

University POLITEHNICA of Bucharest
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND
INFORMATION TECHNOLOGY

Demo application for real-time person identification

Diploma Thesis

submitted in partial fulfilment of the requirements for the
Degree of Engineer
in the domain *FACULTY OF ELECTRONICS,*
TELECOMMUNICATIONS AND INFORMATION TECHNOLOGY
study program *Telecommunications*

Thesis advisor(s)

Conf. dr. ing. Horia CUCU

Ing. Gabriel Sandu

Student

Gabriel Octavian GALER

Year 2021

DIPLOMA THESIS

of student **GALER M.M. Gabriel-Octavian , 442G-TST**

1. Thesis title: Demo application for real-time person identification

2. The student's original contribution will consist of (not including the documentation part) and design specifications:

Currently, the task of person identification using their pictures and/or audio materials separately is being approached in many ways. The project aims to tackle the task of identifying people in a multimodal way using both images and speech from the selected person. The project will consist in a Python application that will identify in real time the person speaking in the video and the possible change of this person.

The Speech and Dialogue Research Laboratory (Speed) has expertise in identifying people based on voice biometrics and has generated several speaker identification databases [1]. During 2019, Speed obtained the database of video recordings and transcripts of the sittings of the Chamber of Deputies from 1993 to 2018 [2] [3]. This database contains over 2500 persons for whom there are audio-video recordings. In the academic year 2019-2020 the data set was used to create a Deep Learning (DL) model for multimodal identification of people [4].

In this context, the present project aims to integrate the DL system previously developed in a demonstration application for real-time identification of people speaking at the desk in the videos broadcast live from the Chamber of Deputies.

In addition, the project can also address the updating of neural networks used in the multimodal person identification system. Also, given that there will be parliamentary elections in 2020, the system could also aim to update the database of persons.

References

[1] A.L. Georgescu, A. Caranica, H. Cucu, C. Burileanu, "RoDigits - a Romanian connected-digits speech corpus for automatic speech and speaker recognition," UPB Scientific Bulletin, Series C, vol. 80, issue 3, pp. 45- 62, Bucharest, 2018.

[2] Data agreement CDep: <http://www.cdep.ro/pls/dic/site2015.page?id=794>

[3] Example video CDep: <http://www.cameradeputatilor.ro/pls/steno/steno2015.sumar?ids=8027>

3. Academic courses the thesis is based on::

PC, SDA, POO, PI, ISIA, RFIA

4. Thesis registration date: 2020-11-23 08:48:27

Thesis advisor(s),

Conf. dr. ing. Horia CUCU



Ing. Gabriel Sandu

Department director,

Conf. dr. ing. Șerban OBREJA

Student,

GALER M.M. Gabriel-Octavian



Dean,

Prof. dr. ing. Mihnea UDREA

Statement of Academic Honesty

I hereby declare that the thesis "*Demo application for real-time person identification*", submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of Engineer of Science in the domain *ELECTRONICS, TELECOMMUNICATIONS AND INFORMATION TECHNOLOGY*, study program *Telecommunications*, is written by myself and was never before submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations or measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations or measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest,
29.06.2021

Gabriel Octavian GALER


(student's signature)

Table of contents

List of figures	iii
List of tables	iv
List of abbreviations	v
1. Introduction	1
1.1. Current state of the project	2
1.2. Goals for the thesis	2
2. Theoretical concepts	4
2.1. Machine learning. Deep learning	4
2.1.1. Categories of problems in machine learning	5
2.1.2. Image classification	5
2.1.3. Nearest Neighbor	7
2.1.4. Linear classification and SVM	8
2.1.5. Haar cascades	9
2.1.6. Neural networks	11
2.1.7. Gradient Descent	13
2.1.8. Regularization methods	13
2.1.9. Convolutional Neural Networks	14
2.1.10. Non-maximum supression	18
2.1.11. R-CNN and iterations	18
2.1.12. YOLO neural network	20
2.2. Development	21
2.2.1. Python	21
2.2.2. Tensorflow	23
2.2.3. HLS protocol	23
2.2.4. WSL	23
2.2.5. X11 server and xcb	24
2.2.6. Mobaxterm	24
2.2.7. Virtual environments	24
2.2.8. ffmpeg	25
2.2.9. Markdown	25
2.2.10. git	25

3. The neural network integration	26
3.1. Shortcomings of the raw implementation	26
3.1.1. Bad face samples	26
3.1.2. No undefined class	27
3.1.3. Inference time in a blocking flow	27
3.1.4. Other persons talking over	27
3.1.5. Masks covering faces during the pandemic	28
3.2. Improvements & additions	28
3.2.1. Creating the dataset	29
3.2.2. Analyzing kNN for plenum detection	31
3.2.3. Implementing a CNN classifier for plenum detection	31
3.2.4. Implementing the YOLO network	32
4. Application development	34
4.1. Script for testing inference	34
4.2. Script for audio trimming	35
4.3. Adapt input data to network's input format	37
4.4. Script for adding rectangle on frame	37
4.5. Software design decisions	39
4.5.1. Using <code>concurrent.futures</code>	39
4.6. Handling stream input	40
5. Conclusions	42
5.1. Personal contributions	42
5.2. Further steps for development	42
5.3. Final words	43
Bibliography	44
Anexa A. Code for extracting samples for the training dataset	46
Anexa B. FileCleaner class	48
Anexa C. Video class	49
Anexa D. VideoOptions class	57
Anexa E. Logger class	58
Anexa F. YOLO predictor	60

List of figures

2.1. Perspective and lighting variations on the same subject	6
2.2. COCO competition metrics	7
2.3. COCO AP versus inference time for state-of-the-art object detection systems . .	8
2.4. Example of image classification with only one neuron	9
2.5. Haar convolutional kernels	10
2.6. Graphical explanation of the integral images	11
2.7. Mathematical model of a neuron	12
2.8. Illustration of a deep learning model	13
2.9. Example of dropout	14
2.10. Example of samples from the CIFAR-10 dataset	15
2.11. Visual representation of a CNN	16
2.12. Graphical example of sparse connectivity	17
2.13. Example of NMS	18
2.14. R-CNN algorithm example	19
2.15. Segmentation example	19
2.16. Example of how YOLO does a prediction	20
2.17. Architecture of YOLO	21
2.18. Flowchart of the desired frame handler	22
2.19. WSL arhitecture	23
2.20. Example of a Markdown log file	25
3.1. Bad face sample	26
3.2. Wrong face samples provided by Haar cascades	27
3.3. An example of an identification when a person is talking over	28
3.4. Politician wearing mask	28
3.5. Camera angles existing in a recording	29
3.6. A proposal for a good crop region which removes unwanted objects	30
3.7. The final dataset for a plenum classifier	30
3.8. Used CNN architecture for the plenum classifier	32
4.1. Video player main function	35
4.2. Frame handler flowchart	36
4.3. Banner showing politician's identity	38
4.4. Pseudocode for obtaining a well-scaled text	38
4.5. Event loop diagram of the application	40
4.6. Example of chunk list from a CDep resource stream	41

List of tables

3.1. Train and test accuracy comparison between different values for k	31
3.2. Train and test accuracy for the plenum CNN	32

List of abbreviations

WAV = Waveform Audio File
FC = Fully Connected
NMS = Non-Maximum Suppression
RPN = Region Proposal Network
IoU = Intersection Over Union
RoI = Region of Interest
FCNN = Fully Convolutional Neural Network
CNN = Convolutional Neural Network
OCR = Optical Character Recognition
MLP = Multi-layer Perceptron
SVM = Support Vector Machine
DNN = Deep Neural Network
STFT = Short-time Fourier Transform

Chapter 1

Introduction

Face and object recognition is undeniably one of the most important task resulted from today's technological advances. A reason for that would be the large amount of unlabeled photos and videos which circulates the internet. It is also true that people tend to prefer to watch a video which summarizes an article then read the article itself, so it motivates developers to create applications that automatically recognize and label information. Recognizing and understanding the objects which are present in a video is a very fast and precise process for us humans, but not so much for computers. In 1996, a study was showed that the reaction time for superordinate tasks, such as detecting human faces, took around 250–290 ms [1]. In a time when computing power is the cheapest it has ever been and ground-breaking researches are achieved more and more frequently, this task becomes easier and easier for computers.

In the political domain, where many recordings are generated from various meetings, a system like the one described can save a large amount of time that would otherwise be used for manual labelling. Also, speech recognition can greatly enhance this process by generating text in real-time from the audio. These two implementations combined can create digital reports of the meetings, which can be later accessed by politicians or other people.

Stenography papers will no longer be necessary and notes will be moved to the digital format, thus discarding the difficult task of translating them back to natural language. Phrases will be easier to synchronize and to timestamp this way since it will be an automatic, real-time process. Also, adding banners with each speaker will not require human intervention anymore. When various identification errors appear, they can be corrected later, since each software decision and prediction will be logged.

As for the reinforcement process of the neural network, acquisition and filtering of additional data can be done in parallel, thus creating datasets for later improvements, which will lead to greater accuracy of the system.

So, it can be seen how such a system will save time for the recording processing and the delivery of the post-processed meeting videos.

Looking closer at face recognition in particular, research papers which aimed for this goal first appeared in 2001 and achieved excellent results at that time [2], making the resulted algorithms to be a good proposal for digital cameras at that time.

The broadcasting and news industry can greatly benefit from this development, in the sense that it can save time and effort from the tedious process of applying banners with the currently speaking person in the video.

1.1 Current state of the project

This work is part of a bigger project of Speech and Dialogue research laboratory (<https://speed.pub.ro>) that started in 2019, with the goal of implementing a person identification system for the CDep's (Chamber of Deputies) meetings, which are held in the Palace of the Parliament. Having the potential of a successful neural network-based application, this project started with the acquisition of the raw data that resulted in a large dataset consisting of video recordings, audio tracks and stenograms of the speeches. Previous work can be analyzed in detail in Gabriel Sandu's [3] and Cristian Manolache's [4] diploma thesis.

Further work was added in the next year, 2020, when a multimodal neural network was designed based on popular networks such as VGGVox for audio feature extraction and FaceNet for image feature extraction. These two extractors are later bound, using late binding, to form the final classifier.

Data processing was an important process here since the only data available was the raw one, so databases with desired formats were needed for this development.

A big portion of the time was invested in creating scripts which validated each input from these created databases. The final format for the input was a sequence of bytes which represented a 3-second audio recording which would include the voice of a person, and an image which would include the face of the speaker. From these two inputs, the person's identity would be predicted, that is, if that person was included in the training process of the network.

After the architecture of the network was done, the training process consisted of 2 experiments, one with 10 multimodal inputs per class with 257 classes and the other with 50 inputs per class with 132 total classes, which ran for 10 epochs. The results were very promising, the 2nd experiment yielding an accuracy of 99.9%.

Gathering all these results, further progress in the development chain can be done by using these results to build a demo application which will showcase the neural network's ability to aid in the identification process.

1.2 Goals for the thesis

Starting from the trained neural network described in the last section, the goal is to build an application which will apply identity banners containing the name of the person for which a prediction was made, when some conditions are met.

For this project, these conditions are based on whether only one person is present in the frame or not. Other conditions that could be added would be a voice activation detector, background checks of the frame (since the background is different when there is only one speaker in the recording can be easily determined when looking at the picture) etc. The last specified conditions would introduce more overhead for the frame processing, making the prediction slower, and would be redundant since great results were achieved with just the check on the number of faces in the frame.

The final, ideal application would accept both video recordings and stream URLs for its inputs, the latter being achieved without downloading the entire video. Looking forward at

the desired output of this application, either a video or a stream representation, that is, if the input is a video or a stream URL respectively, wants to be obtained.

Deep learning applications can lead to a lot of architectural decisions cross-roads because of their high complexity. Multiple implementations are necessary for making the best decision when it comes to the final workflow of this application.

When talking about video or stream inputs, a system which can run a piece of code periodically on the current frame is wanted.

To summarize, the specific objectives of this thesis are the following:

- Develop a software application that receives at input a prerecorded video in CDep and generates an output video with name banners applied over the parts of the video where the speaker is identified
- Save the output video
- Implement and train an `undefined` class for the classifier
- Offer a way to debug the script execution which created the output video
- Create a dataset with plenum and non-plenum images
- Create a way to classify plenum from non-plenum images
- Extend application described above to work with video streams

In Chapter 2, the theory upon which the application is built will be summarized, namely the machine learning and deep learning concepts, followed by the development tools that were used for the actual programming of the application.

Chapter 3 will start denoting some discovered issues which were revealed while implementing the existing network. In the next section of this chapter, additions which solved some of these problems will be described.

In Chapter 4, scripts which were developed independently of one another will be presented, and then the integrations between them in the final application will be described.

Finally, Chapter 5 enumerates conclusions for the project and states next steps for the bigger project.

Chapter 2

Theoretical concepts

2.1 Machine learning. Deep learning

Machine learning is the domains which wants to solve optimization problems in order to fit some piece of data into a pattern, after a system was trained to do so.

Machine learning is a field within computer science, but it brings different solutions than that of the traditional domain. In traditional computing, algorithms are sequential programmed commands used by computers to calculate or solve a problem. Machine learning instead set computers to train on data inputs and use statistical analysis to output values that fall within a specific cluster or a specific range. Because of this, machine learning facilitates computers in building models from sample data in order to automate decision-making processes based on data inputs. [5]

Any technology user today has benefited from machine learning: from facial recognition technology that allow social media platforms to help users tag and share photos of friends, to OCR. In marketing or advertisements, machine learning helps with recommendation engines targeted ads, movies or television shows suggestions and so on.

Machine learning is a continuously developing field. Because of this, there are some considerations to keep in mind as you work with machine learning methodologies, or analyze the impact of machine learning processes. It is also the capability of AI systems to have the ability to acquire their own knowledge, by extracting patterns from raw data.

The majority of deep learning applications train their data in a supervised fashion, so this type of learning is called supervised learning. This means that the developer needs to label the training data in order for the network to extract features from those inputs and correlate them such that it will achieve a good accuracy on different variations of that same label. The process of labeling inputs is, the most of the time, the most labor-intensive step, but also an important one because the success of the neural network is highly dependent on good training data. Even if the dataset isn't large enough, a process called augmentation can be applied on labeled input data in scenarios when the dataset is not large enough.

