University of Macedonia
Dertment of Applied Informatics
postgraduate programme in artificial intelligence and data analytics

Addressing computer vision challenges using an active learning framework

Tzogka Christina

A.M: aid20002

Supervisor:
Ioannis Refanidis

June 2021

Submission Date, June 2021.

Submitted in partial fulfilment of the requirements for THE POSTGRADUATE PROGRAMME IN ARTIFICIAL INTELLIGENCE AND DATA ANALYTICS

Postgraduate Programme in Artificial Intelligence and Data Analytics
Department of Applied Informatics
University of Macedonia

ADDRESSING COMPUTER VISION CHALLENGES USING AN ACTIVE LEARNING FRAMEWORK

Tzogka Christina

Previous academic degrees - A Bachelor of Computer Science, AUTH, 2019

Supervisor:

Ioannis Refanidis

Approved by the three-member Examination Committee on 16/02/2021:
Ioannis Refanidis, Nikolaos Samaras, Dimitrios Hristou-Varsakelis

Department of Applied Informatics

Postgraduate Programme in Artificial Intelligence and Data Analytics

June 2021

Abstract

Machine Learning applications has transformed everyday life as well as industry by providing new successful opportunities in healthcare, transportation, banking, security, media monitoring and more. Computer Vision is an application of Machine Learning that recently has done a lot of progress, particularly in Face Recognition and Object Detection systems. These systems require large data sets to be trained with. Nevertheless, the available data sets contain large amounts of unlabelled samples.

Active Learning is an innovative field that addresses the challenge of labelling large sets of unlabelled samples by leveraging only a small amount of manually labelled data. An efficient way of labelling a small amount of training data is utilizing user-friendly annotation tools. The latter allow playing a whole video streaming and capturing the desired entities. This interactive method could be very efficient as well as time-saving in comparison to traditional data collection methods.

This thesis builds on state-of-the-art Face Recognition and Object Detection models, by implementing optimization methods that enhance the recognition accuracy. Further training is being introduced by making use of a robust Active Learning framework that results in creating extended data sets. Finally, our thesis proposes an integrated system, which involves effective techniques of associating face and object identification information, in order to extract as much knowledge as possible from a video streaming, in real-time.

Index Terms: Face Recognition, Object Detection, Active Learning, Deep Learning, Data Set

Acknowledgments

Firstly, I would like to thank professor Ioannis Refanidis for accepting to be my supervisors for this thesis as well as for being very punctual and supportive.

I am also thankful to the DataScouting company members for lending the necessary equipment, in order to conduct my experiments as well as for inspiring me to come up with a plan for an interesting thesis that would contribute to academia.

Finally, I would like to thank my family for supporting me throughout this work in the same way they are supporting me, since the day I started implementing my life's goals.

Christina Tzogka

Table of Contents

List of Figures

List of Tables

1

Chapter 1: Introduction

Everyday life as well as industry and business functions have been rapidly transformed due to Machine Learning applications. Machine Learning proposes a variety of techniques for extracting knowledge from data that can be rendered into meaningful goals. Computer Vision [1], as a Machine Learning application, has introduced new successful opportunities in an abundance of domains, including healthcare, transportation, banking, media monitoring, statistical data analysis, scientific research and more. Such applications have already been incorporated in smartphone, websites, cyber security, etc.

Computer Vision systems are trying to imitate the complex structure of the human eye, which can see and understand the environment or a digital image. Although this is a pretty challenging job, there has been a lot of progress particularly in Face Recognition [2] and Object Detection [3] systems. Except for commercial Face Recognition and Object Detection systems, there are quite a few open-source builds that achieve state-of-the-art performance, making use of Deep Learning.

Convolutional Neural Networks have been used in nearly all of the top performing methods on the Face Recognition and Object Detection systems. It is widely known that these systems require large amounts of data to be trained with, in order to perform satisfyingly. Nevertheless, in most cases the available data sets include an amount of unlabelled training samples that is too large to be manually labeled. Active Learning [4] [5] techniques address this challenge by leveraging only a small amount of manually labelled data, in order to train good supervised models. This approach could perform better than traditional methods even with an importantly lower amount of training data.

An efficient way of labelling a small amount of training data is utilizing user-friendly annotation tools that could annotate a whole video streaming and export the labelled data in the appropriate format so that they could fit on systems' demands. This method of constructing data sets could be extremely useful and time-saving, since often it is tough

enough or even impossible to find an existing data set that includes the desired classes. Quite often, a time-consuming research on available labelled data sets may turn out to be unsuccessful. For instance, when a Machine Leaning engineer aims to identify a certain face or object, then he starts seeking for the corresponding entity in the freely available data sets. These data sets usually are huge enough and could be extremely hard to save them and extract the desired entity (that is, an object or a face). At the worst case, the available data sets do not even include the corresponding entity.

Consequently, the only way to obtain the desired data is to collect them by utilizing annotation tools. These annotation tools enable a continuous interaction between the system and the Machine Leaning engineer as he can easily decide which classes will enhance the current data set, while watching a video streaming and capturing the desired entities. A handy scenario is playing an annotated video, where the already "known" entities are enclosed in visible bounding boxes. As a result, the Machine Learning engineer could decide which entities would enhance the current data set with new classes or additional samples for the existing classes.

The goal of this thesis is to: a.) provide a comprehensive study on the state-of-the-art Face Recognition and Object Detection systems implementation and further training, b.) propose a robust Active Learning framework of creating extended data sets, c) introduce a sophisticated optimization method that enhances the recognition accuracy, and d.) provide efficient techniques of associating Face Recognition and Object Detection outputs through the integrated system, in order to extract as much knowledge as possible from an input video, in real-time.

Furthermore, our integrated system consists of both Face recognition and Object Detection models, while we introduce the implementation of a detailed metadata repository that includes information for the input's features, such as the duration, the resolution, the frame rate as well as its contents, which correspond to the entities (faces and objects) that the models have detected. The metadata repository as it turned out is extremely functional, since it enables the system's user to access the whole extracted information as well as the further associations between the Face Recognition and Object Detection individual outputs. Also, we enable the metadata repository to prevent multiple execution of an input video with the same model's variables (e.g number of classes in the data set, thresholds etc).

The structure of this report is organized in the following chapters:

- Chapter 2 introduces Computer Vision and makes a review of state-of-the-art Face Recognition and Object Detection models. In this section we also describe the avail-

able options for Face Recognition and Object Detection as well as the key points that led us to the final decision of using the proposed systems.

- Chapter 3 introduces Active Learning methods and enumerates some of most popular Machine Learning fields, where Active Learning could be extremely useful. Meanwhile, this section includes a brief presentation of our Active Learning framework and its benefits as regards to data set enhancement purposes.

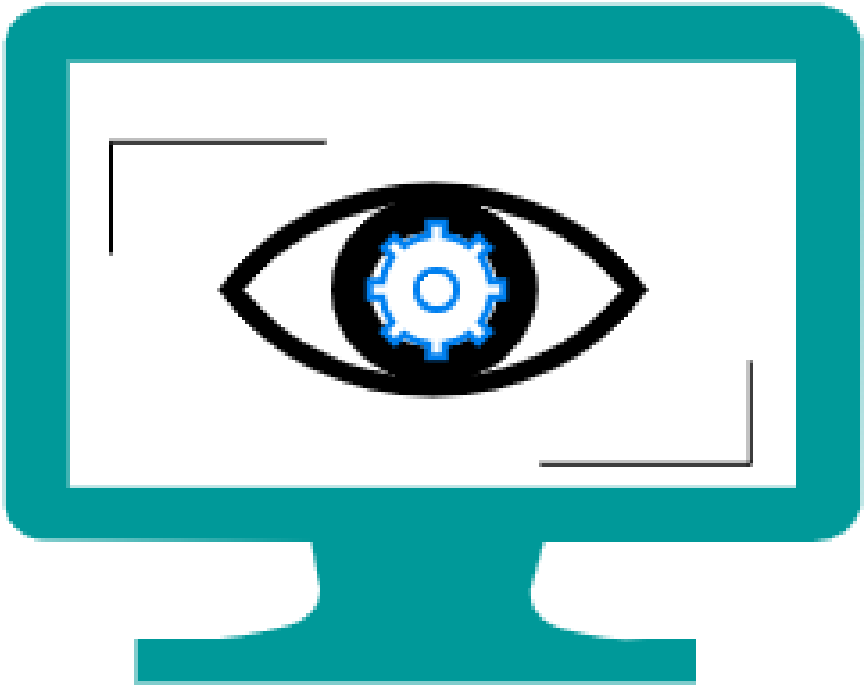
- Chapter 4 describes the training data demands for the proposed Face Recognition and Object Detection system as well as the data preparation process for both systems. At the end of this chapter we will have built the initial data set that is going to be used for further training and evaluation.

- In Chapter 5 we introduce our Active Learning framework that leads to automated data set enhancement. Also, we present our integrated system that consists of the implemented models, which apply optimized Face Recognition and Object Detection tasks (in parallel) as well as a metadata repository, which stores all the meaningful information as regards to the input video features and the entities that appear though the frames.

- In Chapter 6 we conclude this thesis by summarizing our findings and discussing further research challenges.

2

Chapter 2: Computer Vision Background



Computer Vision is broadly called a sub-field of Machine Learning and Artificial Intelligence (Figure 2.1), while it also makes use of general learning algorithms. Computer Vision applications develop techniques, which try to imitate the complex structure of the human eye, in order to enable the computers to "see" and understand the content of images and videos. However, this is a pretty tough job. That's why Computer Vision problem remains largely an unsolved problem.

It is worth noting that Computer Vision is distinct from Image Processing and aims to understanding the content of digital images (e.g. entities, text description, etc). In fact, the most popular Computer Vision applications try to recognize things in images. The following list provides some of the high-level problems, where we have seen success with

Computer Vision:

- Face Recognition
- Optical Character Recognition (OCR)
- Fingerprint Recognition and Biometrics
- Motion Capture (mocap)
- 3D Model Building (photogrammetry)
- Medical Imaging
- Object Classification/Segmentation/Recognition/Detection etc.

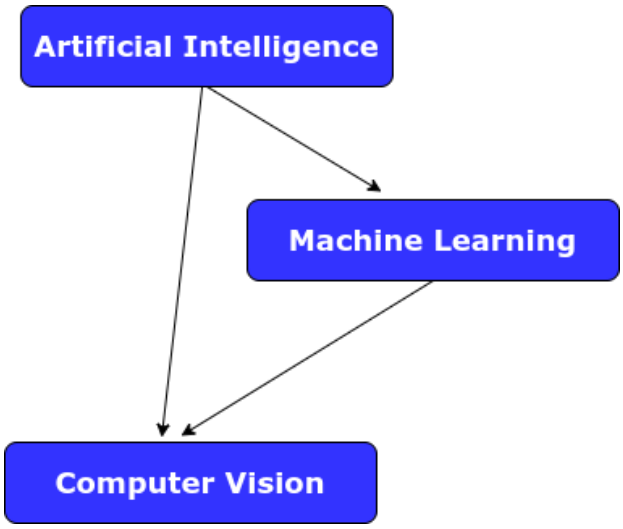


Figure 2.1: *Computer Vision as a sub-field of Machine Learning and Artificial Intelligence [1]*

2.1 Face Recognition System

2.1.1 State-of-the-art Review

Face Recognition is a Computer Vision application, where recently has been a lot of progress. The very latest results (2018) of tests conducted by the US Homeland Security Science and Technology Directorate, provide an indication of the best Face Recognition software available on the market. In 2014, Facebook announced DeepFace, which can determine whether two images belong to the same person, with accuracy 97.25%. Taking the same test, humans answer correctly slightly better than the Facebook program, with accuracy 97.53%.

In 2015, Google announced FaceNet [6], which achieved a new record accuracy of 99.63% on the widely used "Labeled Faces in the Wild" (LFW) [7] dataset. At the same time, OpenFace [8] was released. OpenFace is an open-source implementation, inspired by FaceNet.

Convolutional Neural Networks (CNNs) have been used in nearly all of the top performing methods on the LFW data set. Dlib [9] is an open-source Face Recognition built, using Deep Learning and achieving state-of-the-art performance. Dlib's network architecture is based on ResNet-34 [10] from the "Deep Residual Learning for Image Recognition" paper. The model that is responsible for actually quantifying each face in an image achieves an accuracy of 99.38% (on LFW) and is a part of the OpenFace project.

The approach that we introduce in this thesis includes the open-source Dlib library.

2.1.2 Dlib

The full Face Recognition workflow consists of the following processes:

1. Face detection
2. Face pre-processing
3. Feature extraction
4. Face recognition

Face Detection

The first step in our workflow is face detection, so that the system could be able to locate the faces in an image before passing them to the next step. It is worth mentioning that face detection picks only the faces within the image that are in focus. Face detection is considered to be composed of four main step:

- ◇ Converting image to black and white, since there is no need for colors within the detector.
- ◇ For every single pixel, there is a darkness comparison with all the pixels that directly surround it. The ultimate goal of this step is drawing an arrow showing in which direction the image is getting darker.
- ◇ Once all pixels have been replaced by an arrow-gradient (see Figure 2.2), the flow from light to dark has been figured out across the entire image. As a result, both dark and bright images end up with the same representation, which makes the Face Recognition problem a lot easier to solve.

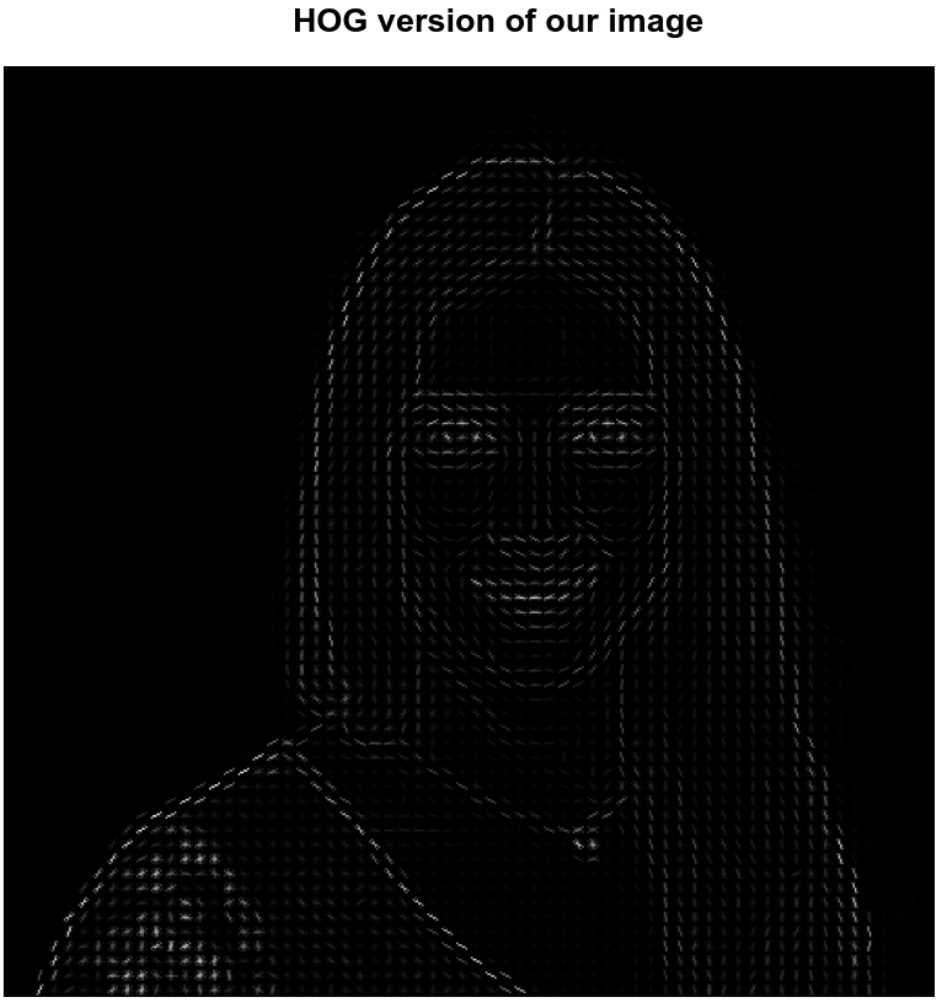


Figure 2.2: *Example of an image on the last step of face detection, where all pixels have been replaced by arrows-gradients. [11]*

Face Pre-processing

After detecting a face within the image, a pre-processing is being applied:

- ◇ All faces are being warped so that their eyes and lips lie always in the same direction. This step is necessary for faces comparison purposes. In fact, a Machine Learning algorithm should be trained, in order to be able to estimate the face landmarks (i.e 68 specific points that exist on every face).

- ◇ The next step is applying rotation, scaling, transformations etc, so that the eyes and the mouth within the image to be centered. This step enhances the accuracy particularly when the face is turned.

Feature Extraction

At this end, it is necessary to extract a few basic measurements from each face. For instance, a measurement could correspond to the ears' size, the eyes' distance, the nose's length etc. These measurements enable comparisons between different faces. A CNN is being trained, in order to be able to generate 128 measurements (encodings) for each face. During the training process the CNN is looking at 3 face images at a time (see Figure 2.3):

- A training sample of a known person.
- Another sample of the same known person.
- A training sample of a different person.

The Machine Learning algorithm that generates the measurements for each of the above three images configures the CNN, in order to ensure that the measurements of the first and second faces differ slightly, while the measurements of the second and third are further apart. The CNN is being trained with millions of images of thousands of different people. Finally, it learns to generate reliable encodings for each person. Furthermore, any person within different pictures should correspond to approximately the same encodings. It is worth mentioning that this step is being executed only once. OpenFace has already trained and published the pre-trained CNN.

Face Recognition

Lastly, a Machine Learning classification algorithm (e.g Support Vector Machines, K-Nearest Neighbors) is being used, in order to find the person in the data set, who has the smallest encodings difference from the test sample.

2.2 Object Detection

2.2.1 State-of-the-art Review

Object Detection is another Computer Vision application which is widely applied. The available systems have recently made great progress resulting in high detection accuracy score.

