Updated Documentation Matters:**

- **Operational Consistency:** Provides clear, unified procedures for all users, reducing errors and confusion.
- **Faster Onboarding:** Helps new team members learn the system quickly and efficiently.
- **Simplified Troubleshooting:** Enables faster issue resolution by referencing known problems and solutions.
- **Regulatory Compliance:** Ensures the system adheres to legal and organizational standards.

#### **Strategies for Effective Knowledge Sharing:**

- **Version Control:** Track changes and maintain historical records of documentation.
- **Scheduled Reviews:** Regularly update documentation to reflect system changes.
- **Collaborative Tools:** Use platforms like Confluence or Google Docs to enable team-wide contributions.
- **Centralized Repositories:** Store all documentation in a single, accessible location for easy reference.

By maintaining clear, up-to-date documentation and enabling effective knowledge transfer, organizations can boost system reliability, support team collaboration, and ensure long-term efficiency and success.

## 10 References

- [1] Ron Harel & Itay Benjamin (2024). AI-Driven Person Recognition and Identification in Videos.
- [2] Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 815-823).
- [3] Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D., & Brox, T. (2015). FlowNet: Learning Optical Flow with Convolutional Networks. IEEE International Conference on Computer Vision (ICCV).
- [4] Horn, B. K. P., & Schunck, B. G. (1981). *Determining optical flow*. Artificial Intelligence, 17(1-3), 185–203.
- [5] Lucas, B. D., & Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI).
- [6] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
- [7] Cao, Q., Shen, L., Xie, W., Parkhi, O. M., & Zisserman, A. (2018, May). Vggface2: A dataset for recognising faces across pose and age. In 2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018) (pp. 67-74). IEEE.
- [8] Wang, M., & Deng, W. (2021). Deep face recognition: A survey. Neurocomputing, 429, 215-244.
- [9] Jeremy Cohen. (2023 September). Computer Vision and Deep Learning: From Image to Video Analysis.
- [10] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25

- [11] Wang, P., Li, W., Gao, Z., Tang, C., & Ogunbona, P. O. (2018). Depth pooling based large-scale 3-d action recognition with convolutional neural networks. *IEEE Transactions on Multimedia*, 20(5), 1051-1061.
- [12] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).
- [13] FlowNet Pseudocode: [GitHub - ClementPinard/FlowNetPytorch: Pytorch implementation of FlowNet by Dosovitskiy et al.](#)
- [14] FlowNetCorr Pseudocode: [FlowNetPytorch/models/FlowNetC.py at master · ClementPinard/FlowNetPytorch · GitHub](#)
- [15] FlowNetSimple Pseudocode: [FlowNetPytorch/models/FlowNetS.py at master · ClementPinard/FlowNetPytorch · GitHub](#)
- [16] YOLO Pseudocode: [YOLO Pseudocode.py · GitHub](#)
- [17] Python: [Welcome to Python.org](#)
- [18] React: [React](#)
- [19] Cocomdataset official [COCO - Common Objects in Context](#)
- [20] FiftyOne Integrations COCO Integration: [COCO Integration — FiftyOne 1.3.0 documentation](#)
- [21] images of person detection : <https://www.ntu.edu.sg/rose/research-focus/deep-learning-video-analytics/person-re-identification>
- [22] Clip: [CLIP](#)
- [23] Clip models: [openai/clip-vit-base-patch32 · Hugging Face](#)
- [24] Vision Transformer (ViT) architecture: [https://www.geeksforgeeks.org/deep-learning/vision-transformer-vit-architecture/?utm\\_source=chatgpt.com](https://www.geeksforgeeks.org/deep-learning/vision-transformer-vit-architecture/?utm_source=chatgpt.com)