Deep learning comes as an improvement for machine learning, which wants to solve the problem of abundance from the previous step and form complex functions by training many neurons in a network. It is based on a simulation of the brain model, and as stated by Andrew Ng, deep learning scales in performance as more data is fed to a model. [6]

A neural network is a structure formed of units called neurons, which can take an input, multiply that value with a weight, and add a bias to the result, thus obtaining the output of the neuron. The size of neural networks can vary, but large ones (100,000+ neurons) can form

very complex functions.

For example, a simple network formed of 2 inputs, A and B, 1 fully connected hidden layer of 1 neuron, and an output layer can be trained to learn the AND or the OR logical function. That is because both functions are described by a linear function which separates the two possible outcomes of that gate. The above examples are part of a subclass classification problems called binary classification. When the output of a problem has a finite number of classes or labels, it's called a classification problem. Binary comes from the fact that there are only two possible outputs, 0 and 1.

Deep learning solves the problem of extracting high-level, abstract features from raw data. Many factors of variation can be identified only using sophisticated, nearly human-level understanding of the data, but they can be introduced by expressing them in terms of other, simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts. [7]

The founding example of a deep learning model is the feedforward deep network, or MLP. This is just a mathematical function which associates some set of input values to output values.

The idea of learning the right representation for the data provides one perspective on deep learning. Another perspective on deep learning is that depth enables the computer to learn a multistep computer program. Each layer of the representation can be thought of as the state of the computer's memory after executing another set of instructions in parallel.

2.1.1 Categories of problems in machine learning

Classification is the task of predicting a discrete class label. Regression is the task of predicting a continuous quantity. There is some overlap between the algorithms for classification and regression. A classification algorithm may predict a continuous value, but the continuous value is in the form of a probability for a class label. A regression algorithm may predict a discrete value, but the discrete value in the form of an integer quantity.

Some algorithms can be used for both classification and regression with small modifications, such as decision trees and artificial neural networks. Some algorithms cannot, or cannot easily, be used for both problem types, such as linear regression for regression predictive modeling and logistic regression for classification predictive modeling.

Other famous classification problems are based on datasets, like the MNIST's handwritten digits database, the Iris flower dataset, the survival of the Titanic passengers from Kaggle etc.

Regression, on the other hand, aims to predict a trend based on the training set and, hopefully, predict also future inputs that the network never encountered before.

2.1.2 Image classification

Many tasks in computer vision, such as object detection or background segmentation, can be reduced to image classification. The task of recognizing a visual concept is relatively easy for humans, but a lot of problems are raised when trying to tackle this using computer vision algorithms.

Some of these problems are:

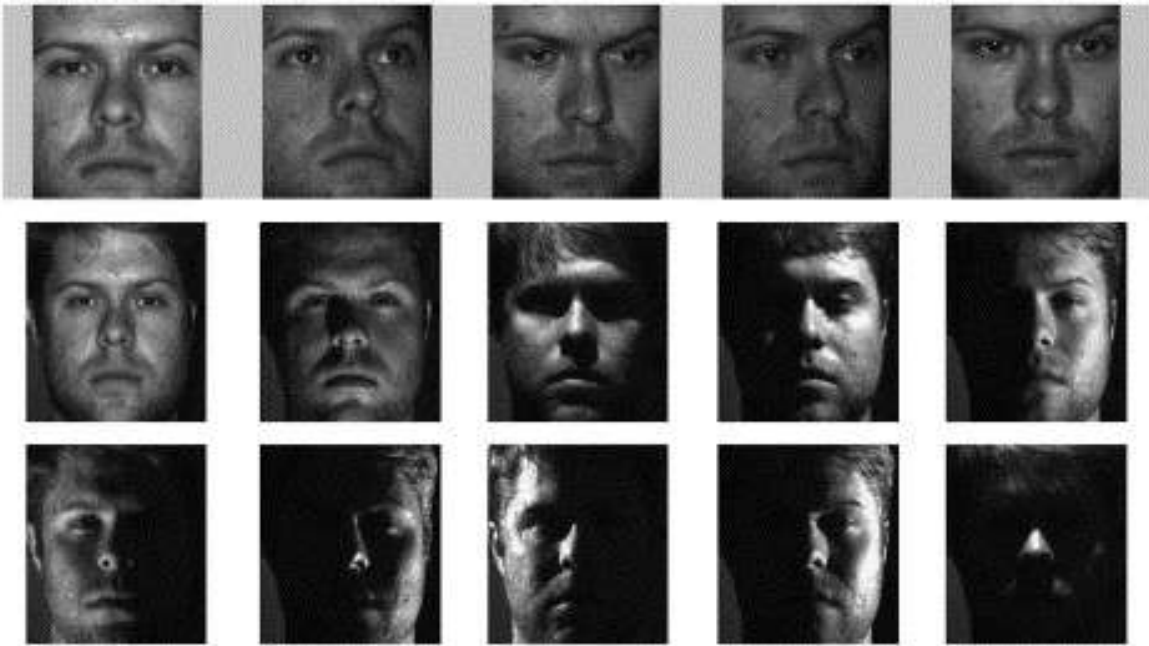


Figure 2.1: Perspective and lighting variations on the same subject

Source: [8]

- Variation of orientation: objects can be oriented differently with respect to the camera.
- Background noise: objects can “blend” with the environment
- Bad lighting: classification can be difficult when an object’s features are hidden by either low or high luminosity, as seen in figure 2.1
- Object warping: some photos can capture objects that are not in their expected position or structure
- Not so different classes: systems may have to tell the difference between very related classes, for example different breeds of cats

The success of a classification system is dependent on the invariance of the model to the above problems.

The simple method to ensure this invariance is called the data-driven approach: the user must provide the computer many examples of each class and then develop methods which will look at these examples and learn about the visual features of each class. This consists of accumulating a large enough training dataset of labeled images.

To classify an image, it first needs to be transformed into a vector. An image is actually a 3 dimensional matrix, formed of width, height and depth or channels. The number of colored pixels in an image is given by the product between the width and the height. To mutate this matrix into an array, an operation called flattening is applied, where the pixel’s values are first concatenated together, from the top left pixel to the top right pixel. When the row finished, the concatenation proceeds from the first pixel of the next row, and so on, until every pixel is flattened.

Image classification has in total three steps that need to be done:

Average Precision (AP):	
AP	% AP at IoU=.50:.05:.95 (primary challenge metric)
AP ^{IoU=.50}	% AP at IoU=.50 (PASCAL VOC metric)
AP ^{IoU=.75}	% AP at IoU=.75 (strict metric)
AP Across Scales:	
AP ^{small}	% AP for small objects: area < 32 ²
AP ^{medium}	% AP for medium objects: 32 ² < area < 96 ²
AP ^{large}	% AP for large objects: area > 96 ²
Average Recall (AR):	
AR ^{max=1}	% AR given 1 detection per image
AR ^{max=10}	% AR given 10 detections per image
AR ^{max=100}	% AR given 100 detections per image
AR Across Scales:	
AR ^{small}	% AR for small objects: area < 32 ²
AR ^{medium}	% AR for medium objects: 32 ² < area < 96 ²
AR ^{large}	% AR for large objects: area > 96 ²

Figure 2.2: COCO competition metrics
Source: [9]

- Ensure a set of N input images are labeled with one of M different classes, where N is the total number of input images, and M is the total number of classes
- Training the classifier, also known as learning a model
- Evaluation. This step evaluates the quality of the classifier by providing a new set of images which it never encountered before, see what the predicted results are, and then compare them with the true answers, also known as the ground truth. This steps will show if the model overfitted on the training set.

For a video application, one more thing to emphasize about image classification would be the inference time of the classifier. In order to achieve real-time speed for the application, the inference time would need to be smaller than the time period between frames. So if the goal would be a real-time application for a 30 FPS video, the inference time should be smaller than:

$$\frac{1}{30 \text{ frames per second}} = 33,33 \text{ milliseconds per frame} \quad (2.1)$$

Latest research papers tend to give results for the COCO dataset only, which is a large-scale object detection, segmentation, and captioning dataset [9]. In COCO mAP, a 101-point interpolated AP definition is used for the benchmarks. For COCO, AP is the average over multiple IoU (the minimum IoU [Intersection over Union] to consider a positive match) [10]. There are multiple benchmarks, but the most important ones are the mAP at different scales, S, M and L, and mAP for a 50% and 75% IoU. All the metrics can be seen in figure 2.2.

As it can be seen in figure 2.3, YOLOv3 is the only classifier which can achieve real-time speeds for a video application, but with the drawback of a lower AP.

2.1.3 Nearest Neighbor

A first approach for image classification in machine learning would be the Nearest Neighbor Classifier. It is a good candidate to analyze before discussing the convolutional neural networks, which will underline the thinking processes that happen under the hood but also the

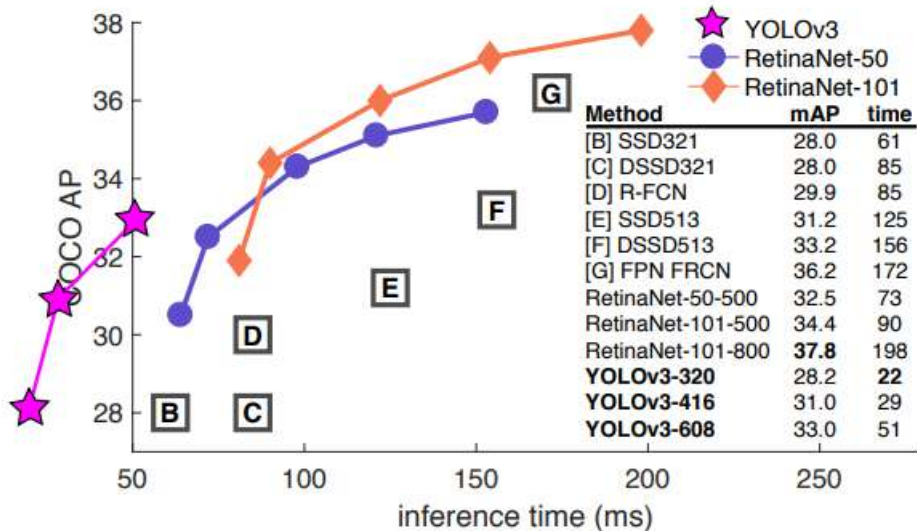


Figure 2.3: COCO AP versus inference time for state-of-the-art object detection systems
Source: [11]

shortcomings of a naive approach.

A nearest neighbor classifier takes an image and compare it to every other image in the training set, and the resulting label will be that of the closest image from the training set. At a first glance, this process is not very efficient, since it loops through the entire training set for each prediction. This type of classifier is known as an in-memory classifier, meaning that the entire training set is stored in the memory when the classifier initializes.

To quantify the distance between two images, there are multiple formulas which can be used. One of the simplest way would be to compare each image pixel by pixel, compute the difference and sum up all the differences. This is known as the L1 distance, and the exact formula is:

$$L_1(I_1, I_2) = \sum |I_1 - I_2| \quad (2.2)$$

Another distance, L2, introduces a square term, which will make it more sensible for big disagreements in pixels, rather than smaller but more mismatches. The formula for it is the following:

$$L_2(I_1, I_2) = \sqrt{\sum (I_1 - I_2)^2} \quad (2.3)$$

A better iteration for nearest neighbors algorithm is the k -Nearest Neighbors classifier, which introduces voting for the label from the k “closest” images to the test image. This algorithm will select the k images with the smallest distance from the test one, will analyze their labels, will detect the predominant label and associate it to the test image.

2.1.4 Linear classification and SVM

As described in the previous subsection, kNN has a number of disadvantages. To fix them, the concept of a score function and a loss function will be introduced.

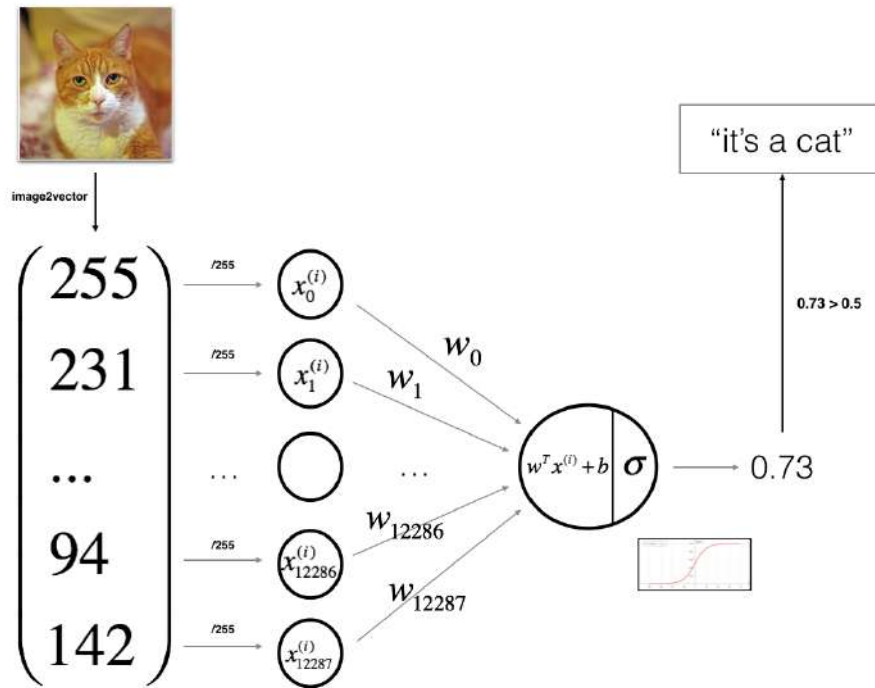


Figure 2.4: Example of image classification with only one neuron
Source: [12]

First, a linear classifier is a classifier that uses a linear function to compute the predicted label of an image, as seen in 2.4. A good function for this case would be:

$$f(x_i, W, b) = Wx_i + b \quad (2.4)$$

x_i is a column vector and its dimensions are computed after flattening the image matrix, so $[D \times 1]$, where D is the product $width \times height \times nr_channels$. A justification for organizing all the weights in a matrix format is because matrix multiplication has become an efficient process for both processors and graphics card. The dimensions of the matrix W are $[K \times D]$, where K is the number of classes. When doing the matrix multiplications, a term with dimensions $[K \times 1]$ is obtained, which is also the size of the bias b .