ImageAI supports a list of state-of-the-art Machine Learning models, trained on the ImageNet-1000 [12] data set, for Image Prediction [13] and Object Detection. The most popular models supported by ImageAI for Image Prediction and Object Detection are RetinaNet and YOLOv3 [14], respectively. YOLOv3 has been trained on COCO [15] data

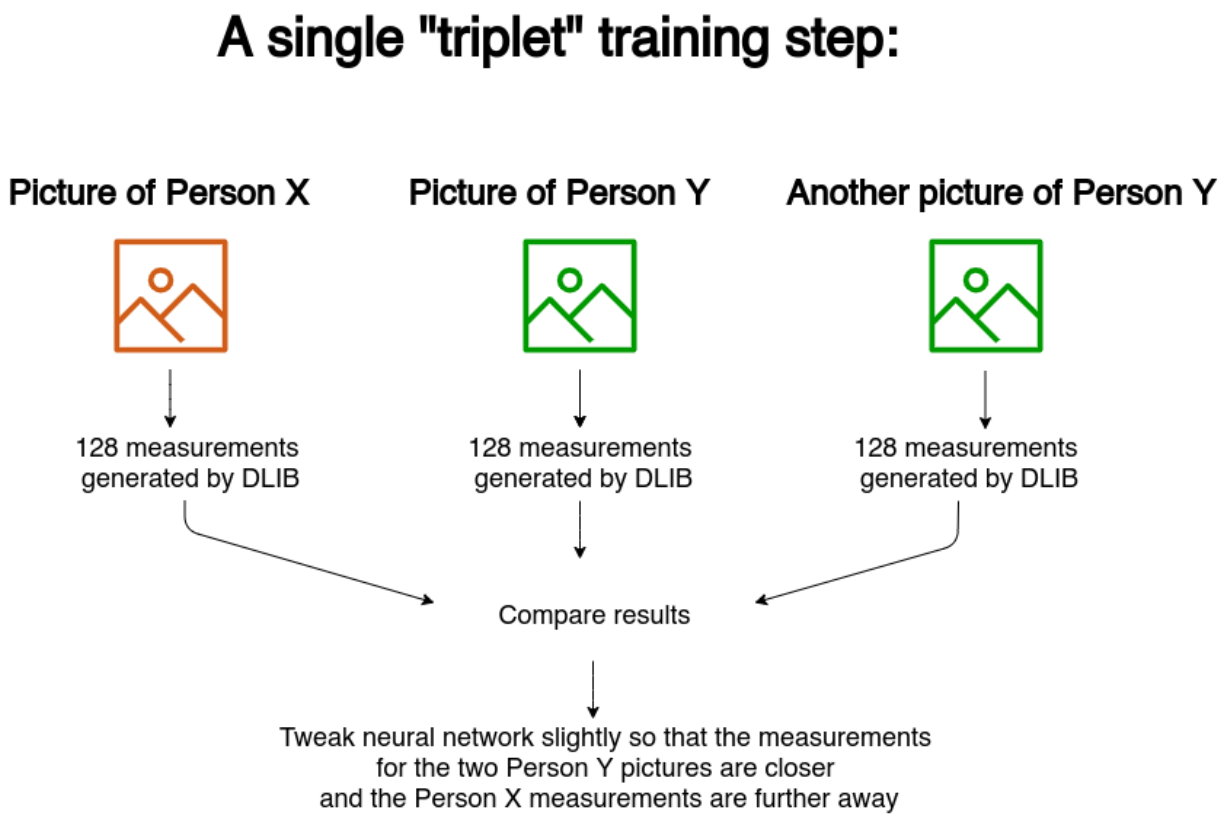


Figure 2.3: *Face recognition via Deep Learning involves a "triplet training step", which consists of 3 unique face images. [11]*

set and is able to recognize 80 different objects in images and videos.

The CNN used in YOLOv3 is inspired by the GoogleNet [16] model. The YOLOv3 model has a 63.7% mean average precision (mAP) score over the 2007 PASCAL VOC [17] data set, while it allows real-time predictions. As it has been proven, YOLOv3 is a robust option for Object Detection, since it could be 4 times faster and certainly more convenient to use in comparison to other well-performed models such as Faster R-CNN [18] and Mask R-CNN [19], which are not implementable for real-time.

2.2.2 YOLOv3

The approach that we introduce in this thesis includes the published YOLOv3 pre-trained. YOLOv3 is very fast and well-performed due to the technique that it follows. More precisely, it detects objects by forwarding the whole image through the network only once. On the contrary, other well-performed detection frameworks (e.g Faster R-CNN and SSD) look at different parts of the same image multiple times at different scales. This approach could be extremely slow and inefficient.

Object detector is a combination of an object locator and an object recognizer. The following technique (non-maximum suppression) represents YOLOv3 Object Detection workflow in an input image:

- ◇ Firstly, the input image is being divided into a 13×13 grid of cells. The size of the cells vary depending on the input's size. For example for a 416×416 input the cell size is 32×32. Every cell predicts a number of bounding boxes in the input.

- ◇ Each predicted bounding box is followed by the confidence value that it actually encloses an object as well as the probability of the enclosed object corresponds to a known class (from the data set).

- ◇ Confidence value is being used for bounding boxes elimination. Hence, a bounding box with extremely low confidence as well as a bounding box that encloses the same object as another bounding box with higher confidence, will be eliminated.

YOLOv3 system predicts the bounding boxes (i.e 4 coordinates for each bounding box) as well as the confidence score of each bounding box. The confidence score is being estimated by the function of logistic regression. When an input image is being processed by the detection system, the latter applies the following sub-processes:

1. it resizes the input image to 448 x 448,

2. it runs a single convolution network on the input image and

3. thresholds the detections by the model's confidence scores

The system's detection model can be interpreted as a regression problem. More precisely, the image is divided into an SxS grid. Each cell in the grid is followed by B bounding boxes with their confidence scores, which are being utilized, in order to reject the bounding boxes with low confidence. The latter probably do not correspond to ground-truth bounding boxes that enclose an object class. For the remaining bounding boxes (i.e high confidence score) the model predicts C class probabilities.

The classes of a bounding box are being predicted by the system with multi-label classification using independent logistic classifiers, in order to appropriately model the data. On the contrary, using softmax for complex data sets that usually involve overlapping labels, appoints the assumption that there is a single class corresponding to each bounding box. Often, this approach is invalid.

YOLOv3 predicts boxes at 3 different scales, using feature pyramid networks. The feature extractor uses the darknet-53 network (53 convolutional layers), which has been proven more powerful and faster than earlier networks. The YOLOv3 mAP score in COCO dataset is equivalent to the latest state-of-the-art models mAP score. From the evaluation phase arises that the YOLOv3 worst performance is observed on medium and big object detections.

The results of a comprehensive comparison to different Object Detection models, in terms of architecture, system performance and implementation styles, clearly has showed that YOLOv3 responds better in the most challenging problem, which is the real-time prediction.

3

Chapter 3: Active Learning

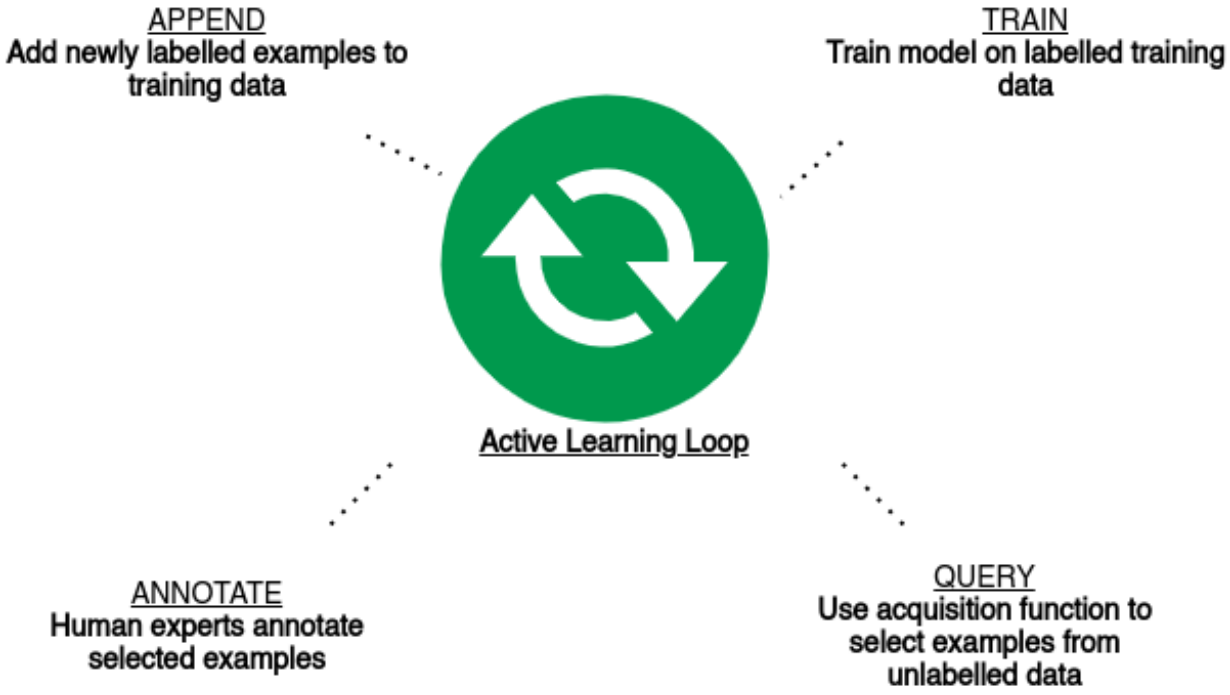


Figure 3.1: *Active Learning loop [22]*

Computer Vision problems require large data sets. Nevertheless, these data sets usually include a great amount of unlabeled data, while Machine Learning Engineers have access to more data than they are capable of analyzing.

Active Learning is a field of Machine Learning that addresses the challenge of training good supervised models with huge data sets by leveraging only a small amount of manually labelled data. In particular, the Active Learning techniques involve an algorithm, which selects to label a subset of the total unlabelled samples that are included in the whole data set. The main idea in Active Learning is based on a learning algorithm that chooses the data it wants to be trained on. This approach could perform better than traditional

methods even with an importantly lower amount of training data.

Traditional methods involve gathering large amounts of data that have been sampled randomly (that is, underlying distribution), in order to train a model. This kind of learning (that is, passive learning) could be really time-consuming, particularly if it invokes collection of labelled data. To sum up, Active Learning could result in higher accuracy, while it demands less data annotation than traditional methods.

We are going to represent the most common Active Learning technique in the consecutive sequential steps:

1. *Data collection:* Firstly, we should ensure that the data set that we are gathering is representative of the true distribution of the data.

2. Data split: Then we split the data set into a small subset of labelled samples (seed) and a larger subset of unlabelled samples.

3. *Data labelling:* Next we label the seed data. There two scenarios for creating the seed. The most popular scenario that researcher follow is collecting a fully labelled data set and making use of a smaller amount of data for seed. The second scenario is utilizing an expert (e.g Oracle) to label the seed.

4. *Model Training:* After the data split, we train our Machine Learning algorithm on seed. Typically, the algorithm is trained using cross-validation. As soon as the training process is over, the trained model is responsible for giving probabilistic response to whether a testing sample has a particular label.

5. *Unlabelled Data Selection:* Once the trained model is ready, we need to determine the type of Active Learning scenario we would use (e.g Membership Query Synthesis, Stream-Based Selective Sampling or Pool-Based sampling). For instance we could use the pool-based sampling with a batch size of 2, which means that we have to select only 2 unlabelled samples that are going to be added in the labelled data set. Usually, the samples with the highest confidence score are being selected.

6. We repeat steps 2 and 3 until a stopping criteria occurs (e.g a number of iterations, a threshold concerning the performance improvement etc.)

In conclusion, Active Learning could be a type of semi-supervised learning, where models are being trained with both labeled and unlabeled data. The challenge behind semi-supervised learning is determining the small labelled set that could result in the same accuracy or better than a fully labeled training data. In particular, Active Learning techniques

involve training a Machine Learning algorithm, in order to identify the most beneficial labels that are going to enhance the training data. As regards to the training process within an Active Learning technique, as it emerges, it involves dynamic and incremental data labelling.

3.1 Active Learning into Computer Vision

As we have already mentioned, Active Learning is still being heavily researched. In this thesis we introduce the efficient usage of Active Learning in Computer Vision problems, such as Face Recognition and Object Detection. More precisely, we investigated Active Learning techniques, in order to implement less time-consuming methods for data set construction purposes.

Is is well known that both Face Recognition and Object Detection systems require large data sets, in order to perform satisfyingly. For this reason, we propose a method of enhancing the data sets with more training samples by making use of Active Learning techniques as well as an user-friendly annotation tool. The latter is highly recommended for the manual labelling of a small amount of training data (that is, the aforementioned seed).

This initial data set is going to train the Computer Vision systems, so that they could be able to make accurate predictions for unlabelled testing samples. More precisely, we are going to train the Face Recognition model with the initial data set that consists of labelled samples. The data set contraction process will be presented in the next chapter.

After the first attempt of training the Face Recognition model we could perform predictions and identifications of the "known" faces, which appear in the input videos that undergo the integrated system. The output of the system's predictions includes the actual appearances (i.e the coordinates of each face region per frame) of the "known" faces through the video's frames. As a result, the current data set could be enhanced with more training samples for the existing classes that have been detected.

It is worth mentioning that that we have incorporated a Machine Learning algorithm that builds on Support Vector Machines, so that we could choose the best (face) samples for enhancing the data set. A handy scenario is isolating the new training samples (i.e those provided by the system's output) that are following by high confidence scores, in order to ensure that they would not harm the quality of the data set. Additionally, the Machine Learning engineer could access the new training samples that correspond to high

confidence values and reject those that do not present clear face landmarks. Thus, our proposed method includes the human-in-the-loop in order to provide meaningful guidance to the Machine Learning algorithm when needed.

Furthermore, except for the identified "known" faces, we have made use of all the "unknown" faces that have appeared in input videos but have not been identified, due to their absence from the current data set. In fact, our Active Learning framework leverages these faces so that once they enter the data set's classes (i.e they are inserted by the Machine Learning engineer) we could automatically access the existing labelled samples. The latter have already been generated while input videos undergo our integrated system. As it emerges, our Active Learning framework facilitates the data collection procedure for a new face that is entering the data set.

In the same way, after the custom training of the Object Detection model we could enhance the current data set with more training samples for the existing classes that have been detected thought the input's frames. The Object Detection output also involves confidence score for the detected objects. Consequently, we could isolate the predictions with the highest confidence scores for training data enhancement purposes.

In conclusion, our integrated system incorporates Active Learning techniques, which leverage the output of the Computer Vision models that we have implemented, in order to provide a large amount of labelled training data. Meanwhile, it enables the samples selection by utilizing the predicted confidence scores that follow every detected face/object.

Finally, we should highlight the significant role of the Machine Learning engineer in data set contraction processes. We should not forget that we have to deal with Computer Vision challenges, where the training samples correspond to images. Every image in the data set should be representative of its content (that is, faces or objects).

Actually, there are a few key points (we are going to present them in the next chapter) that the Machine Learning engineer should takes into account during data collection, in order to ensure satisfying recognition results. A traditional method (i.e passive learning) may probably harm the data set by inserting samples that have negative effect on recognition accuracy. Figure 3.2 shows a representative diagram of the Active Learning technique that we have integrated for the purposes of this thesis.

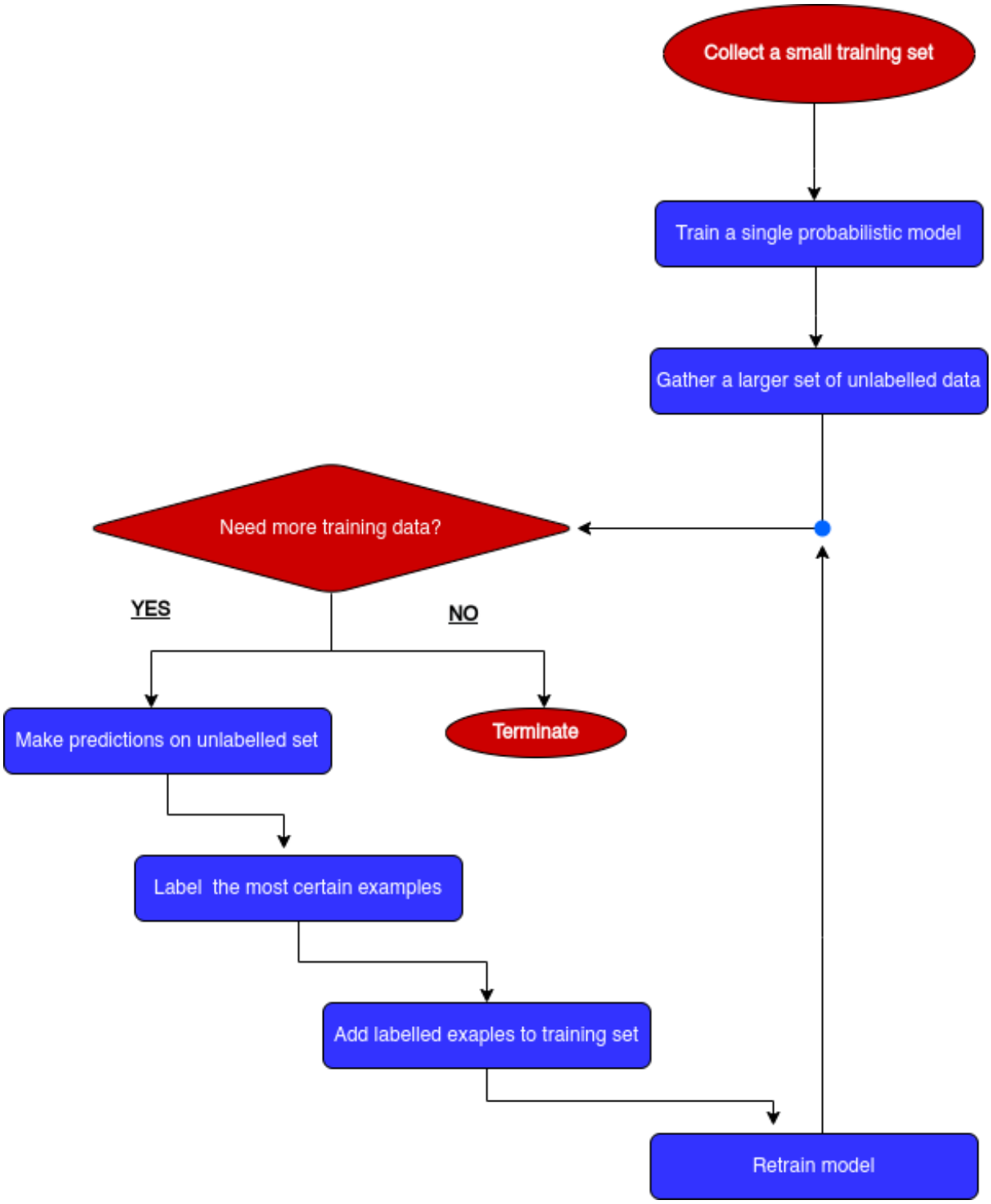


Figure 3.2: *Diagram that represents approximately the Active Learning framework of the integrated system. [5]*

3.2 Research into Active Learning

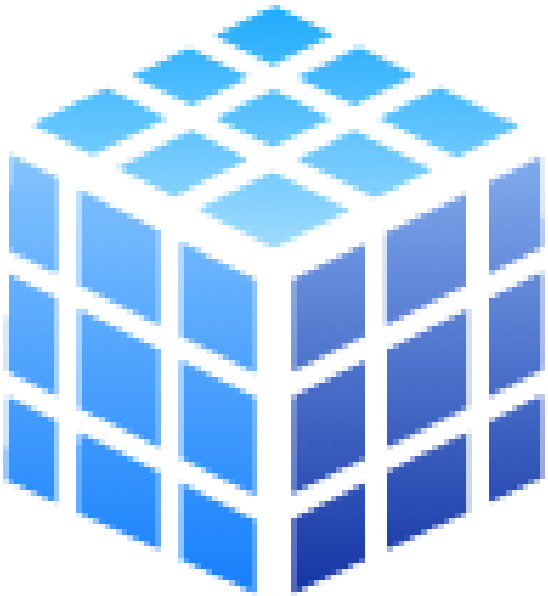
Natural Language Processing (NLP) is an example of Active Learning most popular areas. NLP applications (e.g Name Entity Recognition - NER) require large amount of labelled

data. Nevertheless, there are only a few freely available and labelled data set. Hence, Active Learning is a useful tool for the experts that could enhance the amount of the required accurately labelled data. Furthermore, Active Learning techniques can be applied to many Speech Recognition tasks as well as information retrieval tasks.