By doing the dot product Wx_i , K essentially evaluates separate classifiers with one input with a very fast process. As stated above, each row of W is actually a classifier by itself. What is obtained after the training process, it's called a template for each of the classes. Each pixel in the template corresponds to the pixels which will generate the highest score or activation for that pixel. Normally, W holds values both positive and negative, which couldn't normally be represented in an image. [12]

2.1.5 Haar cascades

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper in 2001 [2]. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images, as described in [13].

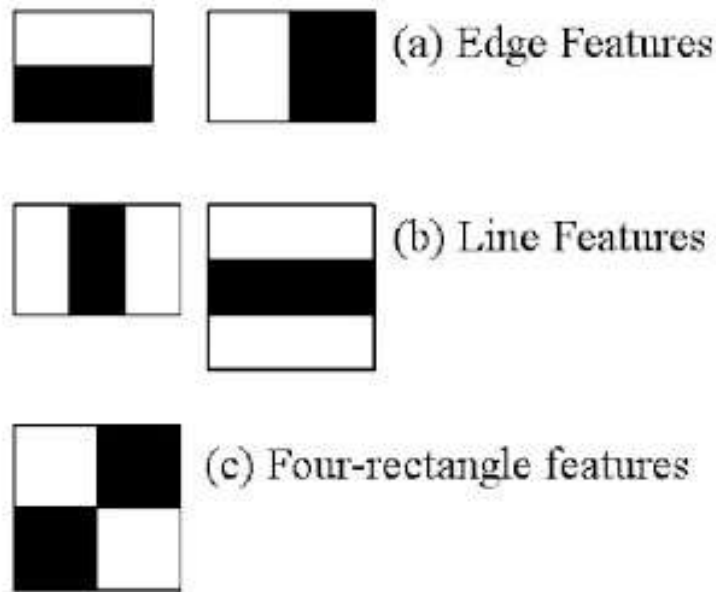


Figure 2.5: Haar convolutional kernels
Source: [13]

The algorithm has 4 steps:

- Calculate Haar features
- Create integral images
- Apply Adaboost
- Implement Cascade Classifiers

It is mainly used in face detection. Initially, the algorithm needs a lot of positive images (in the current context, images containing faces) and negative images (images that don't contain faces) to train the classifier. Next, features will be extracted from them. For this, Haar features will be computed using the kernels represented in figure 2.5. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle. [13]

The kernels are moved all across the image in each possible location. There are $W - S + 1$ possible moves on the horizontal axis, where W is the image's width and S is the kernel's size.

Given that the base resolution of the detector is 24x24, the exhaustive set of rectangle features is quite large, over 180,000.

For each feature calculation, the sum of the pixels under white and black rectangles will be found. To solve this, they introduced the integral image. However, large your image, it reduces the calculations for a given pixel to an operation involving 4 array references, as seen in figure 2.6.

For this, each feature is applied on all the training images. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications.

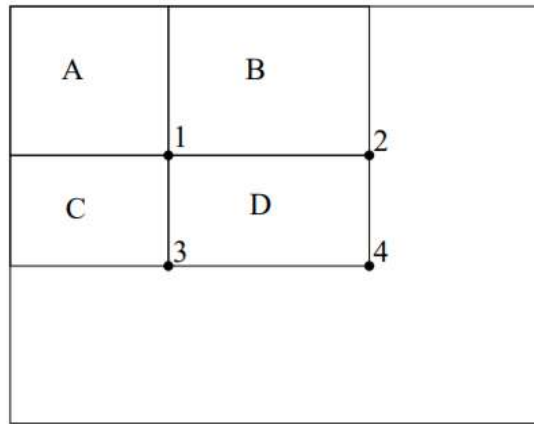


Figure 2: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A . The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

Figure 2.6: Graphical explanation of the integral images
Source: [2]

The features with minimum error rate are selected, which means they are the features that most accurately classify the face and non-face images.

The final classifier is a weighted sum of these weak classifiers. This is done using Adaboost [14]. They are called weak because it alone can't classify the image, but multiple weak classifiers put together form a strong one. Even for as little as 200 features, the detection accuracy is 95%.

2.1.6 Neural networks

The neural networks' domain is primarily driven by the goal of modeling the basic computational unit of the brain, the neuron. Neurons receive inputs from dendrites, and produce a single output through an axon. The weight of a neuron, from the mathematical model, is equivalent with the synaptic strength in the biological model. The bias b doesn't have any association in the biological model, but the idea is that when the total sum goes over a threshold, the neuron can fire. The activation function is applied to the result, and the final value is known as the firing rate of the neuron. A visual example can be found in figure 2.7. [12]

The mathematical form of the model Neuron's forward computation might look familiar to you. As with linear classifiers, a neuron has the capacity to "like" (activation near one) or "dislike" (activation near zero) certain linear regions of its input space.

Deep feed-forward networks are what define the deep learning models. The goal of a feed-forward network is to approximate some function g which maps an input x to a category y .

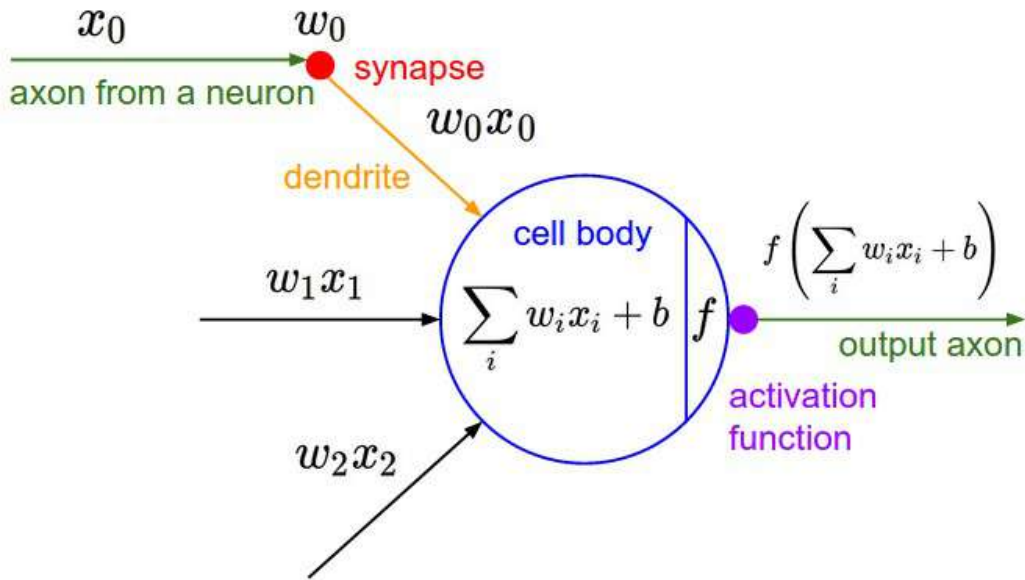


Figure 2.7: Mathematical model of a neuron
Source: [12]

The network defines a mapping $z = f(x; \theta)$, and learns the value of the parameters θ in order to find the best function approximation.

They are called feedforward networks because information flows through the function being evaluated from x , through the intermediate computations used to define f , and finally to the output z . When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks, which are outside the scope of this project.

Feedforward networks are of great importance because they stand at the basis of most of the machine learning applications. Even convolutional neural networks, which will be discussed in a future subsection, are a form of specialized MLP.

Finally, these networks are known as neural networks because they are based on neuroscience. The network's hidden layers are usually vector valued. The width of the model is determined by the dimensionality of these hidden layers. Each element of the vector can be thought of as having a neuron-like function. Rather of representing a single vector-to-vector function, the layer can instead be thought of as a collection of units acting in parallel, each representing a vector-to-scalar function. Each unit functions similarly to a neuron in that it receives input from other units and calculates its own activation value.

The concept of layering vector-valued representations comes from neuroscience. However, modern neural network research is influenced by a variety of mathematics and engineering fields, and the purpose of neural networks isn't to completely imitate the brain. Rather than being models of brain functions, feedforward networks should be thought of as function approximation machines that are designed to accomplish statistical generalization while occasionally taking some insights from what is known about the brain.

They are typically represented as a directed acyclic graph, because they form a chain of functions like so: $f(x) = g(h(i(x)))$. The chain structures are the most commonly used structures of neural networks. In this case, g would be the first layer of the network, h the second and so on. The training data provides the network with noisy samples from different points

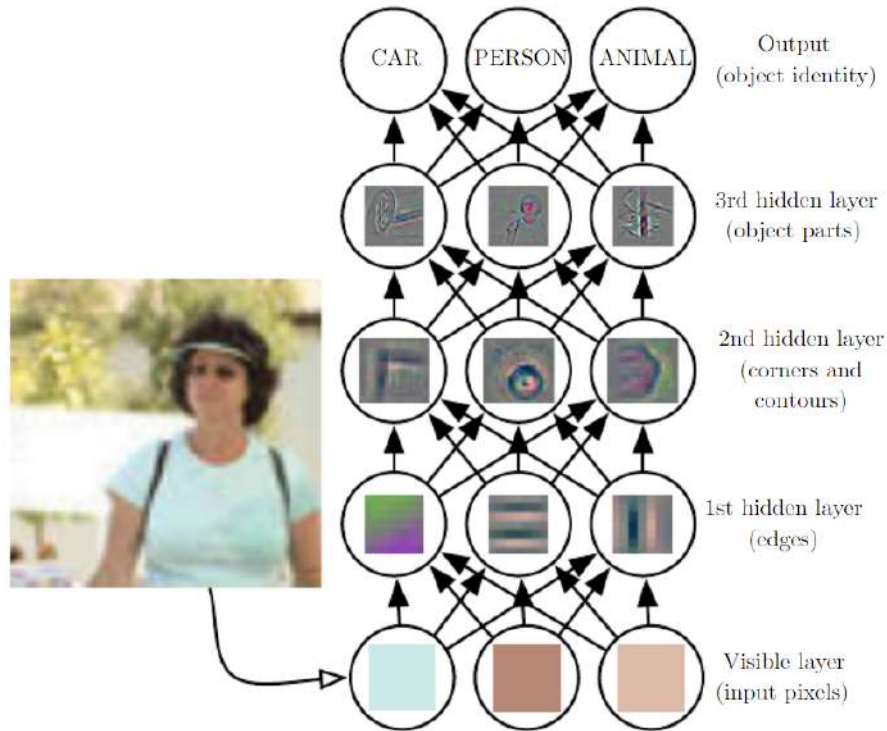


Figure 2.8: Illustration of a deep learning model

Source: [7]

of $g(x)$. During training, the network drives $f(x)$ closed to $g(x)$. The training data doesn't say what those layers should do, so the learning algorithm must decide how to use the deeper layers to produce the desired output. That's why they are called hidden layers. It can be seen in figure 2.8 how the learning process influences the activations for each layer. [7]

2.1.7 Gradient Descent

The gradient descent algorithm is the main solution when it comes to adjusting the weights of a layer in order to minimize the cost function. The formula for it is:

$$W \leftarrow W - \alpha \cdot \frac{\partial J}{\partial W} \quad (2.5)$$

Gradient descent has a reputation for being slow and unreliable. Gradient descent was once thought to be reckless or unethical when applied to non-convex optimization problems. Machine learning models that are trained via gradient descent perform exceptionally well. Although the optimization process cannot ensure that it will arrive at a local minimum in an acceptable length of time, it frequently reaches a very low cost function value quickly enough to be effective.

2.1.8 Regularization methods

There are several ways of controlling the capacity of Neural Networks to prevent overfitting.

The most prevalent type of regularization is L2 regularization. It can be achieved by penalizing all parameters' squared magnitudes directly in the objective. That is, the phrase $\frac{1}{2}w^2$ is

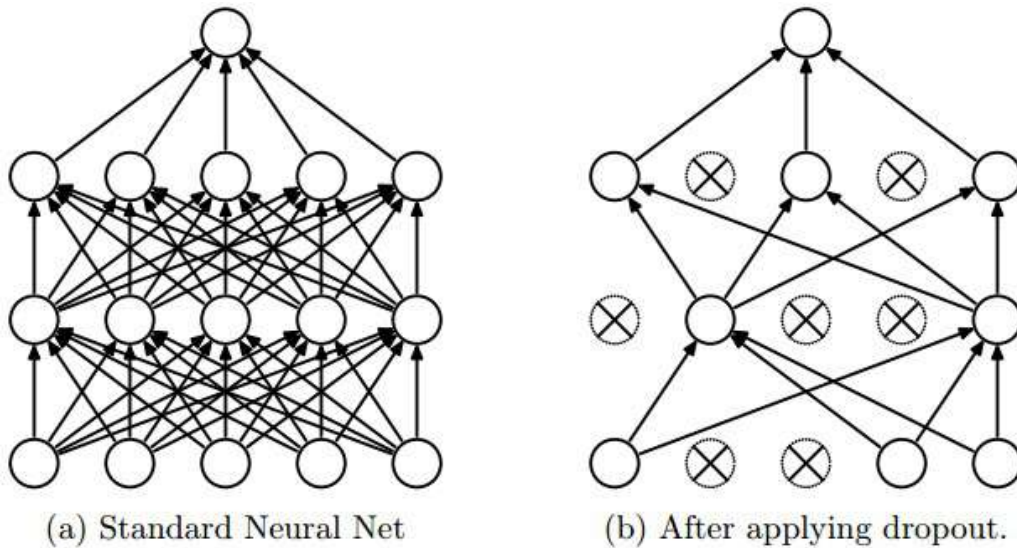


Figure 2.9: Example of dropout
Source: [12]

added to the objective for each weight w in the network, where λ is the regularization strength. Because the gradient of this term with respect to the parameter w is just w instead of $2\lambda w$, it is typical to see a factor of $\frac{1}{2}$ in front. The natural interpretation of the L2 regularization is that it significantly penalizes peaky weight vectors while favoring diffuse weight vectors. Finally, utilizing the L2 regularization during gradient descent parameter update means that each weight is degraded linearly: $W += -\lambda W$ towards zero.

The term $\lambda|w|$ is added to the target for each weight w in L1 regularization. Combining the L1 and L2 regularizations is possible: $\lambda_1|w| + \lambda_2 w^2$ (this is called Elastic net regularization). During optimization, the L1 regularization has the interesting virtue of making the weight vectors sparse (i.e. very close to exactly zero). To put it another way, neurons with L1 regularization use only a sparse selection of their most critical inputs and become nearly invariant to “noisy” inputs. In practice, if explicit feature selection is not a priority, L2 regularization is likely to outperform L1 regularization.

Dropout is implemented during training by keeping a neuron active with a probability p (a hyperparameter) or changing it to zero if it is not, as seen in figure 2.9.