There are still many research challenges concerning Active learning. Researchers are trying to use Deep Learning algorithms (e.g CNNs, LSTMS) instead of the Machine Learning algorithms, in an attempt to improve their efficiency. There is also an increasing interest into Generative Adversarial Networks (GANs) as well as into Deep Reinforcement Learning.

4

Chapter 4: Data Set Construction



The construction of the data set is a crucial step for any Machine Learning related task. The implemented systems that we introduce in this thesis have different needs concerning the training samples. The manual construction of a small amount of labelled data as well as the utilization of Active Learning techniques is highly recommended, in order to get good results as well as avoiding the time-consuming construction of heavy data sets.

4.1 Annotation Tool

For data labelling purposes we decided to use an open-source video annotation tool, OpenCV-Video-Label [1], so that we could address both systems requirements. OpenCV-Video-Label tool supports two object tracking algorithms (Re3 [23] and CMT [24]). More precisely, it enables playing a video streaming, defining a bounding box, where a class

[1] https://github.com/natdebru/OpenCV-Video-Label

(face/object) appears and tracking the entity over multiple frames (see Figure 4.1). Each bounding box refers to a class name that is inserted by the user. The tool provides multiple output formats, based on the system's training demands. Our implementation supports two scenarios:

1. Face Recognition scenario corresponds to the cropped images that include the face regions.

2. Object Detection scenario corresponds to the original images followed by xml/txt files with the precise coordinates of objects' bounding boxes and class names.

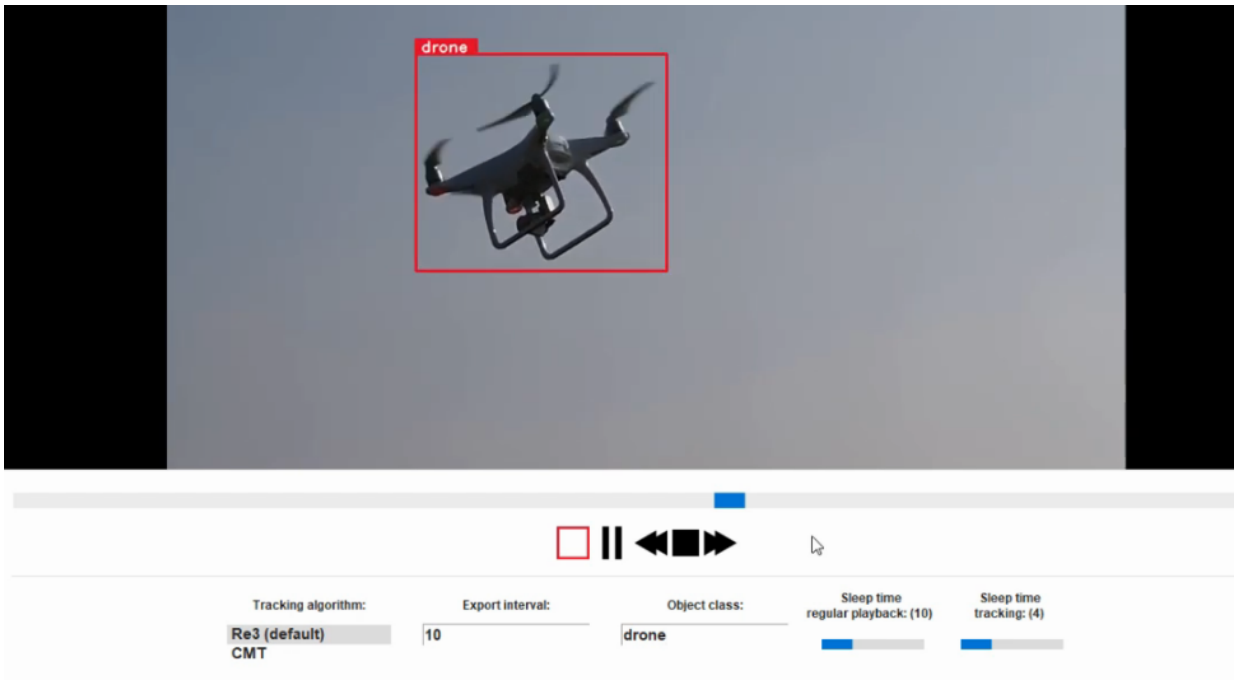


Figure 4.1: *Example of object tracking via the environment of the OpenCV-Video-Label annotation tool.*

4.2 Face Recognition Data

A Face Recognition system requires images of faces, where the face landmarks [25] are visible. To ensure that the accuracy is not reduced due to training samples issues, the following key points have been noted:

- ◇ The total number of image samples per person (class) in the data set is suggested to be higher that 50.

- ◇ Every image should contain a unique face. Hence, images with two or more faces should be avoided.

- ◇ The face within the image should be facing the camera, with both eyes visible. Thus, it is preferable to avoid samples with high percentage of hidden face landmarks (e.g., profile view).

- ◇ The dimensions of the face's bounding box should be at least 130x130px, since image resolution is a determining factor in recognition results.

As we have already mentioned, in order to initialize our experiments we built a small data set consisting of 10 classes. During data collection we made use of freely available video streaming, where the desired classes have appeared, as well as the OpenCV-Video-Label annotation tool. The average number of samples per class was approximately 65 items, while the whole data set consisted of almost 665 image samples.

After the data collection process further pre-processing (alignment) on training samples is highly recommended by literature, in order to achieve the optimal recognition accuracy. Face alignment [26] includes rotation, scaling, translating, etc., of facial landmarks and aims:

- ✓ All faces across the entire training set to have identical size (256 x 256 in our case).

- ✓ Every face to be centered in the image that it appears.

- ✓ The eyes to lie on the x-axis.

In particular, face alignment involves the following workflow, in order to address the above goals:

1. Rotation of the face in the image, so that the upright to be based on the eye positions.

2. Definition of a central point on the face making use of the mid-point between the leftmost and right most landmarks.

3. Estimation of the average of the eyes landmarks as well as the average of the mouth landmarks, so that the corresponding center points could be defined.

4. Centering the faces in the x-axis, based on the central point (blue point).

5. Placement of the center points of eyes and mouth at the 45% from the top of the image and the 25% from the bottom of the image, respectively. This step fixes the position of the face along y-axis.

6. Image resize to 256×256.

The next step is converting the current data set to encodings (i.e 128-d vectors). Dlib's face recognition network is responsible for generating a feature vector per face in the data set. Hence, a 3-D matrix is built, where all the encodings that correspond to the training samples of the Face Recognition system are appended. In particular, every face in the data set is being represented by:

1. the image file path,
2. the face encodings and
3. the class name

Once a new face sample or a completely new class is being inserted in the data set, the matrix is being updated, whereas the existing items that are being identified by their file path do not change.

4.3 Object Detection Data

In this section we must highlight the main key points for collecting data that are going to train a custom Object Detection model. An Object Detection model requires objects' images followed by a file (xml/txt) that includes the actual coordinates of each object within the image as well as the object's class. The total number of training samples per class is suggested to be higher than 50. The Object Detection accuracy is being significantly enhanced when a variety of object's features (i.e color, size, shape) is being included in the data set and all image samples are representative of their class.

In is worth noting that the current pre-trained model, which is going to make the baseline predictions, has been trained on 80 classes (see Figure 4.2). For the purposes of this thesis we attempted to build two individual data sets, in order to investigate the results of the custom training as well as training an Object Detection model, specialized in detecting persons that wear a surgery mask. Therefore, we collected image samples that include masks with different sizes, colors, shapes etc. The data construction process was totally based on the usage of the OpenCV-Video-Label annotation tool. Specifically, we played freely available video steaming via the OpenCV-Video-Label tool and we captured people in crowd wearing masks.

The data set that has been constructed contains 220 image samples and 220 xml files. Every xml file includes information about the class name and the precise bounding box coordinates that encloses the object within every image. The data set has been splitted

into training (90%) and validation (10%) sets.

The second data set that we have built for further experimenting with more classes includes the following 5 classes with the corresponding number of total samples:

1. mask: 220 samples
2. watch: 248 samples
3. flag: 137 samples
4. glasses: 147 samples
5. hat: 151 samples

The split percentage into training and validation data was approximately 90% and 10%, respectively.

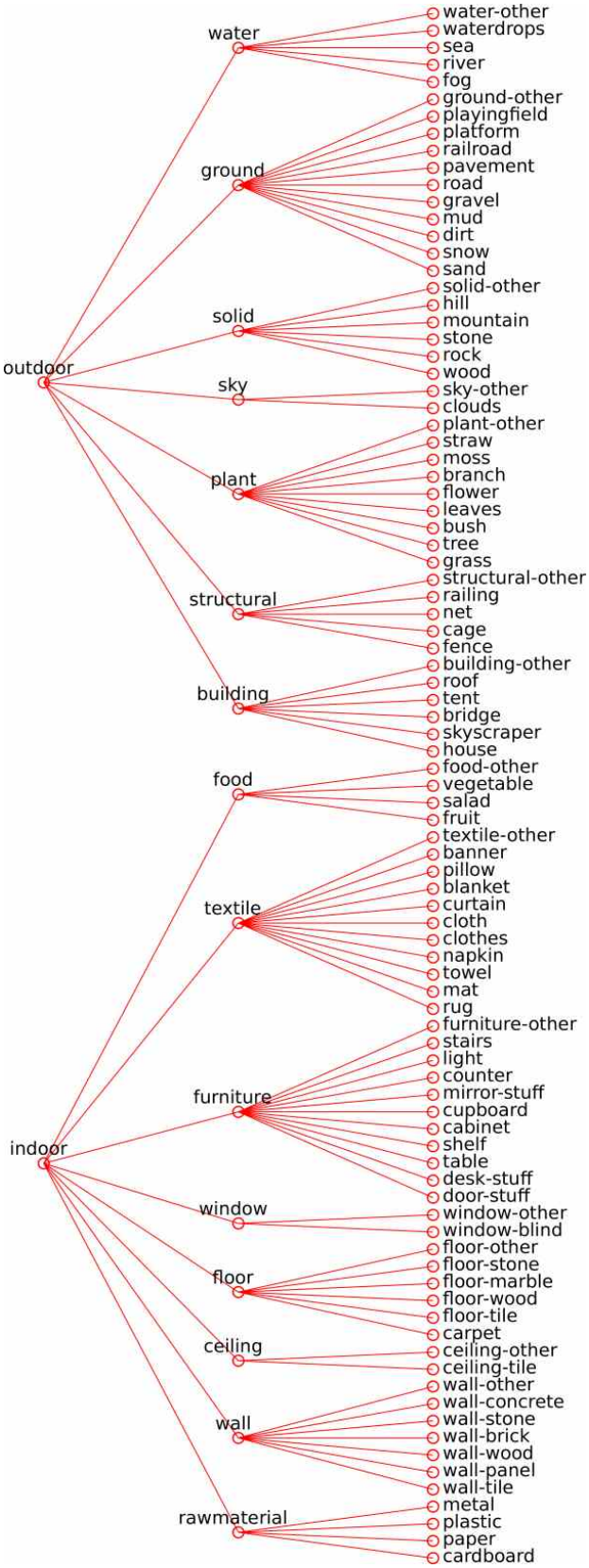


Figure 4.2: *The COCO data set (80) classes divided into groups. [28]*

5

Chapter 5: Active Learning Framework

This section introduces the framework for Face Recognition and Object Detection, which makes use of Active Learning techniques leading to automated data set enhancement. Our integrated system combines Face Recognition and Object Detection models and results in extracting knowledge from both of them as well as associating their outputs providing us additional information about frames' contents.

Furthermore, a sophisticated optimization method is incorporated into our system, which enhances the recognition accuracy. Meanwhile, we built a metadata repository that consists of details about every input that undergoes the system (e.g unique id, duration, frame rate etc) as well as system's variables (e.g number of classes per model) and output/logs.

5.1 Face Recognition

5.1.1 Classifier Training

After generating faces encodings, a Machine Learning classifier is being trained, in order to make probabilistic predictions for every detected face in the input video. Training is being conducted on encodings column (training samples), in association to the corresponding class name column (labels), which are included in the aforementioned 3-D matrix. According to bibliography, Support Vector Machines (SVM) [29] and K-Nearest Neighbours (KNN) [30] have been proven the most efficient classifiers for encodings classification tasks. A classifier comparison between SVM and KNN is necessary, so that the optimal classifier, in terms of time and accuracy, could be decided.

For the purpose of this comparison we defined $threshold_A = 0.8$. Hence, every true and false prediction higher than $threshold_A$ were categorized as true positive (TP) and false positive (FP), respectively. The initial data set that was created for our experiments consisted of 10 classes. From the very first results, we observed the SVM classifier saving time in both training and recognition process. On the contrary, KNN classifier required further experimentation concerning the appropriate K-value, which as it turned out depends on the number of classes.

For instance, we implemented the following experiments for a variety of number of classes (from the initial data set) and K-value = {2,5,10,20}:

1. low number of classes (e.g $2 \leq$ number of classes ≤ 5)
2. normal number of classes (e.g $10 \leq$ number of classes ≤ 20)
3. high number of classes (e.g $100 \leq$ number of classes)

During the experiments we estimated the precision score, since it's a pretty hard task to estimate recall and accuracy scores for the reason that there is no rule defining both true and false negative (TN and FN) predictions. More precisely, a Face Recognition system is expected to fall in identifying a turned face, since a great percentage of the face landmarks are hidden. As a result there would be a conflict for which observations correspond to TN/FN. Therefore, we noted the above observations as well as the precision score that we calculated for each set of experiments:

1. There is a significant increase of FP predictions for K-values ≤ 20, while the number of TP predictions is quite low for every tested K-value. It is worth noting that all the confidence values equal either 0 or 1.

2. ◇ The results seem to be remarkably improved in comparison to the previous set of experiments. The optimal precision score for 10 classes is approximately 90% (when K = 5).

   ◇ In addition, there is a further improvement for 20 classes. As a result, the precision score exceeds 95%.

   ◇ The total number of TP predictions is being significantly increased during this set of experiments (see Figures 5.1 and 5.2). Furthermore, there is a higher variability as regards to the confidence values, since there are values different from 0 and 1.

3. During the last set of experiments, where the number of classes exceeds 100, we observed a noticeable delay during recognition process (see Figure 5.3), while the TP predictions are quite fewer in number.

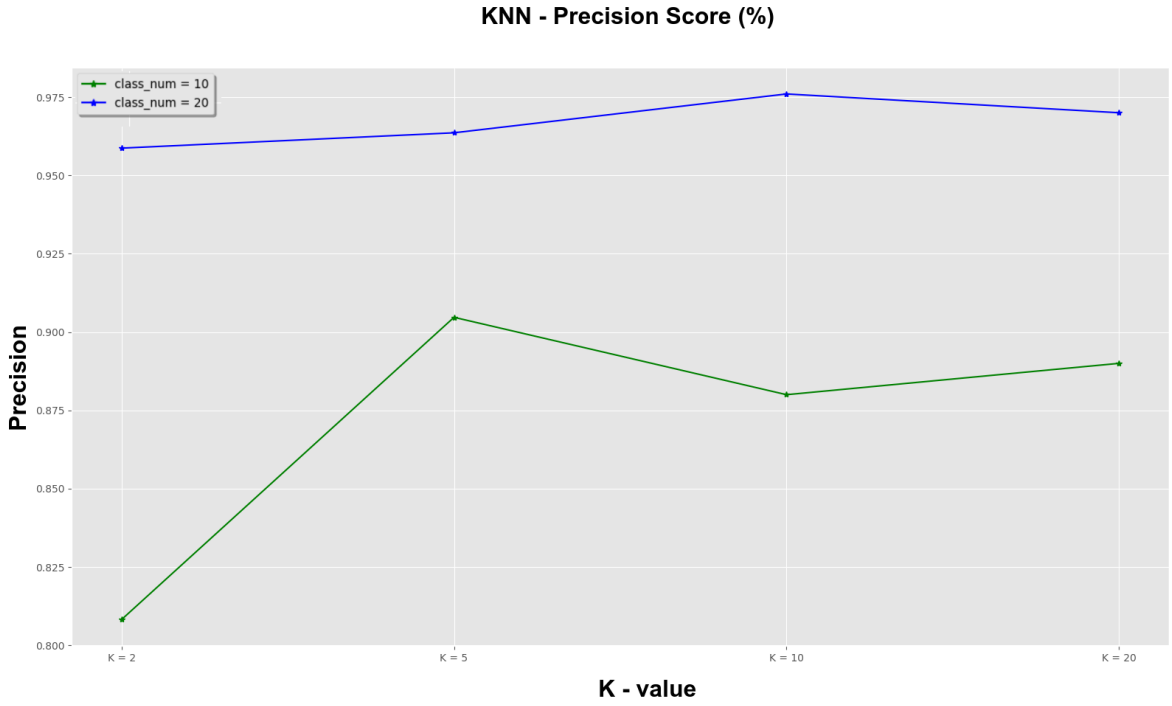


Figure 5.1: *KNN classifier. Precision score for different K-values and number of classes = {10,20}.*

Then, we repeated the same experiments (same input video and data set) for SVM classifier. Nevertheless, there is no K-value to investigate, so we only made observations for the different number of classes.

1. There is a significant increase of FP predictions as well as a low deviation between the confidence values that correspond to TP and FP. As a result, the precision score is significantly low.

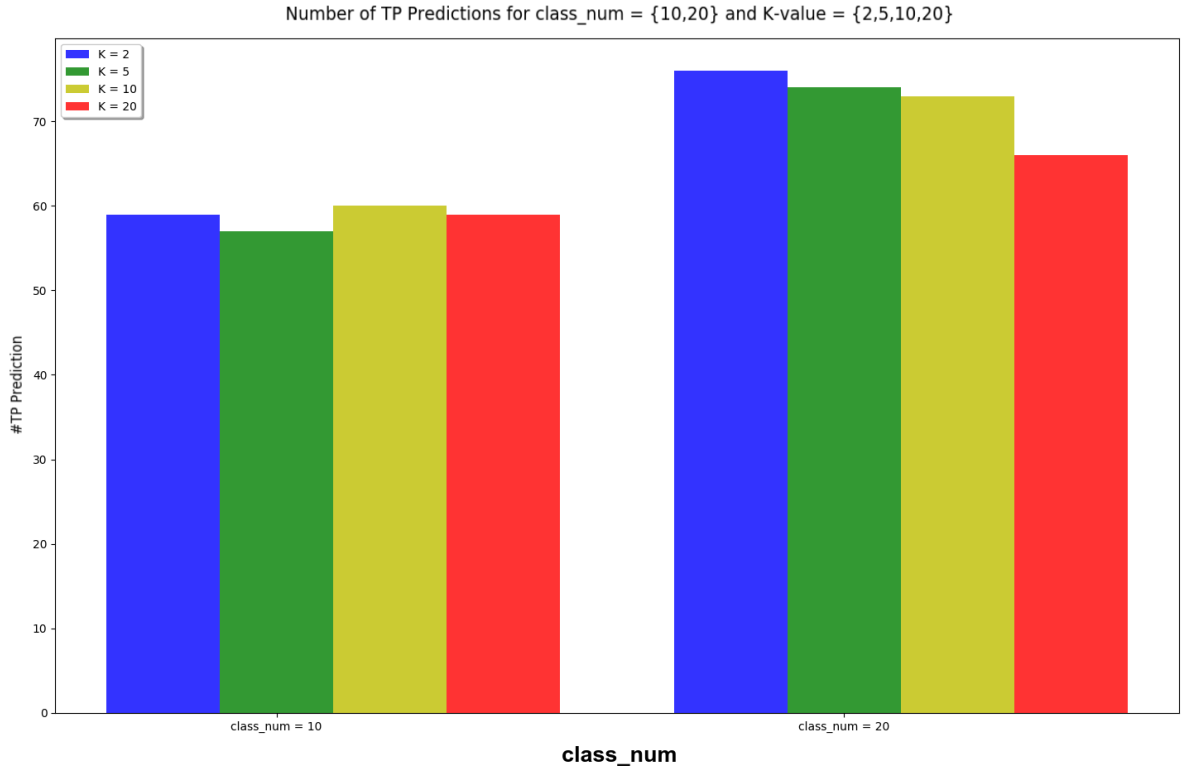


Figure 5.2: *KNN classifier. Number of TP predictions for number of classes = {10,20} and K-value = {2,5,10,20}.*

2. The results seem to be remarkably improved in comparison to the previous set of experiments, while the precision score exceeds 95%.

3. During the last set of experiments, where the number of classes exceeds 100, we observed a remarkable delay during classifier training process, while the duration of the recognition process is quite lower than the duration of the recognition process that was performed with the KNN classifier (see Figure 5.5)).

To sum up, we concluded to the SVM as the appropriate classifier for encodings classification tasks for the 3 basic reasons:

1. Both classifiers are accurate for a normal number of classes (e.g 10 $\leq$ number of classes $\leq$ 20) and there is not a remarkable difference, in terms of precision score.

2. KNN demands investigation as regards to the K-value, as the number of classes is being increased. In Figures 5.1 and 5.2 we can see that the optimal performance may differ for different number of classes (e.g 10-classes and 20-classes data set). This means that there should be further experimentation concerning the K-value, which could cause extra delay.

3. KNN is extremely time-consuming during recognition process for a high number of classes in comparison to SVM. Nevertheless, SVM training process was proved more time-consuming than KNN training process. However, classifier training is being performed less frequently (only if the data set is being updated), while the recognition process corresponds to the main sub-process of the Face Recognition workflow, which is being executed regularly and for a variety of inputs. Figure 5.3 and 5.4 represent the duration of the sub-processes (that is, training and recognition processes) for different number of classes (i.e 2,10,100) per classifier, while Figure 5.5 is the bar chart that shows the duration of each sub-process per classifier, when the number of classes in data set is quite high (e.g number of classes = 100). It is worth mentioning that the severe delay underlies the main reason for avoiding KNN classifier during recognition process.

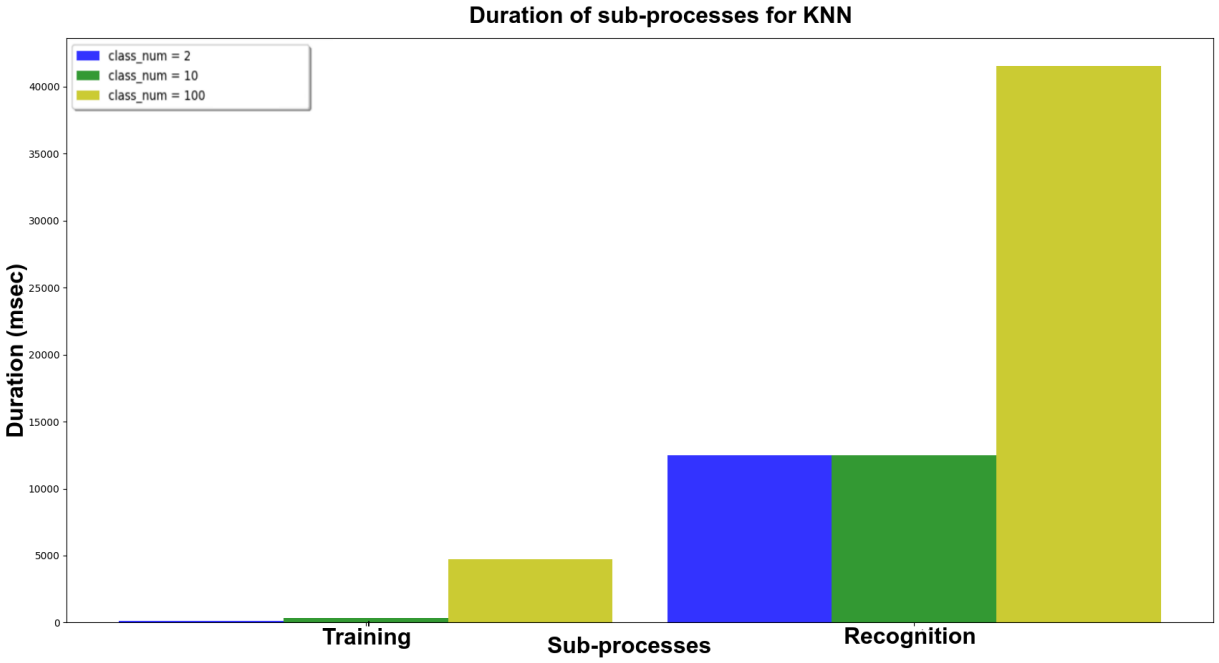


Figure 5.3: *KNN classifier. Duration of Face Recognition sub-processes for different number of classes.*

As we have already highlighted, the number of classes reflects on results. In fact, an extremely low number of classes (e.g ≤ 5) could cause increase of FP predictions, while an extremely high number of classes could result in significant delay during the Face Recognition sub-process. For this reason, we copied the initial 10 classes multiple times, so that we could create larger data sets of 65, 650 and 6500 classes. We aimed to estimate only the duration of classifier training and recognition process per number of classes, therefore duplicates of classes do not reflect on our observations.

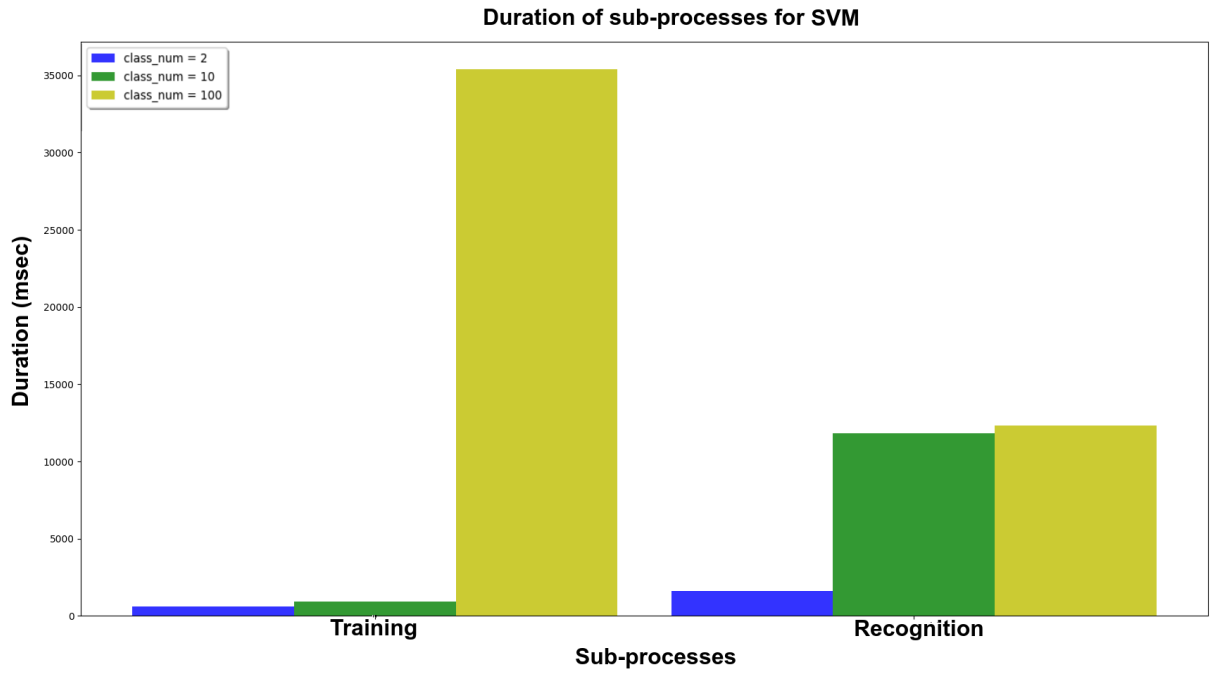


Figure 5.4: *SVM classifier. Duration of Face Recognition sub-processes for different number of classes*

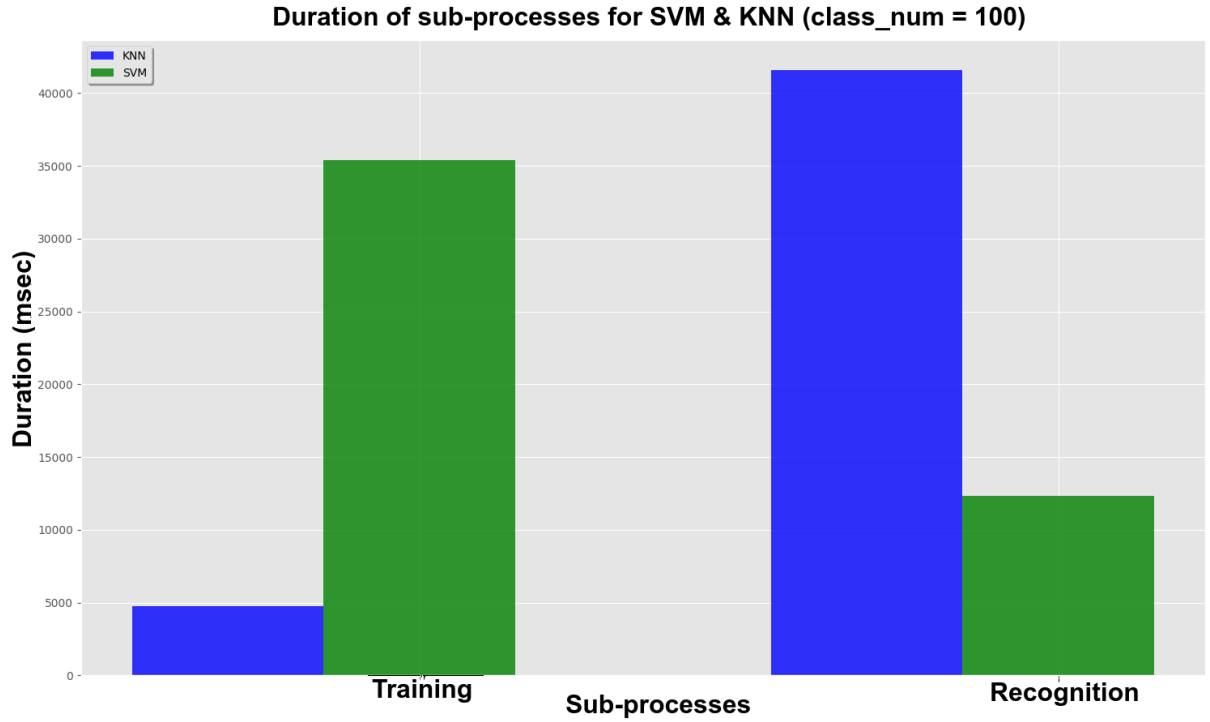


Figure 5.5: *KNN vs SVM. Duration of Face Recognition sub-processes for number of classes = 100*

Table 5.1 shows the training and recognition time for the same input but different number of classes. The input video duration equals 10 minutes, while the frame rate has been reduced to 2 frames per second, in order to speedup the process. Usually the original

frame rate of a video equals to 25 frames per second. Nevertheless, Face Recognition could perform satisfyingly and significantly faster for 2 frames per second. In particular, the reduced frame rate does not affect the recognition accuracy.

For the cases of 650 and 6500 classes we attempted to train two individual classifiers instead of one. The two classifiers did not share any class. Taking into account Table 5.1, we decided that the total number of classes per classifier shouldn't exceed 600, since the training process takes many hours to finish for an extended data set (e.g 6500 classes). Recognition time is related to the total number of classes of merged classifiers, too. Finally, we noticed that splitting a heavy data set to two or more classifiers addressed the delay issues, for both training and recognition processes.

Consequently, taking into account all the experiment concerning the number of classes included in the data set that is intended to train a SVM classifier, we concluded that every SVM classifier is recommended to be trained on a normal number of classes (i.e $5 \leq$ number of classes ≤ 20). As a result, when the class number exceeds 20 we could divide the training process into multiple SVM classifier training, in order to ensure the optimal performance, in terms of time and accuracy.

Class Number	Classifier Number	Training Time (sec.)	Recognition Time (sec.)
65	1	2.7	76
650	1	278	146
650	2	3.6	77
6500	1	$6 * 10^4$	977
6500	2	$13 * 10^3$	660

Table 5.1: *SVM classifier. Comparison of the training and recognition duration for different number of classes and classifiers.*

5.1.2 Predictions Filtering

After training one or more SVM classifiers, the system is ready to make predictions on input videos. For every detected face in a video frame the classifier predicts a probability value (confidence) for each class in the data set, resulting in a matrix whose dimensions equals (number of classes x frame number). Face identification results from the highest confidence value among all classes.

Prediction flickering is a common phenomenon in most FR systems. Flickering renders the inconsistency of face identification through frames, which is really intense when the detected face is moving. Thus, we incorporated a filtering technique, which stabilized the

TP predictions. In many cases, the FP predictions may also increased slightly. The actual steps of filtering on an input are demonstrated in the Algorithm 1. In Figures 5.6 and 5.7 we present the confidence diagram for a TP and FP case, respectively. After applying the algorithm, the confidence values have been smoothed, while the flickering phenomenon has been eliminated.

Algorithm 1: Prediction Filtering

1. We zero all the confidence values that are under the $threshold_A$, since they probably correspond to FP.

2. We apply a median filter with a N-size kernel, i.e we re-estimate every class's confidence in a current frame, taking into account the confidences in both previous and next N frames. Finally, we replace the current value with the median value of the list that has occurred:

$$V^C[i] = median(V^C list)$$

- $V^C_{list} = \{V^C[i-2], V^C[i-1], ..., V^C[i+2]]\}$,
- V = confidence value
- C = class
- i = frame number

3. In the same way we apply a sum filter with a M-size kernel on the updated confidence values that have arisen from the previous step:

$$V^C[i] = \sum_{i-M}^{i+M} V^C[i]$$

4. We replace the updated confidence values as follows:
- if $V^C[i] > cutoff$ then $V^C[i] = 0.99$
- else $V^C[i] = 0.01$, where cutoff $\in [5.0 - 6.0]$

Moreover, we executed further experiments concerning the size of the median kernel. We concluded that the median kernel size is associated with the input's frame rate. For example we took an input video with frame rate equal to 6 and we isolated a TP as well as a FP case of predictions (see Figures 5.8 and 5.9).

Figure 5.8 shows the increase of the TP predictions when the kernel size approach the frame rate. Also, there are less FN predictions (i.e flickering) between TP predictions that are located in immediate frames. We expected this phenomenon, since the sliding window than invokes the median kernel is looking in a wider range of immediate frames, in order to estimate the median value of the confidence values list for a current frame.