2.1.9 Convolutional Neural Networks

Convolutional Neural Networks are very similar to ordinary Neural Networks because they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels from the input to class scores at the output. And they still have a loss function on the last FC (Fully Connected) layer, and all the theory applied for learning regular Neural Networks still apply.

CNN (Convolutional Neural Network) architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture, like contextual information. These then make the forward function more efficient to implement and

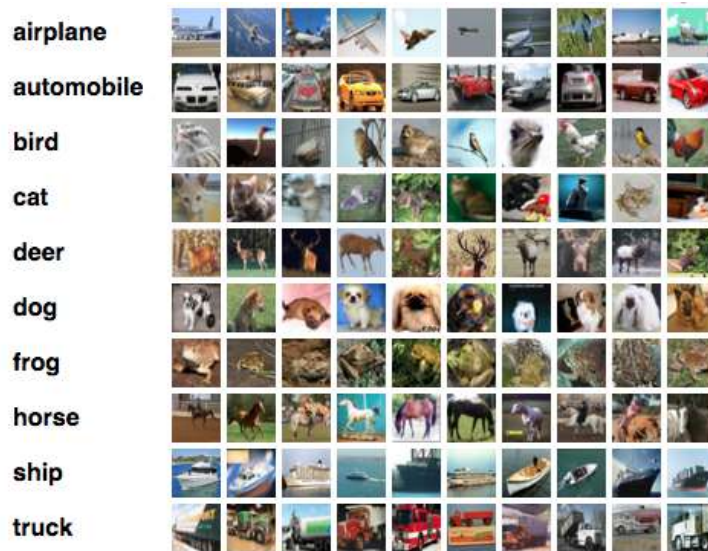


Figure 2.10: Example of samples from the CIFAR-10 dataset
Source: [12]

vastly reduce the amount of parameters in the network.

Neural Networks take an input (a single vector) and change it through a sequence of hidden layers, as seen in the previous chapter. Each hidden layer is made up of a group of neurons, each of which is fully connected to all neurons in the previous layer and which act completely independently of one another, as seen in figure 2.8. The final fully linked layer is known as the "output layer," and it provides the class scores in classification settings.

Regular Neural Networks do not scale well to full-size images. Because images in the CIFAR-10 dataset are only $32 \times 32 \times 3$, a single fully-connected neuron in a conventional Neural Network's first hidden layer would have $32 \times 32 \times 3 = 3072$ weights. Some examples of the dataset can be seen in figure 2.10. This quantity appears to be manageable, but the FC structure definitely does not scale to larger photos. An image of a more reasonable size, such as $200 \times 200 \times 3$, would result in neurons with weights of $200 \times 200 \times 3 = 120,000$. Furthermore, numerous of these neurons are wanted, so the parameters would quickly mount up. This complete connection is obviously inefficient, and the large number of parameters would quickly lead to overfitting. [12]

CNNs make use of the fact that the input is made up of images to confine the architecture in a more reasonable way. Unlike a conventional Neural Network, a CNN's layers have neurons arranged in three dimensions: width, height, and depth. (Note that the term "depth" refers to the third dimension of an activation volume, not the total number of layers in a network.) The input images in CIFAR-10, for example, are an input volume of activations with dimensions of $32 \times 32 \times 3$ (width, height, depth respectively). The neurons in a layer will only be connected to a tiny part of the layer before it, rather than all the neurons in a fully connected fashion. Furthermore, the final output layer for CIFAR-10 would be $1 \times 1 \times 10$ in size, as the complete image will be converted to a single vector of class scores grouped along the depth dimension by the end of the CNN architecture [12]. A visualization can be found in figure 2.11.

A simple CNN is a series of layers, each of which translates one volume of activations to another using a differentiable function, as explained above. Convolutional Layer, Pooling

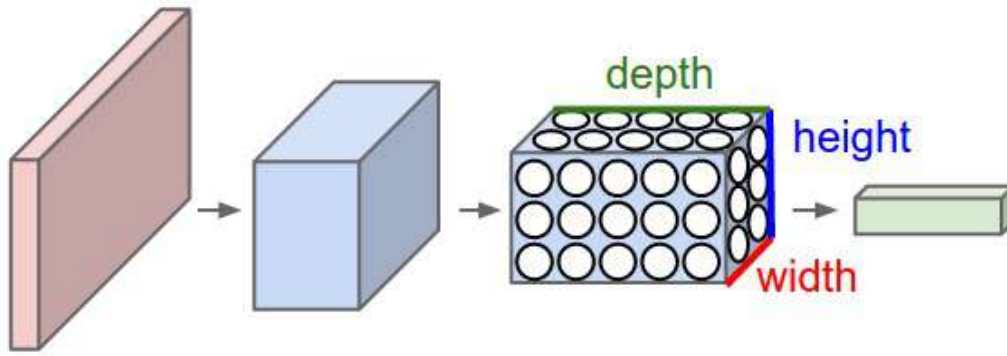


Figure 2.11: Visual representation of a CNN
Source: [12]

Layer, and Fully-Connected Layers are the three major types of layers used to construct CNN architectures. These layers will be stacked to construct a complete CNN architecture.

The convolutional layer's parameters are made up of a sequence of learnable filters. Even though each filter is modest (in terms of width and height), it covers the entire depth of the input volume. A $5 \times 5 \times 3$ filter on a CNN's first layer, for example, is a common filter (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). Throughout the forward pass, each filter slides (or, more accurately, convolves) over the width and height of the input volume, computing dot products between the filter's entries and the input at any point during the forward pass. The filter will produce a 2-dimensional activation map with the filter's responses at every spatial position as it slides across the input volume's width and height. On the first layer, the network will learn filters that activate when it sees a visual feature like an edge of some orientation or a splotch of some color, and on higher layers, whole honeycomb or wheel-like patterns. Each convolutional layer will now contain a full set of filters (for example, 12 filters), each of which will produce a unique 2-dimensional activation map. By stacking these activation maps along the depth dimension, the output volume is formed.

Every entry in the 3D output volume can be understood as the output of a neuron that only looks at a tiny portion of the input and shares parameters with all neurons to the left and right spatially (since these numbers all result from applying the same filter).

A good thing to discuss are the details of the neuron connectivities, their arrangement in space, and their parameter sharing scheme. It is impossible to connect neurons to all neurons in the preceding volume when dealing with high-dimensional inputs such as pictures, as seen previously. Instead, each neuron will be connected to only a small portion of the input volume. The receptive field of the neuron is a hyperparameter that describes the geographic extent of this connectivity (equivalently, this is the filter size). The depth of the input volume is always equal to the extent of connectivity along the depth axis. This disparity in how the spatial dimensions are treated (width and height) and the depth dimension must be emphasized once more. The connections are local in 2D space (along width and height), but always full along the entire depth of the input volume.

In a CNN architecture, a pooling layer is frequently inserted between subsequent convolutional layers. Its purpose is to gradually shrink the representation's spatial dimension in order to reduce the number of parameters and computations in the network, and thus to prevent

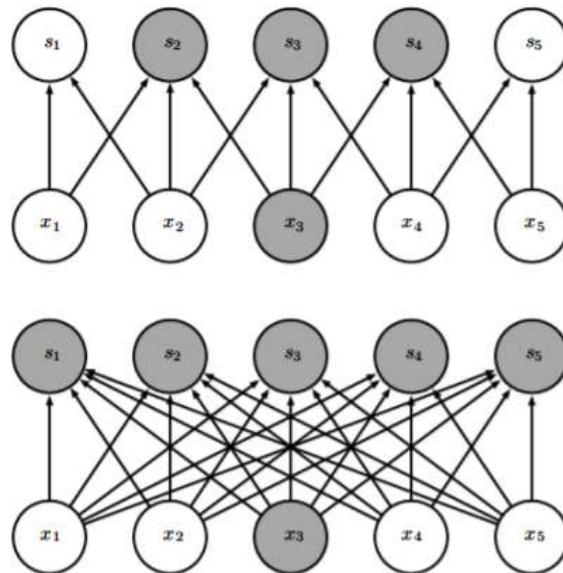


Figure 9.2: Sparse connectivity, viewed from below. We highlight one input unit, x_3 , and highlight the output units in \mathbf{s} that are affected by this unit. (Top)When \mathbf{s} is formed by convolution with a kernel of width 3, only three outputs are affected by \mathbf{x} . (Bottom)When \mathbf{s} is formed by matrix multiplication, connectivity is no longer sparse, so all the outputs are affected by x_3 .

Figure 2.12: Graphical example of sparse connectivity

Source: [7, Figure 9.2]

overfitting. Using the MAX operation, the pooling Layer resizes each depth slice of the input spatially.

A pooling layer with 2×2 filters applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, removing 75% of the activations, and is the most frequent variant. In this situation, every MAX operation would take a maximum of four digits (a 2×2 region in a depth slice). The depth dimension stays the same.

As with conventional Neural Networks, neurons in a fully connected layer have full connections to all activations in the previous layer. As a result, their activations can be calculated using matrix multiplication and a bias offset.

Sparse interactions, parameter sharing, and equivariant representations are three fundamental notions that might help improve a machine learning system. Furthermore, convolution allows you to work with inputs of varying sizes. [7]

Matrix multiplication by a matrix of parameters with a separate parameter indicating the interaction between each input unit and each output unit is used in traditional neural network layers. This means that each output unit has a relationship with each input unit. Convolutional networks, on the other hand, are known for their sparse interactions (also referred to as sparse connectivity or sparse weights). An example is showcased in figure 2.12.

Making the kernel smaller than the input does this. When processing a picture, for example, the input image may comprise thousands or millions of pixels, but small, important features can be detected like edges using kernels that are only tens or hundreds of pixels in size. This way, fewer parameters may be kept, which decreases the model's memory requirements while



Figure 2.13: Example of NMS
Source: [15]

also increasing its statistical efficiency. It also means that computing the output requires fewer operations. These improvements in efficiency are usually quite large. If there are m inputs and n outputs, then matrix multiplication requires $m \times n$ parameters, and the algorithms used in practice have $O(m \times n)$ runtime (per example). If the number of connections each output may have is limited to k , then the sparsely connected approach requires only $k \times n$ parameters and $O(k \times n)$ runtime. For many practical applications, it is possible to obtain good performance on the machine learning task while keeping k several orders of magnitude smaller than m . [7]

2.1.10 Non-maximum suppression

For the upcoming chapters, the term called NMS (Non-Maximum Suppression) must be described. In object detection, multiple predictions for a single object can occur, which can cause disruptions for systems which depend on the output of the detector, but will also add multiple bounding boxes for that object, as seen in figure 2.13.

Considering that bounding boxes with confidence score below a respective threshold were removed and there are still bounding boxes that overlap, the algorithm can be applied.

Next, the IoU is computed between two of the overlapping boxes. Another threshold for IoU must be set, and if the value of the measurement is higher than this threshold, the box with the smaller confidence score will be discarded. If the boxes have the same confidence score, either of them can be discarded.

2.1.11 R-CNN and iterations

In object detection, the desired output size cannot be known, since in an image there can be a variable number of detected objects. A first solution for this problem would be to first detect different regions in the image, and then feed them into a CNN in order to extract features from each of the regions. As one would imagine, this can get computationally expensive really fast.

R-CNN: *Regions with CNN features*

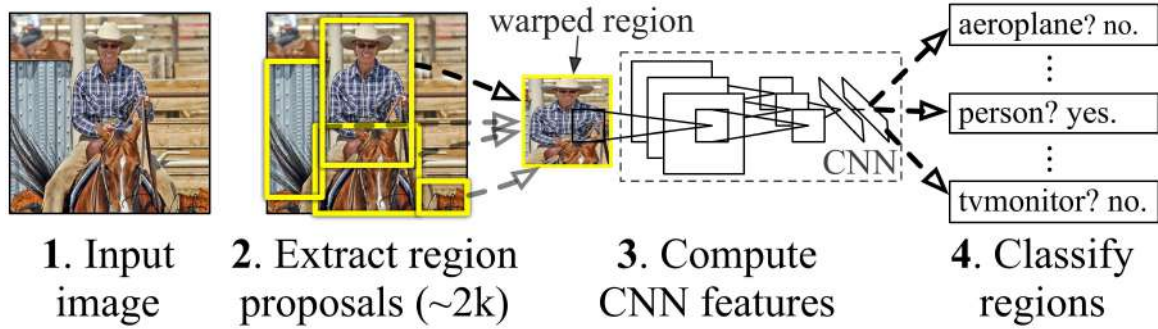


Figure 2.14: R-CNN algorithm example
Source: [17]



Figure 2.15: Segmentation example
Source: [19]

Finally, the features are fed to an SVM, which classifies the image. [16]

So, algorithms like R-CNN and YOLO appeared in the computer vision scene. In 2014, the R-CNN paper [17] was published, which proposed the use of 2000 of what the paper calls “region proposals”. This is done using a selective search algorithm [18], which is a type of segmentation algorithm. The algorithm pipeline can be seen in figure 2.14.

Segmentation is a process which groups pixels based on their spatial context or belonging, as seen in figure 2.15. Context can sometimes be easily determined from pixels’ value, but not always. For example, a bed is formed out of a mattress and a bed frame, which will most likely differ in colors, but nonetheless, they should be put in the same segmentation or context. The paper [18] designs a formula to compute similarity based on color, texture, the size of the regions and the way the regions fit with each other.