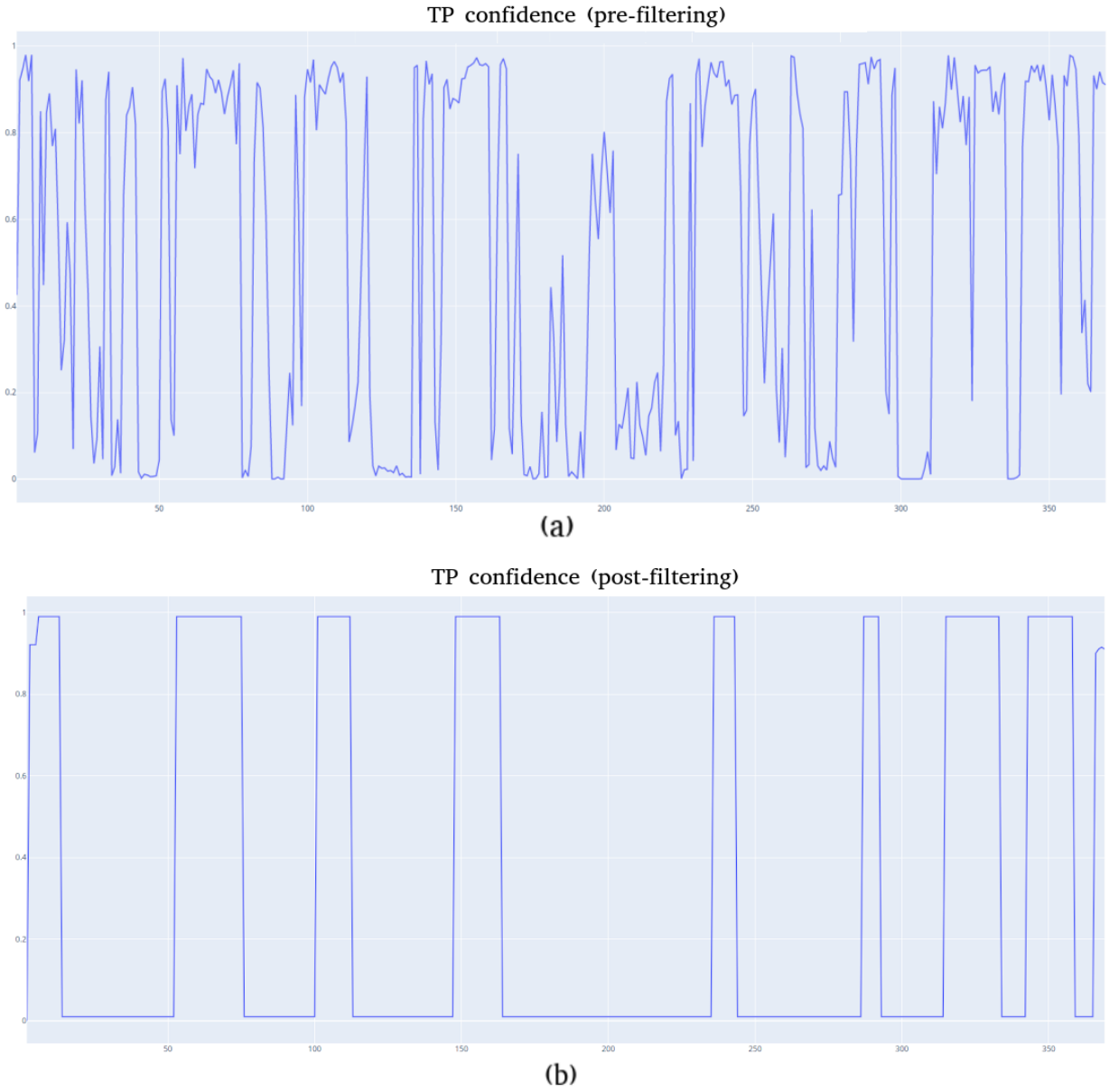


Figure 5.6: *TP example of confidence values through frames, where: (a) represents the pre-filtering confidence values and (b) represents the post-filtering values after applying the Algorithm 1. The confidence values that approach zero in both (a) and (b) correspond to frames where the current person is out of focus or disappears entirely.*

Figure 5.9 reveals that the FP values are being eliminated as the kernel size is being increased. Nevertheless, when FP predictions appear in immediate frames they are expected to join together causing a slight increase of their number. As a result, the filtering algorithm that involves the sliding window succeeds when it exploits confidence values 1 second before and after the every frame.

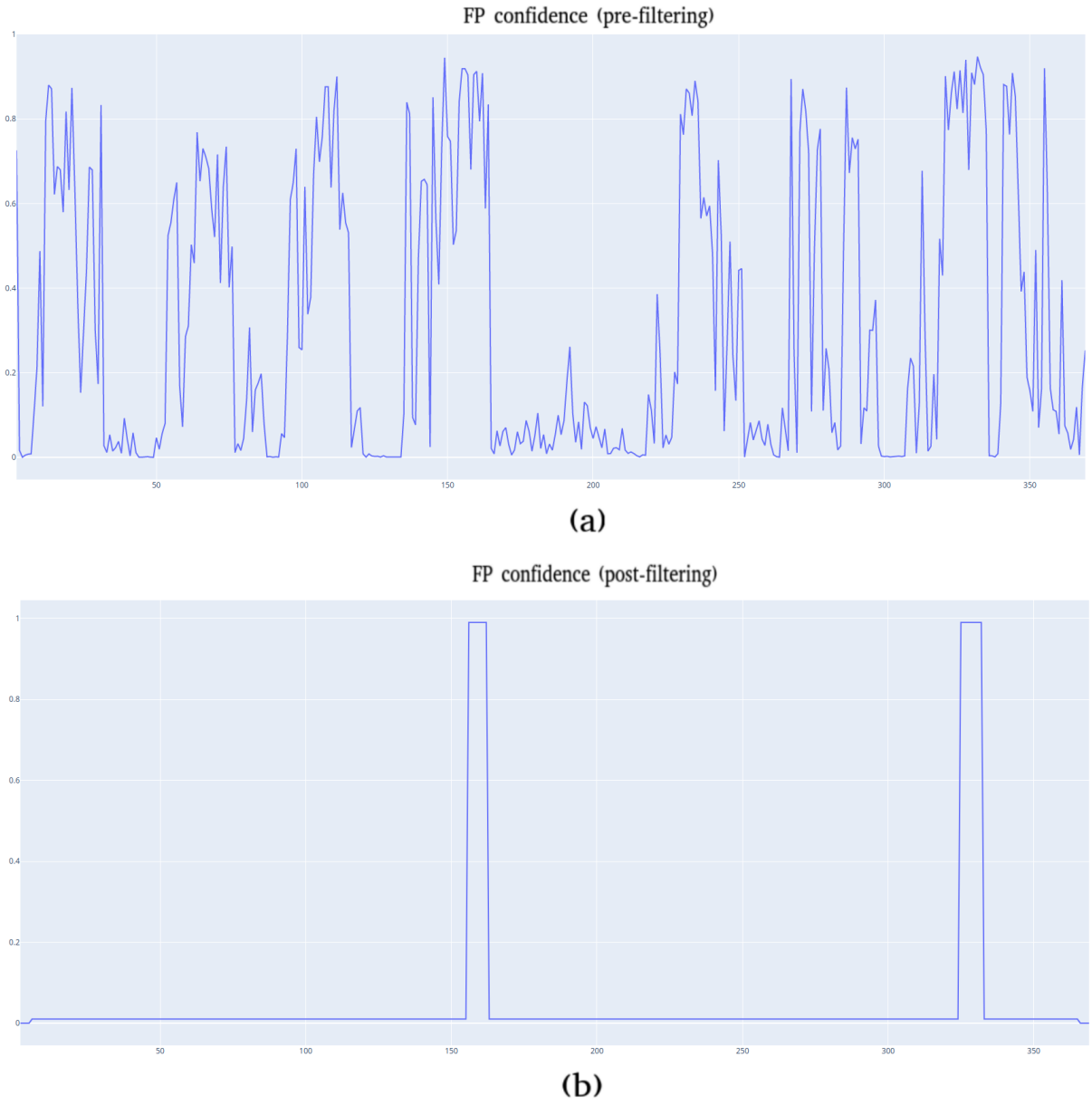


Figure 5.7: *FP example of confidence values through frames, where: (a) represents the pre-filtering confidence values and (b) represents the post-filtering values after applying the Algorithm 1.*

5.1.3 Face Recognition Output

As soon as an input video is being proceeded for face recognition, we extract knowledge for both "known" faces, which correspond to data set's classes, and "unknown" faces, which are not included in the data set, yet. Particularly, a face is being classified as "known" if the class (post-filtering) confidence equals 0.99, while a face is being classified as "unknown" if the maximum (pre-filtering) confidence for all classes is under $threshold_B$ (e.g 0.5). When the Face Recognition process is over, we export:

- ***(FR-out-1)***: A matrix where rows and columns correspond to the data set's classes

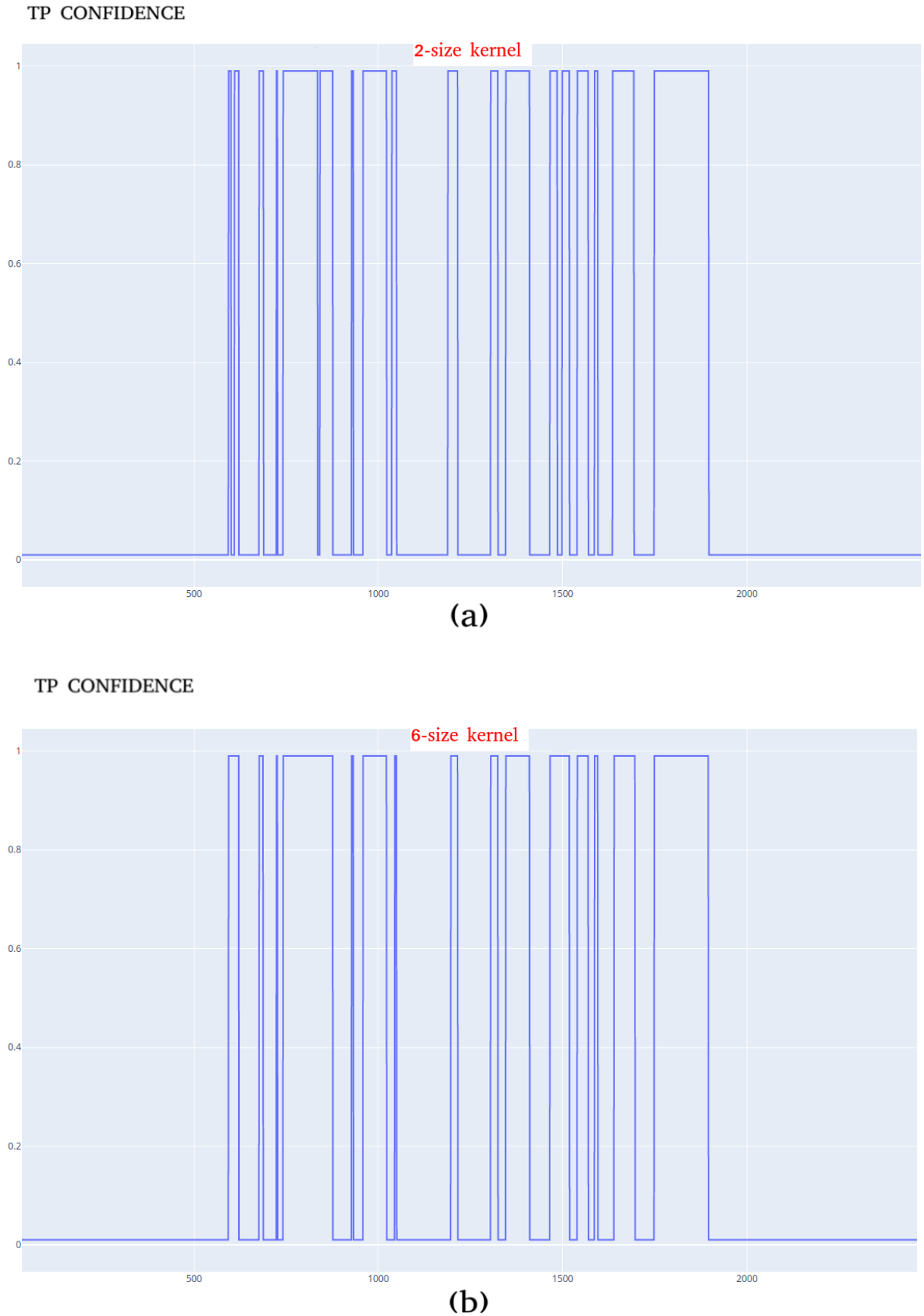


Figure 5.8: *Example of TP post-filtering confidence values, after defining two different kernels with sizes 2 (a) and 6 (b). The TP predictions of immediate frames are joined together, while the median kernel is being increased (from 2 to 6)*

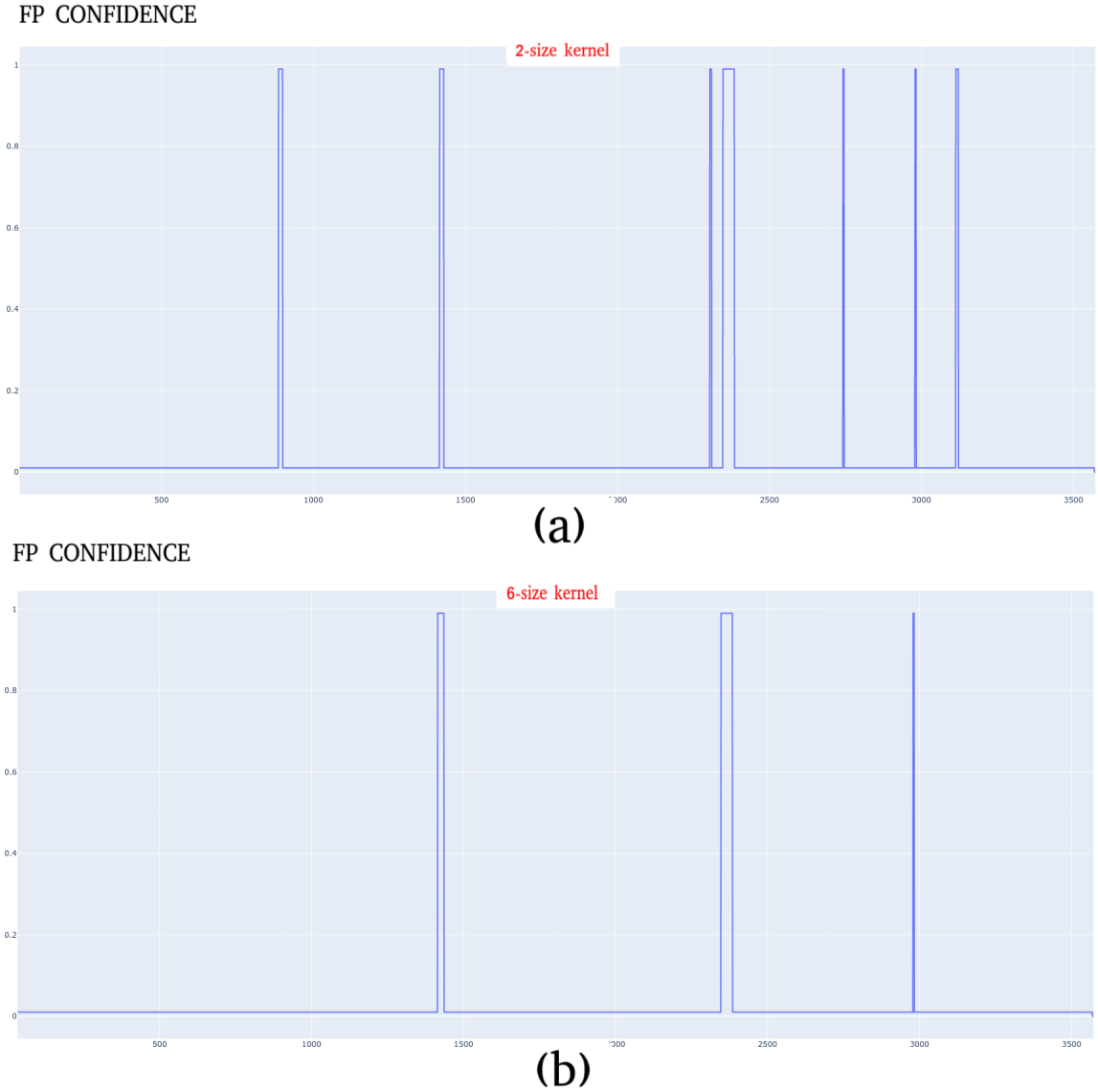


Figure 5.9: *Example of FP post-filtering confidence values, after defining two different kernels with sizes 2 (a) and 6 (b). The FP predictions of faraway frames are being eliminated, while the median kernel is being increased (from 2 to 6).*

and the frame number, respectively. Each cell in the matrix includes the precise coordinates of the bounding box that enclosed the detected face, which has appeared in the current frame (i.e column) as well as the confidence score for the corresponding class (i.e row). Figure 5.10 represents an example of FR-out-1.

- ***(FR-out-2)***: A dictionary where the keys are the "known" faces that have appeared in the video while the values per key are:
  1. the class name;
  2. a list of the frames where the corresponding class appeared and
  3. the coordinates of the bounding boxes that enclosed the face within the frames,

where it has appeared.

Figure 5.11 (a) shows an example of FR-out-2.

- ◇ ***(FR-out-3)***: The encodings of the "unknown" faces that have appeared in the video, in association with the coordinates of the bounding boxes that enclosed the faces within the frames, where they have appeared. Figure 5.11 (b)) represents an example of FR-out-3.

- ◇ An annotated video where all the detected faces are within a bounding box followed by either a class tag or an unknown tag.

	name	frame_1_0.5	frame_2_1.0	frame_3_1.5
1	Person-1	[441, 845, 489, 798, 0.0]	[436, 853, 493, 796, 0.0014]	[441, 826, 489, 779, 7e-05]
2	Person-2	[441, 845, 489, 798, 0.92999]	[436, 853, 493, 796, 0.02665]	[441, 826, 489, 779, 0.71735]

	name	frame_4_2.0	frame_5_2.5	frame_6_3.0
1	Person-1	[436, 821, 484, 774, 7e-05]	[494, 1119, 542, 1071, 0.02215]	[484, 1128, 532, 1081, 0.0108]
2	Person-2	[436, 821, 484, 774, 0.69888]	[436, 816, 484, 769, 0.97228]	[436, 825, 493, 768, 0.88373]

Figure 5.10: *Example of the face appearances per frame, in a random input video. The first 4 numbers in every cell represent the coordinates of the bounding box that encloses the detected face, while the last number corresponds to the confidence value.*

As a result, we have exploited the information about "unknown" faces via the algorithm that we describe below (see Algorithm 2).