At the next iteration of R-CNN, called Fast R-CNN, improves on the previous paper by feeding the input image and the proposed RoI (Region of Interest) into a FCNN (Fully Connected

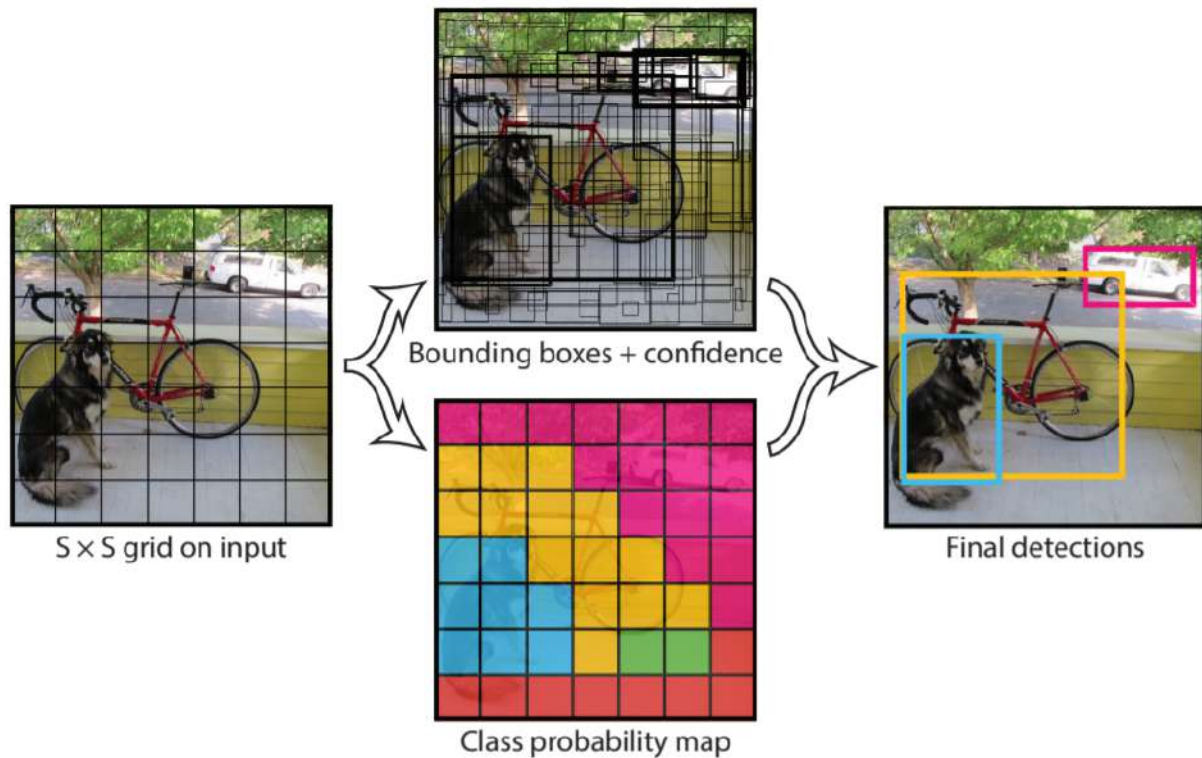


Figure 2.16: Example of how YOLO does a prediction

Source: [22]

Neural Network). Then, for each object proposal, a RoI pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of FC layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a catch-all “background” class and another layer that outputs four real-valued numbers for each of the K object classes. [20]

At the 2nd iteration of R-CNN, called Faster R-CNN [21], actually selective search was opted out in favor of an in-house RPN (Region Proposal Network) [21, Section 3.1].

2.1.12 YOLO neural network

The YOLO (You Only Look Once) [22] neural network offers a new approach to object detection.

Previous articles on object detection reuse classifiers in order to perform detection. Instead, object detection is seen here as a regression problem in which the task is to detect separated bounding boxes and their associated class probability. In one evaluation (from here the name), the bounding boxes and class probabilities are directly predicted from images, as it can be seen in figure 2.16. The detection pipeline is just one network and can be optimized accordingly for the desired detection needs. [22]

YOLO is popular because it achieves high accuracy while also being able to run in real-time. It does that by first dividing the input image into S by S grid. Each of this grid cells will make

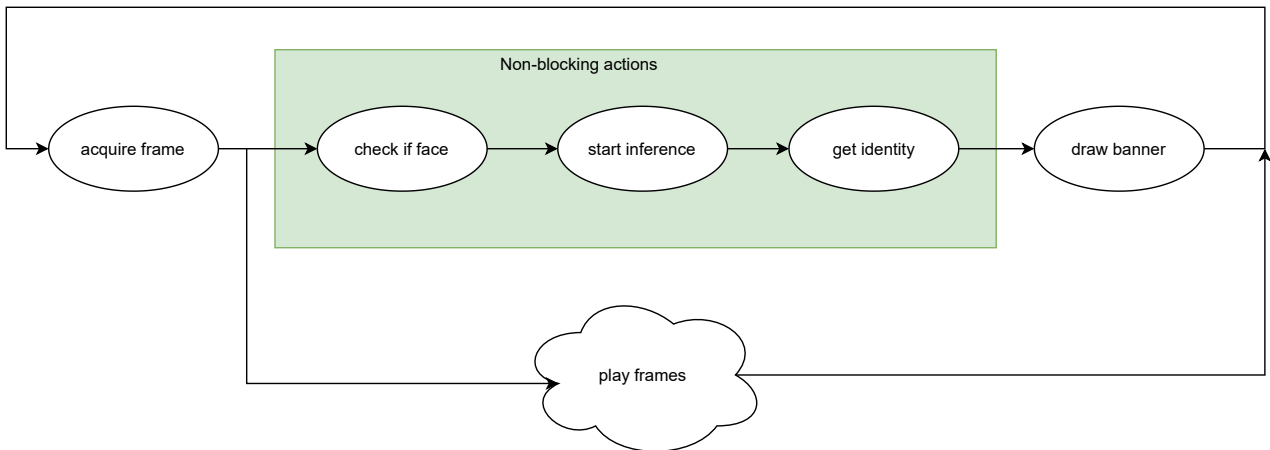


Figure 2.18: Flowchart of the desired frame handler

As stated in the official docs [23],

“The `concurrent.futures` module provides a high-level interface for asynchronously executing callables.”

Futures are used for managing results computed by the workers. To use a pool of workers, an application creates an instance of the appropriate executor class and then submits them for it to run. When each task is started, a `Future` instance is returned. When the result of the task is needed, an application can use the `Future` object to block until the result is available. Various APIs are provided to make it convenient to wait for tasks to complete, so that the `Future` objects do not need to be managed directly. [24]

The `concurrent.futures` module plays a big role in the frame handling function. This library offers structures called `Futures`, which has a pending state and a finished state. With it, parallel jobs can be run by spawning a new thread or process, depending on the user choice. To evaluate this `Future` object, the `done` class method needs to be called in order to see if the result of that specific finished job.

The module works in the following manner: it depends on an executor pool, which, as stated as above, can be a process or a thread one. The executor is configured with the maximum numbers of workers. From there, the developer can `submit` jobs to this executor, command which will return the `Future` object. This `Future` structure has two main methods: `done` and `result`. While the first polls for the result and checks if the job is ready, the second returns the returned value of the job.

This module comes as a complement to the `async/await` feature in PEP 492. While waiting for an event is useful and the offered syntax sugar comes in handy when integrating asynchronous code inside synchronous code, this feature does not provide support in a video application.

Also, another very powerful module in Python is `Pathlib`, which is used a lot in the project. This module offers many path functionalities and saves a lot of time by proposing a graph structure for paths, making resolving a final path very easy.

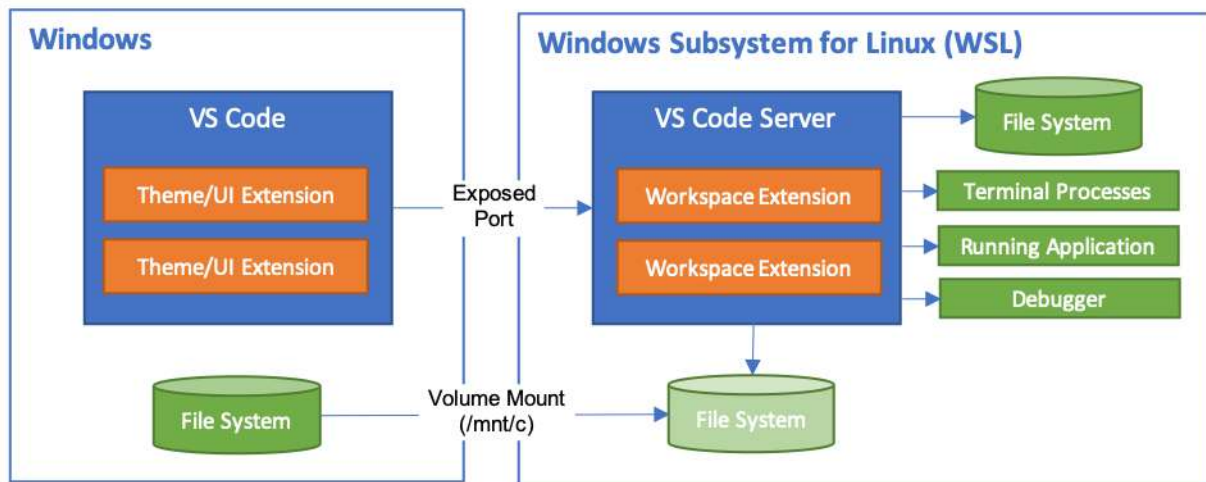


Figure 2.19: WSL architecture
Source: [27]

2.2.2 Tensorflow

TensorFlow is an end-to-end open source platform for machine learning, developed by Google, which is available for multiple programming language, including Python. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that let researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

TensorFlow offers multiple levels of abstraction, so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy. [?]

2.2.3 HLS protocol

The HLS protocol was designed by Apple, and it was released in 2009, and its specification is detailed in RFC 8216 [25]. The CDep's website², <http://www.cdep.ro/>, uses the Wowza Streaming Engine, as shown by its source code.

2.2.4 WSL

WSL is a compatibility layer that runs on top of Windows and allows you to have the best of both worlds. Due to its unrivaled support for development tools and similarity to production environments, Unix-based systems such as Linux and macOS have traditionally been the go-to Operating Systems for Web Developers and Open Source project contributors. Although Microsoft Windows has greatly improved its developer experience over the years, particularly with the addition of tools such as PowerShell and Hypervisor technologies, many developers still prefer Linux to Microsoft Windows as their development platform. [26]

2.2.5 X11 server and xcb

As the development of this application started on top of another developed in a Unix environment, the WSL compatibility utility was used, as the entire work was conducted on a Windows machine. X server came as a solution for a big problem: since the development was carried on a virtual machine, and the application is dependent on a graphic manner for displaying frames, there was no native solution for such a data communications between the systems.

Using OpenCV as the main interface for serving the video frames, functions like `imshow` spawn a named window, which uses the machine's operating system's GUI functions.

To enable frame forwarding to the X server, a command must be added in the `.bashrc` of the environment created with WSL. That command is `export DISPLAY=:0`.

The environment variable `DISPLAY` tells GUI programs how to communicate with the GUI. A Unix system can run multiple X servers, i.e. multiple display. These displays can be physical displays (one or more monitor), or remote displays (forwarded over the network, e.g. over SSH), or virtual displays such as `Xvfb`, etc. The basic syntax to specify displays is `HOST:NUMBER`; if you omit the `HOST` part, the display is a local one.

Displays are numbered from 0, so `:0` is the first local display that was started. On typical setups, this is what is displayed on the computer's monitor.

Like all environment variables, `DISPLAY` is inherited from parent process to child process. For example, when you log into a GUI session, the login manager or session starter sets `DISPLAY` appropriately, and the variable is inherited by all the programs in the session. When you open an SSH connection with X forwarding, SSH sets the `DISPLAY` environment variable to the forwarded connection, so that the programs that you run on the remote machine are displayed on the local machine. If there is no forwarded X connection (either because SSH is configured not to do it, or because there is no local X server), SSH doesn't set `DISPLAY`.

`xcb` provides easy access to the cut buffers built into every X server. It allows the buffers to be manipulated either via the command line, or with the mouse in a point and click manner. This version is also UTF-8 capable.

2.2.6 Mobaxterm

Mobaxterm is a Windows program that can spin an X server, but it also offers loads of functions that are tailored for programmers, webmasters, IT administrators and pretty much all users who need to handle their remote jobs in a more simple fashion.

2.2.7 Virtual environments


In programming, developers must assure that when sharing a project, the environment in which the programs ran will stay the same from developer to developer. For this issue, one of the solutions would be to include in the project's repository a "recipe" that would include every dependency of that project. In python, these are called requirements files, and are usually saved under the name `requirements.txt`. This is just a preference in the Python community,

```

Extracting audio file from video...

00:03:04: Started new req
<observable_pattern.utils.FrameRequest object at
0x7f5779863cd0>.

00:03:08: Inference for request with id 000304 ==> result:
02342.



Person found: "Marilen Gabriel Pirtea".

00:03:08: Current identity: 02342.

```

Figure 2.20: Example of a Markdown log file

but other names can also be chosen.

2.2.8 ffmpeg

FFmpeg is the most used multimedia framework, capable of decoding, encoding, transcoding, muxing, demuxing, streaming, filtering, and playing almost everything that humans and machines have developed. It can handle everything from the most obscure old formats to the most cutting-edge modern ones. Regardless of whether they were created by a standards body, a community, or a corporation. It's also extremely portable: FFmpeg compiles, runs, and passes our FATE testing infrastructure on Linux, Mac OS X, Microsoft Windows, BSDs, Solaris, and other operating systems, as well as a wide range of build environments, machine architectures, and configurations.

2.2.9 Markdown

For generating reports in a nice format, Markdown was chosen. An example can be seen in figure 2.20. Its format is easily programmable, and it suits this application since for debugging purposes, each input that is served to the neural network is saved. Thus, they can be referenced in the Markdown file, which can be compiled into a nicer format like PDF for execution reports of the script.

2.2.10 git

This project was versioned on a Gitlab server on a new branch called "identity_banner", from which new feature branches were created and merged whenever each independent features was done.

Chapter 3

The neural network integration

In this chapter, I describe the steps taken in order to integrate the existing neural network and the obstacles faced in doing so, and the additions brought to the project which helped in tackling most of them.

3.1 Shortcomings of the raw implementation

When building a first and minimal version of the application, many problems were raised by almost random scenarios brought by sections of the recording which weren't handled in the training of the used neural network, since for the training dataset, the best samples were chosen. Thus, a simple loop through all the frames in which a function is called on each one is not enough to achieve real-time implementation, especially on a resource limited machine.

In the following subsections, I will describe what errors the raw implementation brings and how they can be overcome, followed by . In the next chapter, I will describe the application development involved which

3.1.1 Bad face samples

Some errors can occur in two different cases:

- The face polling occurs while a camera transition is happening, like in figure 3.1
- The face detection function fails to provide a face, as shown in figure 3.2

For the first case, nothing can be done in order to prevent it. This is an exception in the code, which will provide an incorrect prediction unless there is a strong undefined class in the neural network's labels.

The face validation in the data acquisition from the previous project was made using Haar cascades and `cvlib`'s `detect_face` function, which uses `OpenCV`'s `dnn` module with a pre-trained caffemodel. This doesn't produce good enough results for the real time video playing feature.