Algorithm 2: Generate training samples for unknown faces

1. Once a new class is being added in the data set we generate its encodings.
2. Then, we estimate the distance between the encodings that have been produced for the new class and those that have been classified as "unknown".
3. We define a low threshold value (e.g $threshold_C = 0.5$).
4. If the estimated distance is under the $threshold_C$ then both encodings probably represent the same face.

For example, if a new class enters the data set then the Algorithm 2 is executed, in order to export a log with the appearances of the new class in all the input videos that have already undergone the system.

Figure 5.12 shows an example of the log that the system have exported after updating the data set with the new class (e.g Person_new). The log is divided into individual parts that represent all the input videos that have been recorded by the system (i.e all the input videos that have undergone our system). For every class's appearance in an input, we present the

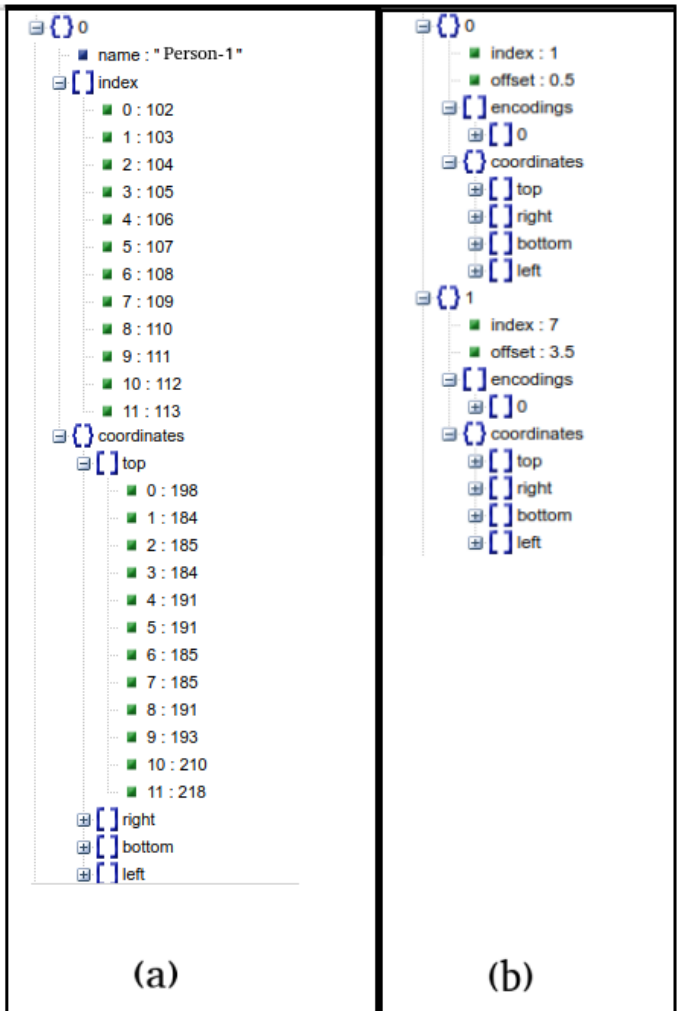


Figure 5.11: *(a) Example of the dictionary that includes the "known" faces that have appeared in a random video. (b) Example of the dictionary that includes the "unknown" faces that have appeared in a random video. The key represents the "unknown" detected face, while the values corresponds to: i. the frame index, where it appeared as well as the exact offset, ii. the encodings list (128-d) and iii. the coordinates of the bounding box that encloses the detected face.*

exact frame, the offset (i.e the exact second in the video) as well as the coordinates of the bounding box that enclosed the new class's face in the video. In Figure 5.12 we can also see input ids with no content, which means that the new class has not appeared in these inputs.

Consequently, when an "unknown" class is being inserted (by the Machine Learning engineer) in the data set, our efficient Active Learning framework detects all its previous appearances in recent input videos and gives access to more images that could be used as training samples for further training on the current class. In the same way, a "known" class could be enhanced with more samples that originates from predictions with high confidence values.

It is worth noting that our framework includes the human-in-the-loop in order to

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 1. input_1 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-Person_new has appeared in video: input_1
 -> frame: 7 - offset: 3.5 - coordinates: {'top': [210], 'right': [425], 'bottom': [292], 'left': [343]}
-Person_new has appeared in video: input_1
 -> frame: 9 - offset: 4.5 - coordinates: {'top': [193], 'right': [433], 'bottom': [275], 'left': [352]}

 ...

-Person_new has appeared in video: input_1
 -> frame: 351 - offset: 175.5 - coordinates: {'top': [210], 'right': [641], 'bottom': [292], 'left': [559]}
-Person_new has appeared in video: input_1
 -> frame: 363 - offset: 181.5 - coordinates: {'top': [361], 'right': [708], 'bottom': [401], 'left': [669]}

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 2. input_2 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

...