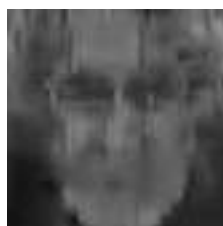


Figure 3.1: Bad face sample



Figure 3.2: Wrong face samples provided by Haar cascades

As seen in figure 3.2, OpenCV's Haar cascades doesn't provide the best results when talking about face validation. A good solution for this problem would be to check if the proposed detected face is placed in the middle of the frame, as it will be presented in the next section.

3.1.2 No undefined class

The current neural network's labels don't have an undefined class. This will cause a lot of wrong predictions if, in the video, there are people which are not trained in the classifier. This will result in a false positive for every person which wasn't trained in the network.

For a computer, the abstract notion of not knowing a face is not yet introduced to it, since it was only taught to recognize them. In order to implement this feature, a dataset comprised of some random number of samples from each class has to be created. This way, in the training phase, the classifier will get as "confused" as when it will see the new person, so it will pick the `undefined` category.

This feature wasn't implemented in this version of the project.

3.1.3 Inference time in a blocking flow

In a real-time application, blockers and high processing times for our functions must be avoided. As expected, the neural network inference takes the most time to complete, especially on a machine without a CUDA enabled GPU.

The application can run in two modes: video player and video writer. For the video player mode, very low waiting time is expected.

3.1.4 Other persons talking over

As seen in figure 3.3, these are cases where, in the 3-second audio captured before the frame was taken, the main speaking person is not the one seen in the shot, thus creating a conflict.



Figure 3.3: An example of an identification when a person is talking over



Figure 3.4: Politician wearing mask

3.1.5 Masks covering faces during the pandemic

As seen in figure 3.4, the recent pandemic brought another problem for the neural network used. Since it uses a multimodal architecture, future development may decouple the 2 classifiers in order to use just the audio classifier, since the image classifier will bring small to none contribution to the prediction.

3.2 Improvements & additions

A first idea which comes to mind would be to see if there are some visible differences which could help the development process. As the meeting proceeds, no more than 3 possible scenarios can be depicted from the recordings. Those 3 scenarios would be the plenum angle, the amphitheater angle and the presidium camera angle, as seen in figure 3.5. As a first step,



Figure 3.5: Camera angles existing in a recording

the last 2 scenes can be combined into one class, since the binary classification of the plenum and the non-plenum frames is wanted for this demo application.

3.2.1 Creating the dataset

The main issue acknowledged in the previous section was that the system implemented for face detection was not a consistent one, and it provided false negatives, as shown before in figure 3.2.

For overcoming that issue and to totally discard the use of Haar cascades in our classifier, a new solution must be implemented. To do that, a good training dataset must be created by using a simple script which collects frames at a given frequency or condition was written and then run on a meeting video.

Ironically, the method which would be discarded, namely the Haar cascades classifier, was used in the logic of this script previously described, but manual cleaning of the obtained images had to be done after the script execution.

In figure 3.7, samples from the resulted dataset can be seen. A good proposal for this dataset would be to apply a crop on each image, in order to capture the middle of the frame, which would remove some of the pre-existing banners which were placed already on the recordings. As seen in figure 3.6, this crop would exclude the pixel patterns which would result in a false positive.

The script ran for a good period of time, in which a total of 7000 of photos were acquired after filtering, out of which 4000 photos were plenum photos and the rest of 3000 were non-plenum samples. In figure 3.7, some samples from each class are presented.

The code for the extraction script can be analyzed in Appendix A.



Figure 3.6: A proposal for a good crop region which removes unwanted objects



Figure 3.7: The final dataset for a plenum classifier

3.2.2 Analyzing kNN for plenum detection

This is a good candidate, but because of the large overhead caused by the in-memory saving of the training data and the heavy processing of the algorithm, the algorithm wasn't implemented in the demo of the application. However, it is left here for the good accuracy obtained by a simple and robust machine learning algorithm.

As seen in table 3.1, the best k for this classifier would be 3 of 7. Of course, the lower the k , the less the processing would be used by the algorithm, so k would be chosen in this scenario.

k	train_acc	test_acc
1	1.0	0.9459
3	0.9868	0.9433
5	0.9842	0.9420
7	0.9815	0.9433
9	0.9808	0.9420
11	0.9796	0.9407

Table 3.1: Train and test accuracy comparison between different values for k

This classifier was trained just on 10 batches out of the entire dataset, where the chosen batch size was 32. Since the dataset was loaded using Tensorflow's `ImageDataGenerator.flow_from_directory` function, the resulted generator had to be converted back to an array of images just for the kNN classifier, since its model wasn't realized using the Tensorflow API, thus the compatibility was missing. So, the main reason of using only 10 batches wasn't just the high training time, but also the transformation of a Tensorflow generator to a numpy array.

These 10 batches come out to be just around 320 photos from both classes. It's a good thing to add that these results are not cross-validated and the batches are randomly created each time from a fixed seed in order to maintain results in the development phase.

Nonetheless, the obtained results were very good for a simple algorithm like kNN.

3.2.3 Implementing a CNN classifier for plenum detection

By creating a standard architecture for image classification, the CNN provided good results at the cost of a high number of trainable parameters, as compared to the previous solutions.

As seen in figure 3.8, the CNN has the following layers:

- Input layer: this is just the input photo after pre-processing (like resizes and normalizing the values of the pixels). For this diagram, this is just one photo instead of an entire batch.
- 1st Convolutional + Max pooling layer: It has the 32 filters of dimensions of 3 by 3. The depth for the resulted volume is 96.
- 2nd Convolutional + Max pooling layer: It has the 64 filters of dimensions of 3 by 3. The depth for the resulted volume is 3072.

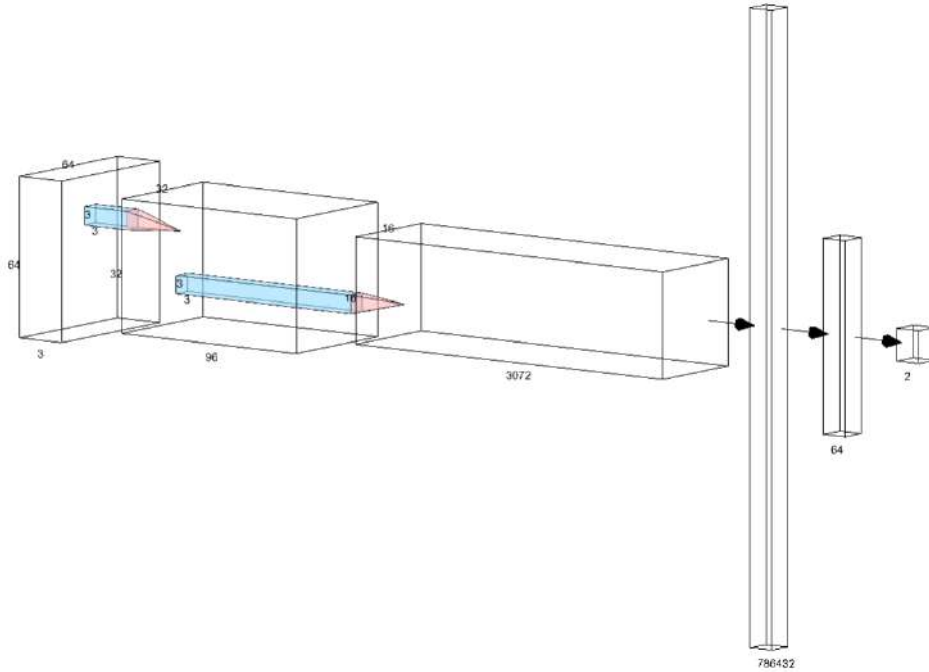


Figure 3.8: Used CNN architecture for the plenum classifier

- Flatten layer: This will just flatten the entire volume from the previous layer and result in a layer of 1 by $16 \times 16 \times 3072$.
- Fully Connected layer of size 64.
- Output layer composed of a fully connected layer with 2 labels. If the highest value is in the 1st label, the outcome is the plenum class, and otherwise for the non-plenum label.

The training was done over 10 epochs, and a checkpoint callback was used, such that the weights were overwritten only with values for which a better accuracy on the validation set was obtained, or if the accuracy on the validation remained the same and the training accuracy increased. For the values in table 3.2, the first checkpoint occurred after the first epoch, and the second checkpoint after the second epoch.

checkpoint	train_acc	test_acc
1	0.99502	1.0
2	0.99941	1.0

Table 3.2: Train and test accuracy for the plenum CNN

3.2.4 Implementing the YOLO network

The choice for implementing YOLO came as a consequence of the poor performance of the cascade classifier. Also, YOLO is a SSD (Single Shot Detector), which is different from the sliding window classifiers, and it was not approached in this project, so it is interesting to see different concepts in actions.

Weights of the YOLOv3 that are trained on the WIDER face dataset can be found on the internet. The resulted code was modified after a public GitHub repository [28] and was adjusted to the application.

It uses the `dnn` module from OpenCV. As described in the theory part of this algorithm and as seen in F, the function `get_one_face` takes the output of the classifier and looks at the score for each class, which are located from index 5 of the output array till the end of the array. After it performs NMS on all the detections, the function checks if there is only one detection left, namely a face. If that is the case, it proceeds to extract the coordinates in order to return a slice operator which will be used in the frame handler.

Chapter 4

Application development

As a specification for the application, the following requirements must be met:

- Offer a video player interface in which the applied banners can be seen in real time.
- Offer log files which will store every request containing the frame on which the algorithm ran, the detected face (if any) and the 3 seconds audio segment.
- Offer summaries of the executions in a nicer format than the one offered by the raw log files.

Next, a flowchart will be presented, which describes the functional decisions of the code, broken in 2 pieces:

- The code structure for the video player mode, as seen in figure 4.1
- The frame handler, as seen in figure 4.2

One of the differences between the video player and the video writer consists of the FFmpeg `image2pipe` pipeline which redirects through the STDIN stream frames and save them one by one, in order, in the output file. This output file doesn't contain audio yet, but it will be added at the end using FFmpeg again. Another difference is that the code execution is not asynchronous anymore in the video writer mode, since there is no waiting time between the frames like in the video player. Instead, Every function is blocking, but the code execution is faster.

Also, the video writer turns all the asynchronous code into synchronous code, thus the results from the two modes are different.

This chapter will be structured in a journal manner in which I will describe every step in the process which lead to the final product.

4.1 Script for testing inference

A trained neural network, as seen by a computer, is made out of the layers of that network and the weights of each neuron-to-neuron connection and the biases of each neuron. The weights and biases are saved by the developer when the training phase has finished in a file, and they are bound to that exact architecture in which the neurons were placed. Weights and biases cannot be loaded on a network structure other than the one from which it resulted.

For an inference test, the model built using the Tensorflow API in loaded, and on top of it the weights with which the best accuracy was achieved in the previous project are loaded.

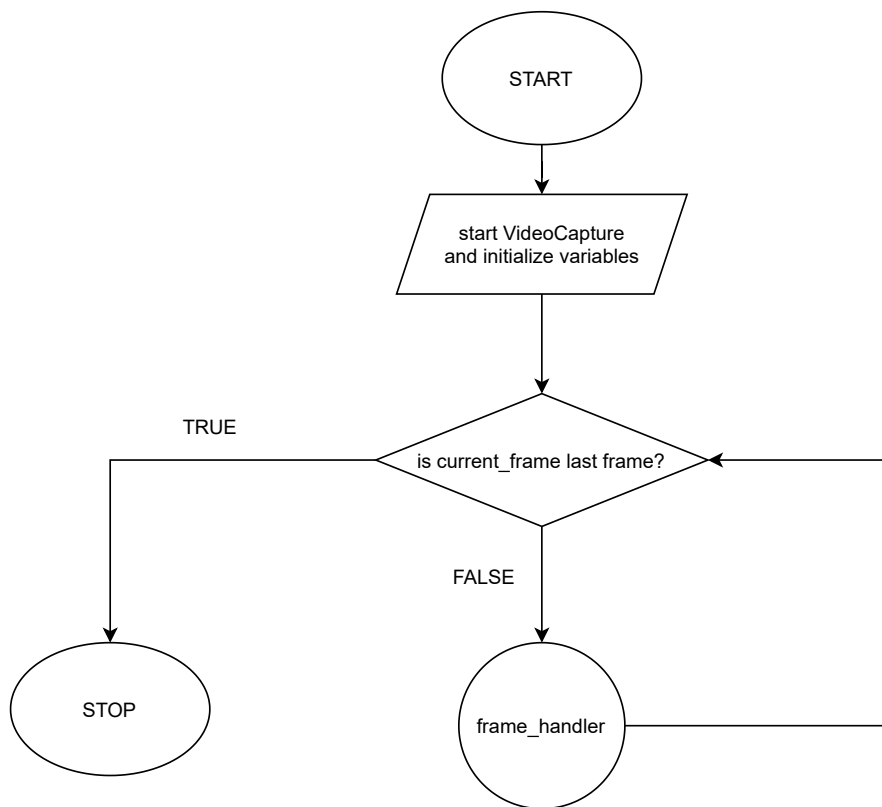


Figure 4.1: Video player main function

Before doing this, all the dependencies and modules used in the original project must be loaded in a virtual environment. A virtual environment ensures a context which can be easily replicable by other developers on their machines, as stated in a previous chapter.

The video and audio inputs were picked randomly from the test database populated in the previous project, in order to not bother with the processing in the first step. This is needed since it must be assured that the environment was passed along without any errors from the previous project.

4.2 Script for audio trimming

Python offers a module called `wave`, which provides a convenient interface to the WAV (Waveform Audio File) sound format. First, the position must be set on the audio frame which corresponds to the video frame. Knowing the current frame index, the relative location is computed by dividing the frame index by the FPS, then multiplying the result with the audio sampling frequency, and then rounding it down. That way, the first audio frame is known.

To gather the next 3 seconds of audio, the number of audio frames is first computed, and the `setpos` function from the `wave` module is used, in order to start the audio capture from that position. Next, the `readframes` function is used to get the next $3 * fs$ frames.

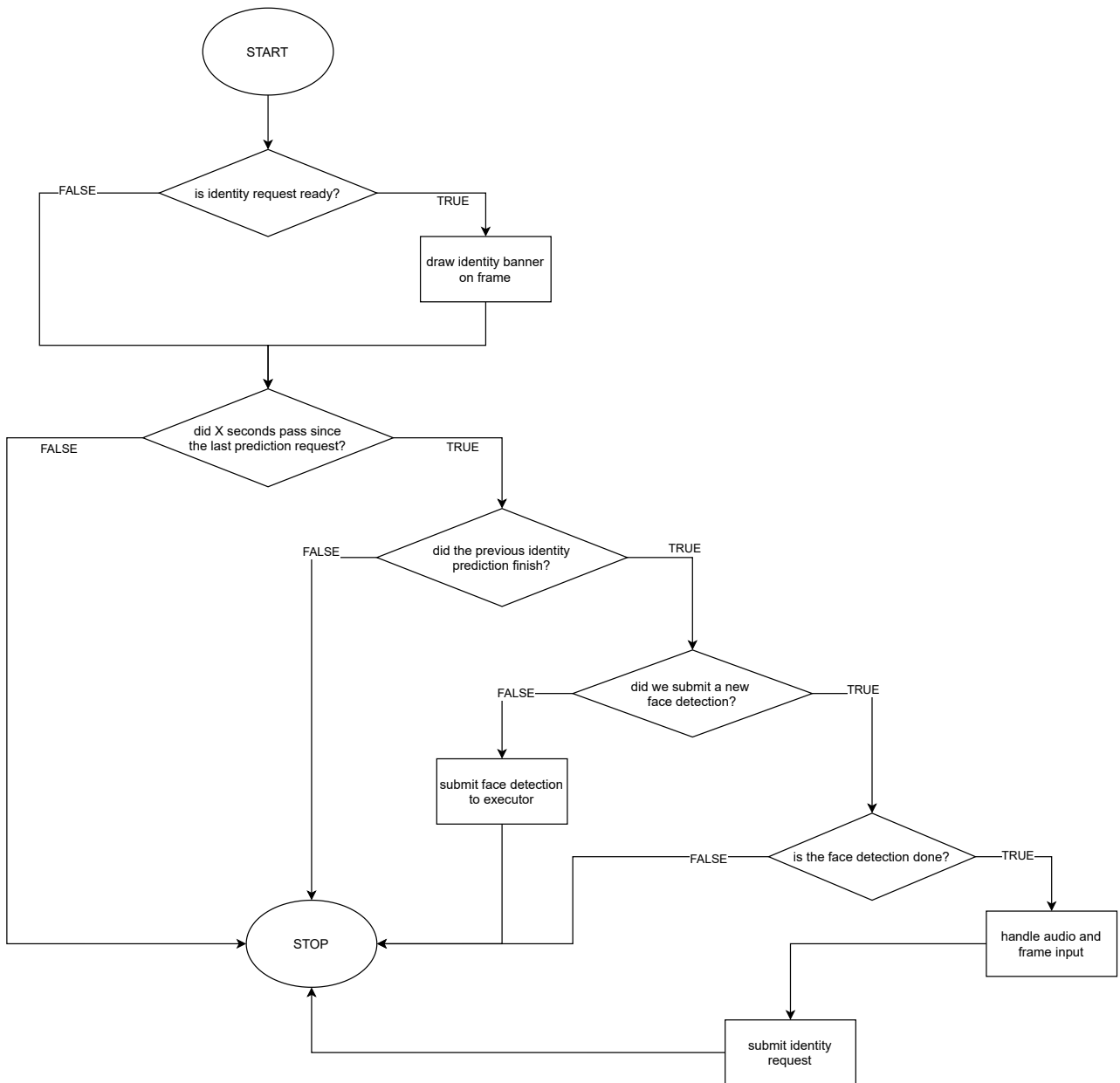


Figure 4.2: Frame handler flowchart

4.3 Adapt input data to network's input format

If a cropped frame containing a face and a 3 seconds audio is simply passed to the input layer of the neural network, the Tensorflow API will throw an error, saying that the input must respect a certain format. That format is exactly the one in which the network was trained.

To ensure compatibility, the two inputs must be processed in a Python script. The steps in the training process must be traced back and reapplied to the inputs, for both image and audio.

Usually, when talking about images in neural networks, they are resized to a fixed size, like 64 by 64 for example (usually a multiple of 2, but this is not mandatory). Also, color images have three channels, red, green and blue, channels which hold 8-bit pixels. Thus, the pixels can take values between 0 and 255. To lower complexity, these values are normalized such that the values will be transferred between 0 and 1.

When talking about audio, a way to quantify and extract important features from it must be found. To acquire the voiceprint characteristics from an audio byte sequence, the spectrogram is obtained from applying the STFT (Short-time Fourier Transform) algorithm on the sequence. The resulted spectrogram is made out of complex values, where the real part is the magnitude and the imaginary part is the phase.

The phase is not of interest, so it is discarded. Each magnitude value must be normalized using the mean and the standard deviation of the series, using the following formula:

$$z = \frac{value - \mu}{\sigma - \epsilon} \quad (4.1)$$

This result is also known as the Z score, or the standard score.

4.4 Script for adding rectangle on frame

The success of the program will be dictated of whether a banner will be placed on the frame on a successful inference. A banner is defined as a rectangle filled with a solid color, over which the correct identity resulted from an inference is written, as seen in figure 4.3.

OpenCV is a good candidate for this job, since the library offers an API for adding shapes and text to an image, which is handled by the OpenCV's API. Since all the frames are provided by the OpenCV's `VideoCapture` interface, each frame is already loaded in an OpenCV instance.

To draw a rectangle, OpenCV offers the `rectangle` function, which accepts as arguments the source image, the top left and bottom right corner coordinates and the desired color of the rectangle.

To place text on an image, `putText` function can be used.

For achieving a well scaled text in the banner's rectangle, the following algorithm must be used. The starting font scale must be one big enough for easy readability, as the algorithm will decrease its size by default, as seen in figure 4.4.



Figure 4.3: Banner showing politician's identity

Algorithm 1: Obtaining a good text scale for the banner

```
while text width is bigger than banner width do  
    decrease font scale by small amount;  
    call getTextSize and update text width;  
end  
place text in the rectangle;
```

Figure 4.4: Pseudocode for obtaining a well-scaled text

4.5 Software design decisions

The code consists of multiple helper classes which are combined into one main class, `Video`, which combines all the classes to form a concern-separated structure. Some helper classes are:

- `Video`
- `VideoOptions`
- `FileCleaner`
- `Logger`
- `FakeFuture`

This structure allows us to separate code based on their objective.

The `Video`'s class constructor accepts the path to the video recording, the options implemented by the `VideoOptions` interface and the path to the file containing the identities, which maps the class names to the real names. See Appendix C for more details.

The `VideoOptions` class binds the keys of a dictionary to attributes of the class with the same name. As a side note, this ensures the IntelliSense feature from VS Code, since simple dictionaries won't have this feature. See Appendix D for more details.

The `FileCleaner` class makes sure to remove helper files (e.g. video files without audio after the merge was done, or the audio files extracted from the input video) after either the end of the program or at a SIGINT invoked by the user (CTRL + C for Windows). In order to listen for the latter, the `KeyboardInterrupt` exception can be used in Python. See Appendix B for more details.

The `Logger`'s class constructor accepts a `Video` instance and takes care of the logging under the hood by accessing paths and the current frame from the `Video` instance's attributes. See Appendix E for more details.

The `FakeFuture` class is used as a wrapper in order to turn asynchronous structure in synchronous structure when the script is run in the video writer mode. It has the class methods `result` and `done`, just like a `Future` object.

4.5.1 Using `concurrent.futures`

As stated in the theoretical chapter, this module is used for asynchronous code execution.

Usually when talking about asynchronous code, an event loop of some sort is implemented under the hood in order to poll/listen and run the callback of the submitted process. Even Python's `asyncio` module has a section in the documentation about the event loop.

Despite what was said earlier, this application isn't using an alternative or built-in event loop, since there is already a loop which can handle all the needs, and that is the frame serving loop. The code is structured such that every time a frame is yielded, `Future` objects are polled and check if they are done or not, as seen in figure 4.5. This way, there is total control over the loop, and custom wrappers can be used in order to decorate these objects.

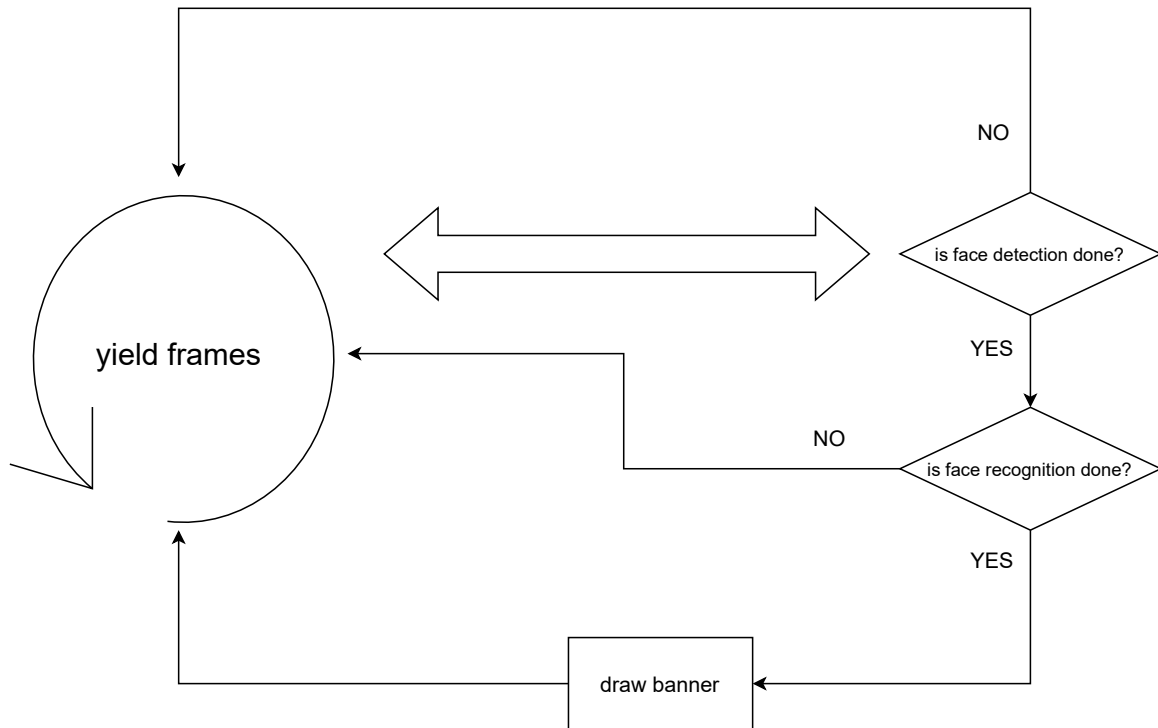


Figure 4.5: Event loop diagram of the application

4.6 Handling stream input

The <http://www.cdep.ro/> website offers the meeting recordings in MP4 format. The stream format can be easily obtained by inspecting the source code of the website and analyze the code which handles the streamed data. This handling happens on the website anyway, since the video has to be buffered in the browser somehow.

The format used in streaming videos is `.m3u8`, more commonly known as `.m3u`, which is also a format for a multimedia playlist. One common use of the `.m3u` file format is creating a single-entry playlist file pointing to a stream from the server.

As it is described in RFC 8216 [25], the HLS distributor must offer a `.m3u8` playlist and `.ts` chunks of the stream. For CDep's case, the initial playlist offers the resource name to another playlist, which holds the chunk list of the stream, as seen in figure 4.6. These chunks are held in `.ts` files, which stands for transport stream.

The playlist format has specific tags, and in figure 4.6 it can be seen that `#EXT-X-TARGETDURATION` has a value of 10, which means that every chunk will have a duration of 10 seconds. In order to implement stream support, frames and audio must be extracted from a fragment, and when all the information from that fragment is consumed, go to the next one and so on, until there are no more chunks to serve by the server.

```
1 #EXTM3U
2 #EXT-X-VERSION:3
3 #EXT-X-TARGETDURATION:10
4 #EXT-X-MEDIA-SEQUENCE:0
5 #EXTINF:9.6,
6 media_w1376649540_0.ts
7 #EXTINF:9.6,
8 media_w1376649540_1.ts
9 #EXTINF:9.6,
10 media_w1376649540_2.ts
11 #EXTINF:9.6,
12 media_w1376649540_3.ts
13 #EXTINF:9.6,
14 media_w1376649540_4.ts
15 #EXTINF:9.6,
16 media_w1376649540_5.ts
17 #EXTINF:9.6,
```

Figure 4.6: Example of chunk list from a CDep resource stream

Chapter 5

Conclusions

5.1 Personal contributions

From the initial listed objectives for the project, a good portion of them were accomplished. The following tasks were completed:

- Designed and implemented a software application that uses an existing multimodal person recognition DNN (Deep Neural Network) to place name banners on frames from videos of politicians giving speeches recordings in the CDep, using an application written in Python which uses the `OpenCV` library
- Managed to save the resulted output video, after the banners were placed successfully
- Debug and log files are created after the execution of the script finishes using Markdown and the Python's `logging` library
- A dataset containing plenum and non-plenum images was created and manually verified, consisting of 7000 images
- Multiple methods to classify the previously mentioned dataset were done by using kNN, Haar Cascades, CNN and YOLO implementations

Two remaining tasks, namely the addition of an `undefined` class and the streaming compatibility, were not completed during this project, remaining as two features which will greatly increase functionality of the application.

5.2 Further steps for development

As for a development overview of the application, its status isn't one close to a production stage, but it provides a good baseline on which further changes can be applied, and different models can be tested.

Some good suggestions and ideas would be to:

- Add an `undefined` class for the classifier
- Add more identities to the classifier
- Add some sort of double check if a face is still in the frame (after the prediction but before the banner drawing) in order to remove scenarios where a banner is drawn, but the person left the camera frame

- Add diacritics for the politicians' names
- Add stream support and buffering in order to not download the entire recording, but rather pipeline the stream into the application
- Test and benchmark application on GPU and add code for compatibility with Tensorflow
- Finish stream support

5.3 Final words

Personally, I feel that this work outlines the problems that most deep learning application will have to face sooner or later in the development chain. Creating a neural network involves a “safe” environment in which the only thing that is trying to be improved is the accuracy of the predictions, which is a good thing, but some steps need to be taken back in order to address problems that appear in time. The majority of these encountered problems will not involve the network itself, but rather miscellaneous steps of integration in the application.