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ 9. input_3 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-Person_new has appeared in video: input_3
 -> frame: 638 - offset: 319.0 - coordinates: {'top': [176], 'right': [409], 'bottom': [245], 'left': [341]}
-Person_new has appeared in video: input_3
 -> frame: 639 - offset: 319.5 - coordinates: {'top': [176], 'right': [409], 'bottom': [245], 'left': [341]}
-Person_new has appeared in video: input_3
 -> frame: 640 - offset: 320.0 - coordinates: {'top': [176], 'right': [409], 'bottom': [245], 'left': [341]}
-Person_new has appeared in video: input_3
 -> frame: 641 - offset: 320.5 - coordinates: {'top': [166], 'right': [417], 'bottom': [248], 'left': [335]}
-Person_new has appeared in video: input_3
 -> frame: 652 - offset: 326.0 - coordinates: {'top': [170], 'right': [409], 'bottom': [238], 'left': [341]}
-Person_new has appeared in video: input_3
 -> frame: 723 - offset: 361.5 - coordinates: {'top': [166], 'right': [191], 'bottom': [222], 'left': [134]}
-Person_new has appeared in video: input_3
 -> frame: 724 - offset: 362.0 - coordinates: {'top': [166], 'right': [197], 'bottom': [222], 'left': [140]}
```

Figure 5.12: *(a) Example of the implemented system's exported log, when a new class is being added in the data set. The "input_{X}" corresponds to the unique id of the input video that has already undergone the system. Each class's instance is followed by the frame number, the offset and the coordinates where Person_new has been detected.*

provide meaningful guidance and control the images samples that enter the data set. More precisely, the human expert is able to exclude image samples that may harm the recognition accuracy (e.g hidden face landmarks, low resolution etc.).

The output video could also be utilized for further training purposes, since it provides knowledge about the detected faces' coordinates and names. Specifically, we could use the annotated (output) video instead of the original during data preparation as it enables the human eye to distinguish the "known" from the "unknown" faces.

5.2 Object Detection

5.2.1 Training and Evaluation

ImageAI provides a detailed documentation [1] for Object Detection training on custom data sets. Firstly, it is recommended to transfer learning from YOLOv3 pre-trained weights to the custom training process. However, the output layer that is responsible for classifying every detected object does not participate in the custom training. As a result,

[1]https://imageai.readthedocs.io/en/latest/customdetection/

every custom trained model makes predictions only for classes included in the current (custom) data set.

During our experiments we concluded that the number of classes per custom training should not exceed 10, in order to achieve good results. Training process ends when the validation loss stops decreasing and mAP score is not being significantly increased in comparison to previous epochs. ImageAI enables saving epoch's checkpoints and provides an evaluation method that estimates the mAP score of the exported models, so that the optimal model between checkpoints could be distinguished.

5.2.2 Mask Model

In the previous section we introduced two different data sets for further training the Object Detection systems, using the YOLOv3 pre-trained weights. The first attempt of training resulted in the mask model which is able to detect surgery masks.

The following detailed reports refer to the training and evaluation process, respectively:

Training report:

- Total number of epochs: 200
- Batch size: 4
- Transfer learning from YOLOv3 pre-trained weights
- Total training duration (for 200 epochs): 13034 seconds (approximately 3.62 hours)

Evaluation report:

- Total validation duration (for 30 checkpoints): 3160 seconds (approximately 52.7 minutes).
- The optimal model was exported from 23th epoch's checkpoint.
- validation loss of the optimal model: 2.53%
- mAP score of the optimal model: 85%.

In Figure 5.13 we present the evaluation results of the custom trained mask model on a random image, which shows people in crowd wearing surgery masks. As it emerges, the mask detection has detected successfully every mask.

Figure 5.13: *Evaluation results of the mask model on a random image sample.*

5.2.3 5-classes Model

The second attempt of the Object Detection system training consisted of 5 object classes that are not included in the COCO data set. The following detailed reports refer to the training and evaluation process, respectively:

Training report:

- Total number of epochs: 200
- Batch size: 4
- Transfer learning from YOLOv3 pre-trained weights
- Total training duration (for 200 epochs): 51295 seconds (approximately 15 hours)

Evaluation report:

- Total validation duration (for 30 checkpoints): 3160 seconds (approximately 52.7 minutes).
- The optimal model was exported at 10th epoch's checkpoint.

- validation loss of the optimal model: 4.428%
- total mAP score of the optimal model: 45%
- mAP score per class:
  - flag: 0.1458
  - glasses: 0.6667
  - hat: 0.5556
  - mask: 0.1633
  - watch: 0.7231

Additionally, we attempted to increase the 5-classes mAP score on the validation data by training the Object Detection model with an alternative framework that is highly recommended by literature. Training with Darknet requires the conversion of the labels (xml files), which have been exported by the OpenCV-VIdeo-Label, to the appropriate format (txt). This step was accomplished by the XmlToTxt [2] open-source tool. It is worth mentioning that Darknet's configuration relied on the available detailed documentation [3], in order to obtain satisfying results. Also, we had to prepare multiple external files that are necessary for activating the training process (e.g object name/data paths etc.).

Darknet's training process had the same duration as the ImageAI training process, so that we could compare the accuracy results. The following detailed reports refer to the training and evaluation process, respectively:

Training report (Darknet):

- Total number of epochs: 10000
- Batch size: 64
- Transfer learning from YOLOv3 pre-trained weights
- Total training duration (for 200 epochs): approximately 15 hours

Evaluation report (Darknet):

- The optimal model was exported at the 10000 epoch.
- validation loss of the optimal model: 0.1047%

[2]https://github.com/Isabek/XmlToTxt
[3]https://manivannan-ai.medium.com/how-to-train-yolov3-to-detect-custom-objects-ccbcafeb13d2

- total mAP score of the optimal model: 56%
- mAP score per class:
  - flag: 0.0833
  - glasses: 0.75
  - hat: 0.8586
  - mask: 0.5456
  - watch: 0.5606

Finally, the Darknet exported model was converted to keras model, so that we could perform Object Detection by ImageAI. We satisfied this demand by the open-source keras-yolo [4] tool. Both Darknet and ImageAI models were evaluated by random input videos, where the 5 object classes have appeared. We observed that both of them were quite accurate as regards the testing input video, despite the noticeable deviation between the metrics' values (validation loss, mAP score etc.) that were estimated during the evaluation on the validation set, which is consisted of image samples.

Nevertheless, for the Darknet model the default anchors that the training process involves resulted in inaccurate bounding boxes for the detected objects. Thus, we conclude the ImageAI as the appropriate framework for training a custom Object Detection model that would fit the demands of our implemented system. In particular, by choosing ImageAI we save time in the preparation of the training process, the configuration as well as the conversion of the exported model.

In Figure 5.14 we present the evaluation results of the custom trained 5-classes model on a random image, which shows people in crowd wearing surgery masks. As it emerges, the model has detected successfully every visible mask, while blurred masks or masks out of focus could not be detected. As regards to persons detection, we can see persons in background that have been detected successfully. An interesting usage scenario of such an model could be the estimation of the percentage of people in crowd wearing a surgery mask or another object that has been included in the custom data set. For instance, in Figure 5.14 there have been detected 3 masks and 5 persons.

5.2.4 Object Detection Output

During the prediction process on an input video, the "known" object classes are being identified only if their predicted probabilities are higher than a pre-defined $tolerance_{OD}$ value (e.g 0.5). When the total Object Detection process is over, we export:

[4] https://github.com/qqwweee/keras-yolo3

Figure 5.14: *Evaluation results of the 5-classes model on a random image sample.*

- ***(OD-out-1):*** The precise coordinates of the bounding boxes that enclose the detected objects in each frame. There is also a confidence value per prediction, which represents the probability that the detected object corresponds to each predicted class. For example Figure 5.15 presents the detected objects in the first 20 frames of a random input video.

- ***(OD-out-2):*** The total number of instances of the object present in every frame. Figure 5.16 represents an example of the first 20 frames for the same input video that we have used for our experiments.

- An annotated video where all the detections that have been identified are within a bounding box followed by a class tag.

It is worth mentioning that when an input undergoes Object Detection multiple times for different custom trained models, the most recent output is being merged with the existing (from previous Object Detection executions), in order to maintain the total information. This step is necessary, since the training process is based on transfer learning that as it has been proven improves the model's generalization. As a result, the custom trained model involves the internal layers of the pre-trained YOLOv3 model. Hence, the

	0	1
2	{'name': 'tie', 'percentage_probability': 94.07007694244385, 'box_points': [236, 269, 311, 564]}	{'name': 'person', 'percentage_probability': 97.28384613990784, 'box_points': [0, 50, 568, 1071]}
3	{'name': 'tie', 'percentage_probability': 66.5198802947998, 'box_points': [974, 467, 1050, 758]}	{'name': 'person', 'percentage_probability': 95.64663767814636, 'box_points': [0, 0, 300, 1042]}
4	{'name': 'tie', 'percentage_probability': 51.82409882545471, 'box_points': [927, 466, 1016, 761]}	{'name': 'person', 'percentage_probability': 99.78691935539246, 'box_points': [567, 201, 1174, 1062]}
5	{'name': 'tie', 'percentage_probability': 53.438758850097656, 'box_points': [744, 443, 849, 492]}	{'name': 'tie', 'percentage_probability': 81.84901475906372, 'box_points': [782, 475, 879, 754]}
6	{'name': 'tie', 'percentage_probability': 89.49825763702393, 'box_points': [781, 471, 879, 775]}	{'name': 'person', 'percentage_probability': 99.33210015296936, 'box_points': [368, 149, 1132, 1063]}
7	{'name': 'tie', 'percentage_probability': 91.43044948577881, 'box_points': [859, 503, 937, 807]}	{'name': 'person', 'percentage_probability': 97.40893840789795, 'box_points': [1, 0, 421, 1055]}
8	{'name': 'tie', 'percentage_probability': 62.730902433395386, 'box_points': [478, 412, 520, 518]}	{'name': 'tie', 'percentage_probability': 84.36029553413391, 'box_points': [1066, 460, 1127, 682]}
9	{'name': 'tie', 'percentage_probability': 59.92063879966736, 'box_points': [466, 418, 513, 596]}	{'name': 'tie', 'percentage_probability': 88.96875381469727, 'box_points': [1089, 478, 1157, 667]}
10	{'name': 'tie', 'percentage_probability': 61.35942339897156, 'box_points': [469, 413, 518, 598]}	{'name': 'tie', 'percentage_probability': 87.89281249046326, 'box_points': [1144, 507, 1218, 698]}
11	{'name': 'tie', 'percentage_probability': 66.25762581825256, 'box_points': [1090, 498, 1166, 701]}	{'name': 'tie', 'percentage_probability': 70.08630037307739, 'box_points': [464, 409, 508, 597]}
12	{'name': 'tie', 'percentage_probability': 81.09486699104309, 'box_points': [438, 409, 495, 594]}	{'name': 'tie', 'percentage_probability': 83.81340503692627, 'box_points': [1075, 466, 1125, 677]}
13	{'name': 'tie', 'percentage_probability': 71.14502787590027, 'box_points': [497, 404, 545, 597]}	{'name': 'tie', 'percentage_probability': 92.73524284362793, 'box_points': [1164, 468, 1219, 683]}
14	{'name': 'tie', 'percentage_probability': 92.34132170677185, 'box_points': [474, 403, 527, 600]}	{'name': 'tie', 'percentage_probability': 95.50362825393677, 'box_points': [1180, 470, 1242, 675]}
15	{'name': 'tie', 'percentage_probability': 79.5747458934784, 'box_points': [466, 410, 515, 597]}	{'name': 'tie', 'percentage_probability': 86.86311841011047, 'box_points': [1174, 475, 1235, 674]}
16	{'name': 'tie', 'percentage_probability': 80.79227805137634, 'box_points': [1173, 473, 1237, 672]}	{'name': 'person', 'percentage_probability': 99.89840388298035, 'box_points': [948, 249, 1416, 1059]}
17	{'name': 'tie', 'percentage_probability': 80.86437582969666, 'box_points': [1177, 480, 1242, 665]}	{'name': 'person', 'percentage_probability': 99.92232322692871, 'box_points': [944, 258, 1423, 1051]}
18	{'name': 'tie', 'percentage_probability': 50.28572082519531, 'box_points': [444, 394, 505, 563]}	{'name': 'tie', 'percentage_probability': 94.55930590629578, 'box_points': [1191, 472, 1259, 674]}
19	{'name': 'tie', 'percentage_probability': 89.35491442680359, 'box_points': [1179, 477, 1236, 667]}	{'name': 'tie', 'percentage_probability': 92.51989722251892, 'box_points': [488, 393, 546, 572]}
20	{'name': 'tie', 'percentage_probability': 81.82121515274048, 'box_points': [521, 413, 570, 590]}	{'name': 'tie', 'percentage_probability': 92.65745282173157, 'box_points': [1130, 479, 1200, 667]}

Figure 5.15: *Example of a matrix with the object classes appearances per frame in a random input video. Rows represent frame index, while columns represent the detected objects. Each object is followed by: i. class name, ii. confidence value and iii. bounding box's coordinates. For example in frame with index 0 there were detected 2 persons and one tie.*

1	person	tie
2	2	1
3	2	1
4	1	1
5	1	2
6	1	1
7	2	1
8	2	2
9	2	2
10	2	2
11	2	2
12	2	2
13	2	2
14	2	2
15	2	2
16	2	1
17	2	1
18	2	2
19	2	2
20	2	2

Figure 5.16: *Example of a matrix with the detected objects' instances per frame in a random input video. Rows represent frame index, while columns represent the detected objects. Each object is followed by a number that corresponds to the total number of object's instance in the current frame.*

custom trained model's output layer correspond to the classes of current (custom) data set.

The annotated video is extremely useful for data set enhancement purposes. Additionally, we should highlight the usage of Active Learning techniques during Object Detection, since at the end of the process we have collected information about brand-new training samples

that are contained in the input's frames as well as their precise position. A handy scenario is enhancing the data set with the samples that correspond to high confidence scores. As we have already mentioned, the human expert is always capable of interfering in the data set enhancement procedure by excluding the low-quality image samples.

5.3 Integrated System

As we have already highlighted, the contribution of this thesis includes the implementation of a system that could apply both Face Recognition and Object Detection to an input video, in order to extract as much information as possible in real-time. It is worth mentioning that the frame rate of every input video that undergoes our system is being automatically reduced (from 2 up to 6 frames per second), so that the whole workflow could be speedup, since the reduced frame rate does not affect the recognition/detection accuracy.

Figure 5.17 represents the architecture of the integrated system. Particularly, there are three basic phases:

1. The training phase that involves the data preparation, as well as the classifier training (per model, Face Recognition & Object Detection)

2. The detection/recognition phase, where the system exports the final outputs that include the identified information.

3. The Active Learning phase, where our framework leverages the information that we have extracted from the outputs, in order to enhance the training set.

In Figure 5.18 there is an example of the total information about both the faces and objects that have appeared in the frame of a random input video. In particular, there have been detected 3 persons, but only 2 of them have been recognized (the 3rd one does not belong to the current data set), as well as 2 ties. All the detected entities (persons, faces and objects) have been enclosed in a bounding box.

Furthermore, we succeeded in combining the information that we extracted from both systems, addressing the challenge of faces and objects association that, to the best of our knowledge, has not been implemented yet. More precisely, we built a list of personal objects (i.e wearables such as mask/tie/glasses or objects that a person could use such as cell phone/umbrella/toothbrush) and we utilized the bounding boxes of both faces and personal objects that have appeared in the same frame's region. The output of such an association could be the following system log:

- *"Object with name: {X} belongs to a person with name {Y}"*.

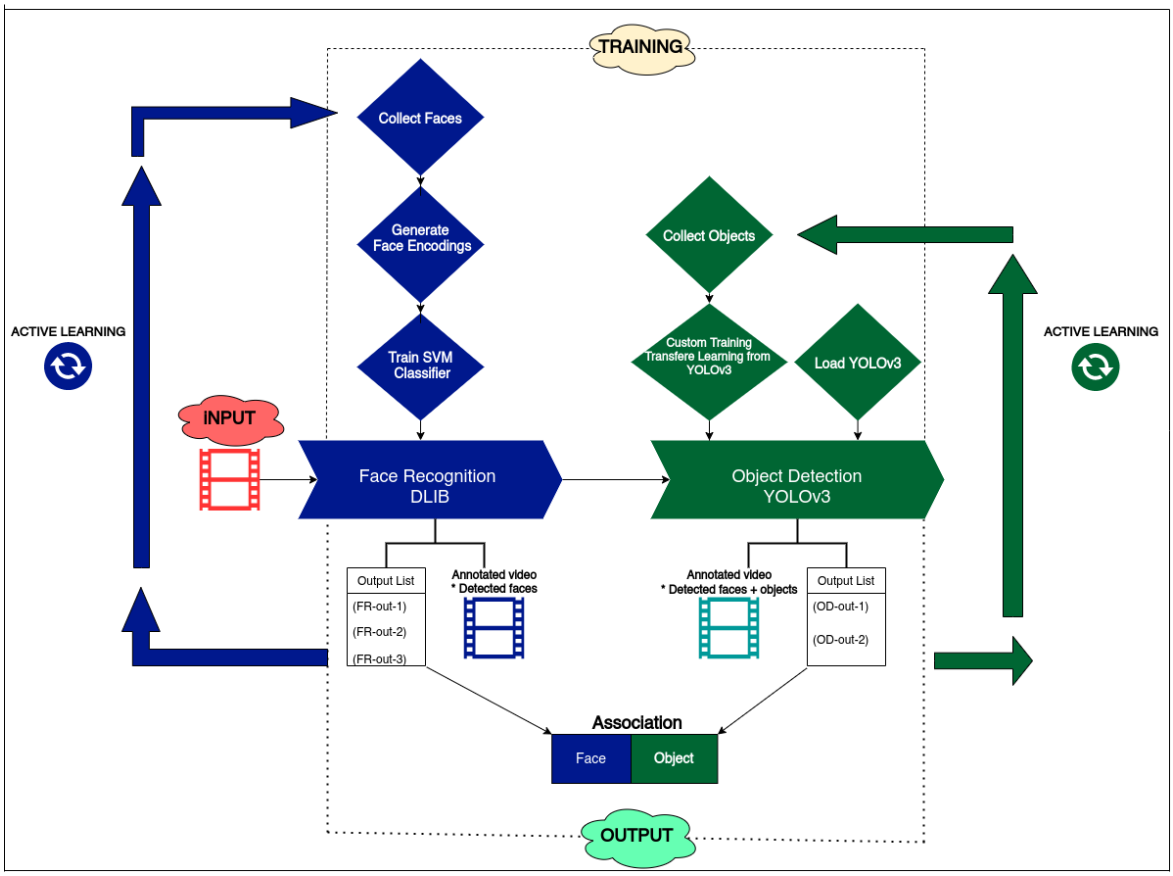


Figure 5.17: *Architecture of the Integrated System. The system is composed of three individual phases: i) the training phase involves the data preparation as well as the classifier training (per model), ii) the detection/recognition phase results in the final outputs (per model) and iii.) the Active Learning phase leverages the identified faces and objects, in order to enhance the training data.*

Making use of the example in Figure 5.18 we extracted the following log:

- *"Object with name: tie belongs to a person with name: Person-1".*

It is definitely admitted, that our metadata repository has a significant role in the whole implementation. Every time we "forward" an input video for Face Recognition and Object Detection, through the integrated system, the system always searches in the records, in order to check if the current input with the pre-defined parameters (e.g number of classes, object detection model, $tolerance_{OD}$ etc.) has already undergone the system.

For example, an input video may have already exported an output for Face Recognition and Object Detection default model, so when we "forward" the video through the system and we define another Object Detection model (e.g custom) then the Face Recognition process is being skipped, in order to save time, since the Face Recognition output that we expect to deliver is already available.

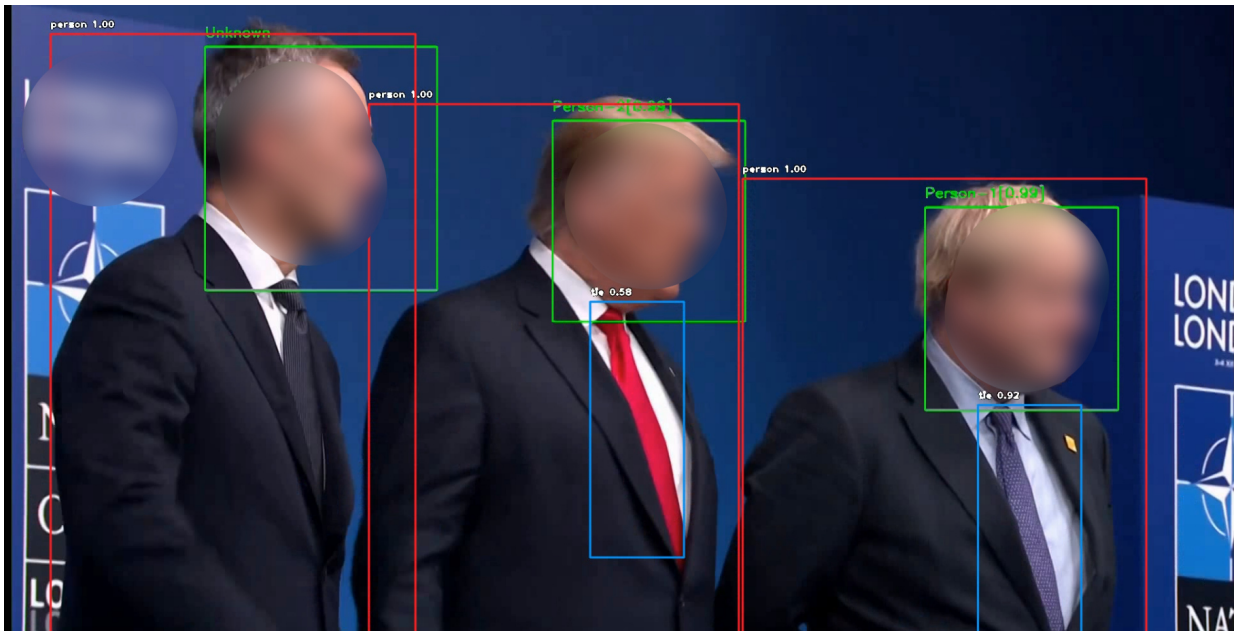


Figure 5.18: *Example of a video frame, where all the detected entities (faces and objects) have been identified and enclosed in a bounding box. The tags "person" and "tie" correspond to objects classes while "Person-1" and "Person-2" indicate that two face classes have been recognized.*

Finally, a new record is going to be added, containing the input's details as well as the system's outputs. Nevertheless, the Object Detection outputs correspond to the current output that have been merged with the existing outputs. Hence, we succeed to extract and preserve the whole information. In the same way, our metadata repository avoids multiple execution of the system for the same input video and pre-defined parameters.

6

Chapter 6: Future Challenges

In conclusion, this thesis provided a comprehensive study on Face Recognition and Object Detection. Face Recognition system was developed by making use of the state-of-the-art open-source Dlib model. For Object Detection the YOLOv3 pre-trained models in association to ImageAI library is recommended, in order to perform predictions on input videos as well as train the system on more object classes that are not included in the COCO data set.

As regards to data construction, in this project we proposed a robust Active Learning framework that creates extended data sets by utilizing only a small amount of manually labeled training data and results in automated labelling of more samples from the unlabelled data. For manual data labelling purposes we highly suggested the open-source annotation tool, OpenCV-Video-Label.

Our integrated system includes the execution of both Face Recognition and Object

Detection models in parallel. The system's outputs are being recorded, while the detections of the "known" entities (faces and objects) enhance the current data set and proceed for further training, since the exported outputs indicate the actual frames and the coordinates of their appearances in the input videos, which have undergone the framework. As regards to the "unknown" faces that appear in a video, the identified information is being utilized, too. As soon as these faces enter the data set we could access a set of labelled samples for the corresponding class (person). As a result, we avoid time-consuming data collection processes for new faces that enter the data set.

Furthermore, we include the human-in-the-loop, in order to provide meaningful guidance to the algorithms when needed. In fact, the human expert is able to control the image samples that enter the training set and ensure that only high-quality samples are being included in the data set.

An innovative contribution of our system involves the implementation of efficient techniques that accomplished the association of Face Recognition with Objects Detection outputs through a workflow. As a result, we have extracted as much knowledge as possible from the input videos, combining the recognized faces, the detected objects as well as the association that may exist between the identified faces and objects.

Moreover, we introduced optimization methods that addressed accuracy issues concerning the Face Recognition flickering phenomenon. In fact, we succeeded in stabilizing the true positive predictions as well as eliminating the false positive predictions. Our implementation involves a sliding window that applies both median and sum filter after defining the corresponding kernel size for each filter. We concluded that the size of the kernel within the median filter should be defined based on the input's frame rate, so that we could ensure the optimal performance of the recognition.

There are several challenges for future research. Firstly, the proposed method addresses only face to objects associations. To this end, further associations between either individual faces or individual objects could be incorporated, in order to extract statistics about persons or objects common appearance in videos.

Secondly, we plan to include more features in our system's capabilities, in order to satisfy more use-cases. For instance, we could incorporate an Image Prediction (Image Classification) model, so that we can automatically generate tags for each input's frames. ImageAI support such a model, which builds on the pre-trained weights of the ResNet50 Neural Network. As soon as we have generated tags for every frame we could investigate the

tag variability through the frames and, in addition, determine a possible association that may exist between the frames' contents (that is, persons and objects) and the generated tag.

In the same way, we could apply Color Analysis [32] to the input's frames and extract the N dominant colors that prevail all over the video. For Color Analysis purposes we could use a Machine Learning clustering algorithm (e.g K-means). K-means starts from K random pixels-centroids and at the end of the clustering process results in N clusters. These clusters correspond to the N dominant colors of the frame. The output palette of the Color Analysis model would enable us to predict the change of the scenery/background in the individual video's parts.

As a result, a combination of Face Recognition, Object Detection, Color Analysis and Image Prediction corresponds to a complete Image Analysis procedure, which results in a detailed description of video frames. Ultimately, we could correlate the scenery/background and tag variability as well as the frame's contents.

Additionally, a more sophisticated Deep Learning approach could further develop both Face Recognition and Object Detection systems. For instance, they could eliminate the inconsistency concerning the confidence values within Face Recognition classification tasks, when the class number is extremely low (e.g < 5). As regards the Object Detection system, every exported custom model should extend the latest model's output layer, so that we can avoid the multiple system executions for the same input in an attempt to detect all the newly added classes.

Lastly, we could perform a grid search concerning the Face Recognition prediction filtering, in order to further enhance the recognition accuracy. In particular, the kernel size that we mentioned in steps 2 and 3 (see Algorithm 1) as well as every pre-define threshold value $\{threshold_A,\ cutoff\}$ could be further investigated. Also, there is a large set of threshold values $\{threshold_B,\ threshold_C,\ tolerance_{OD}\}$ that have been utilized for both Face Recognition and Object Detection experiments. As it has been proven, these thresholds affect the system's performance. Thus, we could incorporate comparisons between different values and point out the most efficient combinations that result in the optimal recognition results.

7 APPENDIX

7.1 Face Recognition

1. **Python Source Code: Align the faces in the data set**

```
# loop over the face detections
for input_file in input_files:
    try:
        image = cv2.imread(input_file)
        image = imutils.resize(image, width=450)
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # show the original input image and
        # detect faces in the grayscale
        rects = detector(gray, 1)
        for rect in rects:
            # extract the ROI of the *original* face
            # then align the face using facial landmarks
            try:
```

```
                faceAligned = fa.align(image, gray, rect)
            except:
                print("CANNOT SAVE")
                continue
    except:
        pass
```

2. **Python Source Code: Create a 3-D matrix consisting of the training samples**

```
for (i, imagePath) in enumerate(imagePaths):
  # extract the person name from the image path
  name = imagePath.split(os.path.sep)[-2]

  # load the input image and convert it from RGB (OpenCV ordering)
  # to dlib ordering (RGB)
  image = cv2.imread(imagePath)
  rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
  short_img_path = '/'.join(imagePath.split('/')[-2:])
  # check if pickle is empty and append encodings (if they don not
   already exists)
  if os.path.getsize(encodings_path) > 0:
    data = pickle.loads(open(encodings_path, "rb").read())
    # loop over the encodings
    if short_img_path in data["image-path"]:
      print(f"{short_img_path} already exists in database so it has
   been skipped...")
    else:
      # detect the (x, y)-coordinates of the bounding boxes
      # corresponding to each face in the input image
      boxes = face_recognition.face_locations(rgb,model="cnn")

      # compute the facial embedding for the face
      encodings = face_recognition.face_encodings(rgb, boxes)

      if encodings != []:
        # add each encoding + name to our set of
        # known names and encodings
        imageDirs.append(short_img_path)
        knownEncodings.append(list(encodings[0]))
        knownNames.append(name)
        counter += 1
        print(f"{short_img_path} encoding have been
        "successfully inserted in database!")

  else:
    # detect the (x, y)-coordinates of the bounding boxes
```

```
    # corresponding to each face in the input image
    boxes = face_recognition.face_locations(rgb, model="cnn")
    # compute the facial embedding for the face
    encodings = face_recognition.face_encodings(rgb, boxes)

    if encodings != []:
      # add each encoding + name to our set of
      # known names and encodings
      imageDirs.append(short_img_path)
      knownEncodings.append(list(encodings[0]))
      knownNames.append(name)
      counter += 1
      print(f"{short_img_path} encoding have been successfully
   inserted in database!")

# dump the image path + facial encodings + names to disk
data = {"image-path": imageDirs, "encodings": knownEncodings,
"names": knownNames}
f.write(pickle.dumps(data))
```

3. **Python Source Code: Train one or more Machine Learning classifiers (either SVM or KNN)**

```
# define the maximum class number per classifier
# in order to ensure good results
max_classes_per_clf = 20
# define the ML classifier that is going to be trained
recognizer = "svm" # alternative  "knn"

# check if class number exceeds the pre-defined max_classes_per_clf
# and split the classes in equal parts
if (number_of_classes / max_classes_per_clf).is_integer():
    num_of_recognizers = int(number_of_classes /
    max_classes_per_clf)
else:
    number_of_recognizers = int(number_of_classes /
    max_classes_per_clf) + 1

for rec in range(number_of_recognizers):
    classes_to_train = list(chunks(unique_classes,
    number_of_recognizers))[rec]
    embeddings_to_train = [e for e in embeddings if
    labels[embeddings.index(e)] in classes_to_train]
    labels_to_train =  [l for l in labels if l in classes_to_train]

    # initialize classifier
    # train the model with the 128-d embeddings of the face
```

```
    recognizer = None
    if recognizer_type == "svm":
        print(f"Training SVM classifier #{rec + 1}..." )
        recognizer = OneVsRestClassifier(SVC(C=1, kernel="linear",
        probability=True, gamma='auto'), n_jobs=-1)
    elif recognizer_type == "knn":
        k = len(np.unique(labels))
        print(f"Training KNN (k = {k}) classifier #{rec + 1}...",)
        recognizer = OneVsRestClassifier(KNN(k))

    recognizer.fit(embeddings_to_train, labels_to_train)
    f = open(recognizer_path, "wb")
    f.write(pickle.dumps(recognizer))
    f.close()
```

4. **Python Source Code: Analyse the input's frames, in order to generate confidence scores and save the identified information (frames, coordinates, confidence scores) in a data frame**

```
# loop over frames from the video file stream
while True:
    # grab the next frame
    (grabbed, frame) = stream.read()

    # if the frame was not grabbed, then we have reached the
    # end of the stream
    if not grabbed:
        break

    # detect the (x, y)-coordinates of the bounding boxes
    # corresponding to each face in the input frame,
    # then compute the facial embeddings for each face
    boxes = face_recognition.face_locations(frame, model='cnn')
    encodings = face_recognition.face_encodings(frame, boxes)

    # initialize empty dictionary to insert keys (class name)
    # and values (coordinates + confidence)
    dictionary = {}
    # loop over the facial embeddings
    for encoding, box in zip(encodings, boxes):
        # confidence list for each encoding detected on the
        # current frame append confidence values for each class
        predictions = []
        confidence = []
        # reshape needed in order to transform to (1,128) embedding
        for recognizer in recognizer_list:
            predictions.append(recognizer.predict_proba(encoding
```

```
            .reshape(1, 128))[0])

        for i in range(len(predictions)):
            for pred in predictions[i]:
                confidence.append(pred)

        # loop over names (classes) to append coordinates
        # & confidence values per encoding in frame
        # in order to construct dictionary
        for i in range(len(lbl_list)):
            key = lbl_list[i]
            # extract top, right, bottom, left from coordinates and
            # concatenate with confidence value
            # each cell is a list of numbers per class
            value = []
            for coordinates in box:
                value.append(int(coordinates))
            value.append(round(confidence[i], 5))
            # check if key exists in dictionary or current key's
            # confidence value is smaller than existing
            # then continue and do not update value
            if key not in dictionary or \
             dictionary[key][-1] < value[-1]:
                dictionary[key] = value
            else:
                continue

            if np.max(confidence) < unknown_thres:
            unknown_encodings.append(encoding)
            unknown_coords.append(box)
            unknown_offset.append(str(frame_num) +
            "_" + str(frame_num / fps))

    # create dataframe with classes as keys &
    # [coordinates + confidence] as values
    keys = list(dictionary.keys())
    values = list(dictionary.values())

    df_per_frame = pd.DataFrame({"name": keys, \
     "confidence": values})
```

5. **Python Source Code: Filter the confidence scores (within every input's data frame), in order to eliminate flickering**

```
def filter(array_cp, cutoffA, cutoffB, median_window, sum_window):
    # STEP-1: zeroing values under cutoff
    for i in range(array_cp.shape[0]):
```

```
        for j in range(array_cp.shape[1]):
            if array_cp[i][j] != 0:
                conf = array_cp[i][j][-1]
                if conf > cutoffA:
                    array_cp[i][j].append(conf)
                else:
                    array_cp[i][j].append(0)
            else:
                continue

    # STEP-2: median filtering
    for i in range(array_cp.shape[0]):
        for j in range(median_window, array_cp.shape[1]
        - median_window):
            if array_cp[i][j] != 0:
                array_cp[i][j].append(np.median(find_conf(
                    array_cp[i][j - median_window:j
                    + median_window])))
            else:
                continue

    # STEP-3: sum filtering
    filtered_array = copy.deepcopy(array_cp)
    for i in range(array_cp.shape[0]):
        for j in range(sum_window, array_cp.shape[1] - sum_window):
            if filtered_array[i][j] != 0:
                filtered_array[i][j].append(np.sum(find_conf(
                    array_cp[i][j - sum_window:j + sum_window])))
            else:
                continue

    # STEP-4: final thresholding (not for edges)
    for i in range(filtered_array.shape[0]):
        for j in range(sum_window, filtered_array.shape[1]
        - sum_window):
            if filtered_array[i][j] != 0:
                conf = filtered_array[i][j][-1]
                if float(conf) > float(cutoffB):
                    filtered_array[i][j].append(0.99)
                else:
                    filtered_array[i][j].append(0.01)
            else:
                continue

    return filtered_array
```

6. **Python Source Code: Seek for old appearance (frames in association to coordinates) of a new class that enters the data set**

```
match_threshold = 0.5

# isolate new classes encodings
for i, lbl in enumerate(data["names"]):
    if lbl in new_classes:
        new_encodings.append(list(data["encodings"][i]))
        new_names.append(data["names"][i])

# load unknown encoding that have been identified in recent videos
# load new class encodings that has entered the dataset
# estimate the distance between all new classes encodings
# and all previously unknown encodings
count = 1
for path in unknown_encodings_paths:
    logs = []
    # define all combinations new-unknown
    combinations = [[x, y] for x in new_encodings
    for y in unknown_encodings]
    for c in combinations:
        if np.linalg.norm(np.array(c[0])
        - np.array(c[1])) < match_threshold:
            new_class_idx = new_encodings.index(c[0])
            unknown_encoding_idx = unknown_encodings.index(c[1])
            frame = unknown_frames[unknown_encoding_idx]
            offset = unknown_offset[unknown_encoding_idx]
            coords = unknown_coords[unknown_encoding_idx]
            logs.append([int(frame),
                f"-{new_names[new_class_idx]} has appeared in video:
                {video_id} \n -> frame: {frame} - offset: {offset}
                - coordinates: {coords}"])
```

7.2 Object Detection

1. **Python Source Code: Train the system**

```
trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory=data_directory)
trainer.setTrainConfig(object_names_array=new_class_name, batch_size
    =batch_size, num_experiments=epochs,
                       train_from_pretrained_model=pretrained_model)
trainer.trainModel()
```

2. **Python Source Code: Evaluate the exported trained models**

```
trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory=new_class_name)
metrics = trainer.evaluateModel(model_path=model_path, json_path=
    json_path, iou_threshold=0.5, object_threshold=0.3,
                                    nms_threshold=0.5)
```

3. **Python Source Code: Analyse the input's frames, in order to generate probabilistic predictions and save the identified information (frames, coordinates, probabilities) in a data frame**

```
def run_object_detection(input_path, fps, tolerance,
    detection_model_weights):
     detector = VideoObjectDetection()
     # set model type: RetinaNet, YOLOv3, and TinyYOLOv3.
     detector.setModelTypeAsYOLOv3()
     # models: resnet50_coco_best_v2.0.1.h5 / yolo.h5 / yolo-tiny.h5
     detector.setModelPath(detection_model_weights)
     # detection_speed= fast / faster/ fastest
     detector.loadModel(detection_speed="fast")

     # Function returns an array of dictionaries with each dictionary
     # corresponding to the number of objects detected in the image
     # Each dictionary has the name of the object, the
     # percentage_probability of the detection
     # and box_points (the x1,y1,x2 and y2 coordinates of the
    bounding box of the object).
     detector.detectObjectsFromVideo(input_file_path=input_path,
         output_file_path=object_output_video_path,
         frames_per_second=fps,
         minimum_percentage_probability=tolerance,
         video_complete_function=forFull)

     return object_output_video_path, skip_process
```

4. **Concatenate individual outputs of single input's multiple executions, in order to maintain the total identified information**

```
# function that concatenate multiple raw files
def concate_raw(objects_raw_to_merge):
     df_dict = {}
     merged = pd.DataFrame()
     # list non empty raw files
     non_empty_raw = [o for o in objects_raw_to_merge if os.stat(o).
    st_size>1000]
```

```
    # define merged df rows
    df_rows = pd.read_csv(non_empty_raw[0], index_col=0).shape[0]
    # initialize stop column list per row
    df_stop_idx = [0] * (df_rows + 1)

    # loop through raw files
    for df_idx, file in enumerate(objects_raw_to_merge):
        if os.path.exists(file) and os.stat(file).st_size>100:
            if df_idx == 1:
                print()
                print("[INFO] Generate merged raw file
                 for all object detection models...")

            df_dict[df_idx] = pd.read_csv(file, index_col=0)
            current_df = df_dict[df_idx]

            # build the merged df
            for row in range(current_df.shape[0]):
                # update stop column
                current_stop = df_stop_idx[row]
                if df_idx == 0:
                    for col, cell in enumerate(current_df.
                    iloc[row, :]):
                        if is_nan(cell):
                            df_stop_idx[row] = col
                        else:
                            merged.loc[row, col] = str(cell)
                            df_stop_idx[row] = col + 1
                else:
                    for col, cell in enumerate(current_df.
                    iloc[row, :]):
                        if is_nan(cell):
                            df_stop_idx[row] = col
                        else:
                            merged.loc[row, current_stop + col]
                            = str(cell)
                            df_stop_idx[row] = col + 1
        os.remove(file)

    # sort merged df rows
    merged = merged.sort_index()
    merged_path = os.path.join(objects_raw_to_merge[0].
    split('raw_objects')[0]) + "raw_objects_merged.csv"
    merged.to_csv(merged_path, index=True)

# function that concatenate multiple count files
```

```
def concate_count(objects_count_to_merge):
    merged = pd.concat([pd.read_csv(f).drop(pd.read_csv(f).columns[0], axis=1) for f in objects_count_to_merge if os.stat(f).st_size>100], sort=False, axis=1)

    for file in objects_count_to_merge:
        os.remove(file)

    object_detection_models = "_".join([f"{o.split('count_objects_')[-1].split('.csv')[0]}" for o in objects_count_to_merge])
    merged_path = os.path.join(objects_count_to_merge[0].split('count_objects')[0]) + f"count_{object_detection_models}.csv"
    merged.to_csv(merged_path, index=False, encoding='utf-8-sig')

    return merged_path
```

7.3 Face-Object Association

Python Source Code: Generate associations between the recognized entities (faces & objects)

```
def associate(video_id, object_detection_model, fps, face_classes,
    personal_objects = ["tie", "mask", "hat", "glasses", "watch", "suitcase", "handbag", "backpack", "umbrella"]

  if os.path.exists(objects_raw_file) and os.path.exists(faces_raw_file):
        objects_raw_df = pd.read_csv(objects_raw_file, sep=",")
        with open(faces_raw_file) as json_file:
            known_faces_dict = json.load(json_file)
            known_names = [known_faces_dict[str(i)]['name'] for i
            in range(len(known_faces_dict))]
            known_frames = [known_faces_dict[str(i)]['index'] for i
            in range(len(known_faces_dict))]
            known_offset = [known_faces_dict[str(i)]['offset'] for i
            in range(len(known_faces_dict))]
            known_coords = [known_faces_dict[str(i)]['coordinates'] for i
            in range(len(known_faces_dict))]

        f = open(logs_filepath, 'w')
        for frame_num in range(objects_raw_df.shape[0]):
            personal_objects_per_frame = []
            objects = []
            boxes = []
            person_box = []
            for item in range(1, objects_raw_df.iloc[frame_num, :]
```

```
            .shape[0]):
                object_in_frame = objects_raw_df.iloc[frame_num, item]
                if is_nan(object_in_frame):
                    continue
                else:
                    objects.append(ast.literal_eval(object_in_frame)
                    ["name"])
                    boxes.append(ast.literal_eval(object_in_frame)
                    ["box_points"])
            objects_cp = copy.deepcopy(objects)
            objects_cp = list(filter(("person").__ne__, objects_cp))
            if "person" in objects and objects_cp != []:
                person_indices = [[index for index, element in
                enumerate(objects) if element == "person"]]
                for pi in person_indices[0]:
                    person_box.append(boxes[pi])
                for obj in objects_cp:
                    idx = objects.index(obj)
                    for p in range(len(person_box)):
                        if obj in personal_objects and
                        contains(person_box[p], boxes[idx]) == True:
                            # ----------- faces usage -------------- #
                            for k in range(len(known_names)):
                                if frame_num in known_frames[k]:
                                    index = known_frames[k]
                                    .index(frame_num)
                                    face_in_frame = known_names[k]
                                    box_in_frame = [known_coords[k]
                                    ["left"][index],
                                    known_coords[k]["top"][index],
                                    known_coords[k]["right"][index],
                                    known_coords[k]["bottom"][index]]
                                    if contains(person_box[p],
                                    box_in_frame) == True:
                                        personal_objects_per_frame
                                        .append([frame_num,
                                        f"Object with name:
                                        '{obj}' belongs to a
                                        person with name:
                                        {face_in_frame}"])
                    # update objects list by removing the current object
                    del objects[idx]
                    del boxes[idx]
                    objects_cp = copy.deepcopy(objects)
                    objects_cp = list(filter(("person").__ne__,
                    objects_cp))
```

Bibliography

[1] J. Brownlee, "A Gentle Introduction to Computer Vision." Machine Learning Mastery, 5 July 2019, machinelearningmastery.com/what-is-computer-vision/. xi, 1, 6

[2] S. Balaban, "Deep learning and face recognition: the state of the art," Biometric and Surveillance Technology for Human and Activity Identification XII, 2015. 1

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 1, pp. 142–158, 2016. 1

[4] O. Sener and S. Savarese, "Active Learning for Convolutional Neural Networks: A Core-Set Approach," arXiv.org, 01-Jun-2018. [Online]. Available: https://arxiv.org/abs/1708.00489. [Accessed: 13-Dec-2020]. 1

[5] D. Mishra, "Active Learning-Say Yeah!" Medium, Towards Data Science, 20 Oct. 2020, towardsdatascience.com/active-learning-say-yeah-7598767806b2. xi, 1, 17

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015. 7

[7] E. Learned-Miller, G. B. Huang, A. Roychowdhury, H. Li, and G. Hua, "Labeled Faces in the Wild: A Survey," Advances in Face Detection and Facial Image Analysis, pp. 189–248, 2016. 7

[8] T. Baltrusaitis, P. Robinson, and L.-P. Morency, "OpenFace: An open source facial behavior analysis toolkit," IEEE Winter Conference on Applications of Computer Vision (WACV), 2016. 7

[9] S. Suwarno and K. Kevin, "Analysis of Face Recognition Algorithm: Dlib and OpenCV," Journal Of Informatics And Telecommunication Engineering, vol. 4, no. 1, pp. 173–184, 2020. 7

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. 7

[11] A. Geitgey, "Machine Learning Is Fun! Part 4: Modern Face Recognition with Deep Learning." Medium, Medium, 24 Sept. 2020, medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78. xi, 8, 10

[12] L. Fei-Fei, J. Deng, and K. Li, "ImageNet: Constructing a large-scale image database," Journal of Vision, vol. 9, no. 8, pp. 1037–1037, 2010. 9

[13] D. Nelson, "How Does Image Classification Work?," Unite.AI, 05-Sep-2020. [Online]. Available: https://www.unite.ai/how-does-image-classification-work/. [Accessed: 16-Dec-2020]. 9

[14] P. Hurtik, V. Molek, and P. Vlasanek, "YOLO-ASC: You Only Look Once And See Contours," International Joint Conference on Neural Networks (IJCNN), 2020. 9

[15] Lin, Tsung-Yi, et al. "Microsoft COCO: Common Objects in Context." Microsoft Research, 17 Oct. 2018, www.microsoft.com/en-us/research/publication/microsoft-coco-common-objects-in-context/. 9

[16] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015. 10

[17] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," International Journal of Computer Vision, vol. 88, no. 2, pp. 303–338, 2009. 10

[18] A. F. Gad, "Faster R-CNN Explained for Object Detection Tasks," Paperspace Blog, 13-Nov-2020. [Online]. Available: https://blog.paperspace.com/faster-r-cnn-explained-object-detection/. [Accessed: 14-Dec-2020]. 10

[19] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," arXiv.org, 24-Jan-2018. [Online]. Available: https://arxiv.org/abs/1703.06870. [Accessed: 14-Dec-2020]. 10

[20] Cagriyoruk. "Cagriyoruk/State-of-the-Art-Object-Detection-Models." GitHub, github.com/Cagriyoruk/State-of-the-art-Object-Detection-Models.

[21] S. Nayak, "Deep Learning Based Object Detection Using YOLOv3 with OpenCV (Python / C++): Learn OpenCV." Learn OpenCV — OpenCV, PyTorch, Keras, Tensorflow Examples and Tutorials, 5 June 2020, www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/.

[22] B. Bradtke, "Social Network for Programmers and Developers." RSS, morioh.com/p/278ba1bb7711. xi, 13

[23] D. Gordon, A. Farhadi, and D. Fox, "Re^3: Real-Time Recurrent Regression Networks for Visual Tracking of Generic Objects," IEEE Robotics and Automation Letters, vol. 3, no. 2, pp. 788–795, 2018. 19

[24] G. Nebehay and R. Pflugfelder, "Clustering of static-adaptive correspondences for deformable object tracking," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015. 19

[25] A. Rosebrock, "Facial landmarks with dlib, OpenCV, and Python," PyImageSearch, 18-Apr-2020. [Online]. Available: https://www.pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/. [Accessed: 14-Dec-2020]. 20

[26] H. Li, P. Wang, and C. Shen, "Robust Face Recognition via Accurate Face Alignment and Sparse Representation," International Conference on Digital Image Computing: Techniques and Applications, 2010. 21

[27] D. Wang, C. Otto, and A.K. Jain, "Face Search at Scale: 80 Million Gallery." ArXiv.org, 28 July 2015, arxiv.org/abs/1507.07242.

[28] C. Holger, J. Uijlings and V. Ferrari, "COCO-Stuff: Thing and Stuff Classes in Context." 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, doi:10.1109/cvpr.2018.00132. xi, 24

[29] S. Zhang and H. Qiao, "Face recognition with support vector machine," IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, vol.2, pp. 726-730, 2003. 26

[30] E. Setiawan and A. Muttaqin, "Implementation of K-Nearest Neighbors Face Recognition on Low-power Processor," TELKOMNIKA (Telecommunication Computing Electronics and Control), vol. 13, no. 3, p. 949, 2015. 26

[31] J. Qingge, H. Jie, H. Wenjie, S. Yankui, "Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images." Algorithms, vol. 12, no. 3, 2019, p. 51., doi:10.3390/a12030051

[32] T. Lertrusdachakul, K. Ruxpaitoon and K. Thiptarajan, "Color Palette Extraction by Using Modified K-means Clustering," 2019 7th International Electrical Engineering Congress (iEECON), Hua Hin, Thailand, 2019, pp. 1-4, doi: 10.1109/iEECON45304.2019.8938867. 51

[33] M. Nedrich, “Palette Maker - A tool for Extracting Color Palettes From Images”. Atomic Spin, 30 Dec. 2017, spin.atomicobject.com/2016/12/07/pixels-and-palettes-extracting-color-palettes-from-images/.