The bigger project has a great potential to become a successful application which can solve a real life problem, and I feel that this work has set a good baseline, described current problems and ideas which can be explored in the next iterations.

Bibliography

- [1] Thorpe S;Fize D;Marlot C;. Speed of processing in the human visual system.
- [2] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features, 2001.
- [3] Gabriel Sandu. Multimodal person identification using deep neural networks. https://speed.pub.ro/speed3/wp-content/uploads/2020/07/Sandu_Marian-Gabriel2020.pdf, 2020.
- [4] Cristian Manolache. Speech recording web service and application. <https://speed.pub.ro/speed3/wp-content/uploads/2017/07/2017-Proiect-Diploma-Manolache-Cristian.pdf>, 2017.
- [5] An introduction to machine learning. <https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning>.
- [6] What is deep learning? <https://machinelearningmastery.com/what-is-deep-learning/>.
- [7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [8] Shan Du and Rabab Ward. Face recognition under pose variations. *Journal of the Franklin Institute*, 343(6):596–613, 2006. Winners of the student paper competition at the 2005 IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP) held in Philadelphia PA, provide the state-of-art in their fields of research.
- [9] Coco set. <https://cocodataset.org/#home>.
- [10] map (mean average precision) for object detection. <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>.
- [11] Yolo: Real-time object detection. <https://pjreddie.com/darknet/yolo/>.
- [12] Cs231n: Convolutional neural networks for visual recognition. <https://cs231n.github.io/>.
- [13] Opencv’s cascade classifier tutorial. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html.
- [14] A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.
- [15] Object detection: using non-max supression over yolov2. <https://medium.com/@sarangzambare/object-detection-using-non-max-supression-over-yolov2-382a90212b51>.
- [16] R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms. <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.

- [17] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.
- [18] Jasper Uijlings, K. Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 09 2013.
- [19] My experiment with unet – building an image segmentation model. <https://analyticsindiamag.com/my-experiment-with-unet-building-an-image-segmentation-model/>.
- [20] Ross Girshick. Fast r-cnn, 2015.
- [21] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.
- [22] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.
- [23] concurrent.futures documentation. <https://docs.python.org/3/library/concurrent.futures.html>.
- [24] Effortless concurrency with python’s concurrent.futures. <https://rednafi.github.io/digressions/python/2020/04/21/python-concurrent-futures.html>.
- [25] R. Pantos and W. May. Http live streaming. RFC 8216, RFC Editor, August 2017.
- [26] An introduction to wsl. <https://dev.to/davehowson/an-introduction-to-wsl-2igf>.
- [27] Developing in wsl. <https://code.visualstudio.com/docs/remote/wsl>.
- [28] Deep learning based face detection using the yolov3 algorithm. <https://github.com/sthanhng/yoloface>.

Anexa A

Code for extracting samples for the training dataset

```

1 import os
2 import cv2
3 import cvlib
4
5
6 def main():
7     if not os.path.isdir('./data'):
8         print("Make sure you create a folder called data which contains a folder named input
9             which contains the video from which you want to extract frames!")
10
11         return
12
13     _PLENNUS_DIR_NAME = "plennus"
14     _NOT_PLENNUS_DIR_NAME = "not_plennus"
15     _PLENNUS_DIR_PATH = os.path.join('./data/output3', _PLENNUS_DIR_NAME)
16     _NOT_PLENNUS_DIR_PATH = os.path.join(
17         './data/output3', _NOT_PLENNUS_DIR_NAME)
18
19     if not os.path.isdir('./data/output3'):
20         os.mkdir('./data/output3')
21
22     if not os.path.isdir(_PLENNUS_DIR_PATH):
23         os.mkdir(_PLENNUS_DIR_PATH)
24
25     if not os.path.isdir(_NOT_PLENNUS_DIR_PATH):
26         os.mkdir(_NOT_PLENNUS_DIR_PATH)
27
28     at_each_frames = 120
29
30     for input_video in os.listdir('./data/input'):
31
32         input_video_path = os.path.join('./data/input', input_video)
33
34         capture = cv2.VideoCapture(input_video_path)
35         total_nr_frames = int(capture.get(cv2.CAP_PROP_FRAME_COUNT))
36         file_name = os.path.basename(input_video_path)
37
38         current_frame_index = -1
39
40         while current_frame_index < total_nr_frames:
41             ret, frame = capture.read()
42             current_frame_index += 1
43
44             if not ret:
45                 break
46
47             if current_frame_index % at_each_frames == 0:
48
49                 if check_if_one_face(frame, True):
50                     cv2.imwrite(os.path.join(
51                         _PLENNUS_DIR_PATH, file_name + f'_{current_frame_index}.jpg'), frame)
52                 else:

```

```
52         cv2.imwrite(os.path.join(_NOT_PLENNUS_DIR_PATH,
53                                 file_name + f'_{current_frame_index}.jpg'), frame
54     )
55
56 def check_if_one_face(frame, cvlib_verif=False):
57     '''
58     Return True/False if only one face in frame
59     '''
60     face_cascade = cv2.CascadeClassifier(
61         cv2.data.harcascades + 'haarcascade_frontalface_default.xml')
62     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
63
64     faces = face_cascade.detectMultiScale(gray, 1.1, 6, 8)
65
66     if len(faces) == 1:
67
68         if cvlib_verif:
69             faces, _ = cvlib.detect_face(frame, threshold=0.5)
70
71             if len(faces) == 1:
72
73                 # x, y, w, h = [v for v in faces[0]]
74
75                 return True
76         else:
77             # x, y, w, h = [v for v in faces[0]]
78
79             return True
80
81     return False
82
83
84 if __name__ == '__main__':
85     main()
```

Anexa B

FileCleaner class

```
1 import os
2 from pathlib import Path
3
4
5 class FileCleaner:
6     def __init__(self) -> None:
7         self.files_to_remove = []
8
9     def add(self, path: Path):
10        self.files_to_remove.append(path)
11
12    def clean(self):
13        not_removed = []
14        for path in self.files_to_remove:
15            if path is None:
16                continue
17
18            path_str = str(path.resolve())
19            if os.path.exists(path_str):
20                os.remove(path_str)
21            else:
22                not_removed.append(path)
23        self.files_to_remove = not_removed.copy()
```

Anexa C

Video class

```

1 from __future__ import annotations
2 from pathlib import Path
3 import cv2
4 from subprocess import Popen, PIPE
5 import subprocess
6 from .logger import Logger
7 from .file_cleaner import FileCleaner
8 import numpy as np
9 import wave
10 from PIL import Image
11 from licenta_2020.utils import handle_wav_input, handle_image_input
12 from .video_options import VideoOptions
13 from .future_wrapper import FakeFuture
14 from concurrent import futures
15 from ..yolo.predict import yolo_wrapper, load_yolo_net
16 from typing import Union
17
18 # print(TYPE_CHECKING)
19
20 # if TYPE_CHECKING:
21 #     from .video_options import VideoOptions
22
23
24 class Video:
25     '''
26     Class which holds video capture generator and audio, and handles each frame based on
27     options
28     '''
29
30     def __init__(self, video_path: Union[Path, str], options: VideoOptions, audio_path=None,
31                 identities_path=None, inference_every=3.0) -> None:
32         """Create a Video instance
33
34         Args:
35             video_path (Union[Path, str]): path to the video. if string, provide .m3u8 URL
36             options (VideoOptions): video options
37         """
38
39         # video related
40         self.options = options
41         self.video_path = video_path
42         self.audio_path = audio_path
43         self.cap = cv2.VideoCapture(str(video_path.resolve()))
44         # self.fps = int(self.cap.get(cv2.CAP_PROP_FPS))
45         self.fps = 25
46         self.frame_width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
47         self.frame_height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
48
49         # let models decorate the Video instance
50         self.options.options.init_plen_classifier(self)
51
52         # parallel process
53         if self.options.mode == VideoOptions.PLAY_VIDEO:

```

```

51         self.executor = futures.ThreadPoolExecutor(max_workers=1)
52     self.predict_identity_future = None
53     self.predict_one_face_future = None
54
55     # services
56     self.logger = Logger(self)
57     self.cleaner = FileCleaner()
58
59     # frame related
60     self.last_frame_check_at = 0
61     self.current_frame_index = -1
62     self.total_nr_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
63
64     # identity related
65     self.current_identity = None
66     self.log_identity_once = False
67     self.inference_every = inference_every
68
69     if identities_path:
70         self.identities = create_identities_dict(identities_path)
71
72     if audio_path is None:
73         self.audio_path = self.video_path.with_suffix('.wav')
74
75     self.extract_audio()
76
77     self.run = None
78
79     if self.options.mode == VideoOptions.PLAY_VIDEO:
80         print('playing video')
81         self.run = self.play_video
82     elif self.options.mode == VideoOptions.WRITE_VIDEO:
83         self.run = self.write_video
84
85     def play_video(self):
86         '''
87         Break into scenarios in order to not slow down main while loop
88         This function will run when the options
89         '''
90
91         try:
92             while self.current_frame_index < self.total_nr_frames:
93                 self.current_frame_index += 1
94                 ret, self.current_frame = self.cap.read()
95
96                 if not ret:
97                     break
98
99                 self.current_option = self.options.options
100
101                 # common code:
102                 self.common_frame_handler()
103
104                 cv2.imshow(self.current_option.window_name,
105                             self.current_frame)
106                 cv2.waitKey(
107                     int(1000 / self.fps))
108
109         except KeyboardInterrupt:
110             self.cap.release()
111             self.cleaner.clean()
112
113     self.cap.release()
114     self.cleaner.clean()

```

```

115
116 def write_video(self):
117     '''
118     Break into scenarios in order to not slow down main while loop
119     This function will run when the options param is a dict
120     '''
121     self.current_option = self.options.options
122
123     output_no_audio_path = self.video_path.parent / (self.video_path.stem +
124                                                     f"_output_without_audio_{self.
current_option.window_name}.mp4")
125
126     output_with_audio_path = self.video_path.parent / (self.video_path.stem +
127                                                       f"_output_{self.current_option.
window_name}.mp4")
128
129     out = Popen(['ffmpeg', '-y', '-f', 'image2pipe', '-vcodec', 'mjpeg', '-r', str(
130                 self.fps), '-i', '-', '-vcodec', 'h264', '-q:v', '5', '-r', str(self.fps), '-vf',
f'scale={self.frame_width}x{self.frame_height},setsar=15:11,setdar=20:11', str(
131                 output_no_audio_path.resolve())], stdin=PIPE)
132
133     try:
134         while self.current_frame_index < self.total_nr_frames:
135             self.current_frame_index += 1
136             ret, self.current_frame = self.cap.read()
137
138             if not ret:
139                 break
140
141             self.current_option = self.options.options
142
143             # common code:
144             self.common_frame_handler()
145
146             Image.fromarray(
147                 self.current_frame[:, :, :-1], 'RGB').save(out.stdin, 'JPEG')
148
149             self.cap.release()
150
151     except KeyboardInterrupt:
152         self.cleaner.add(output_no_audio_path)
153         self.cleaner.clean()
154
155         out.stdin.close()
156         out.wait()
157
158     if self.current_frame_index == self.total_nr_frames:
159         out.stdin.close()
160         out.wait()
161
162         self.logger.create_markdown_log()
163
164         self.cleaner.add(output_no_audio_path)
165
166         print("Merging audio and video files...")
167         cmd = f"ffmpeg -i {output_no_audio_path} -i {str(self.video_path.resolve())} -c
copy -map 0:0 -map 1:1 -shortest {output_with_audio_path}"
168
169         subprocess.call(cmd, shell=True)
170
171 def common_frame_handler(self):
172     self.handle_identity_future()

```

```

173     if self.time_passed() - self.last_frame_check_at > self.inference_every and self.
predict_identity_future is None:
174         # do another frame_req
175
176         # norm_frame = self.current_frame[:, :, :-1] * 1./255
177         # res_norm_frame = cv2.resize(norm_frame, dsize=(
178         #     64, 64), interpolation=cv2.INTER_CUBIC)
179         # classifier_input = np.expand_dims(res_norm_frame, axis=0)
180
181         # print(classifier_input.shape)
182         # classifier_output = np.argmax(
183         #     self.current_option.plen_classifier(classifier_input))
184
185         # index 0 is for not_plennus
186         # index 1 is for plennus
187
188         # is_plen = True if classifier_output == 1 else False
189
190     if self.options.mode == VideoOptions.PLAY_VIDEO:
191         if self.predict_one_face_future == None:
192             self.predict_one_face_future = self.executor.submit(
193                 self.get_one_face)
194             face = None
195         else:
196             face = self.predict_one_face_future.result()
197             self.predict_one_face_future = None
198     else:
199         if self.predict_one_face_future == None:
200             self.predict_one_face_future = FakeFuture(
201                 self.get_one_face()
202             )
203             face = None
204         else:
205             face = self.predict_one_face_future.result()
206             self.predict_one_face_future = None
207
208
209     if face is not None:
210
211         audio, audio_raw, audio_options = self.get_last_3s_audio()
212
213         self.logger.debug_data(
214             face, audio_raw, audio_options, self.generate_request_id())
215
216         audio = handle_wav_input(audio)
217         face = handle_image_input(face)
218
219         X = [face, audio]
220
221         # if we run as video player, we need to run async
222         if self.options.mode == VideoOptions.PLAY_VIDEO:
223             self.predict_identity_future = self.executor.submit(
224                 self.current_option.identity_predict,
225                 X,
226                 self.current_option.identity_model,
227                 self.current_option.identity_labels,
228             )
229         else:
230             self.predict_identity_future = FakeFuture(
231                 self.current_option.identity_predict(
232                     X,
233                     self.current_option.identity_model,
234                     self.current_option.identity_labels
235                 )

```

```

236         )
237
238     else:
239         self.logger.info(
240             f"{self.time_passed_human_readable():} No person found in frame.")
241         print(
242             f"{self.time_passed_human_readable():} No person found in frame.")
243         self.last_frame_check_at = self.time_passed()
244         self.current_identity = None
245
246     def handle_identity_future(self):
247         if self.predict_identity_future is not None and self.predict_identity_future.done():
248             self.log_identity_once = True
249             self.last_frame_check_at = self.time_passed()
250             self.logger.info(
251                 f"{self.time_passed_human_readable():} Inference for request ==> result: {self
                .predict_identity_future.result()}." )
252
253             # TODO: implement None class
254             if self.predict_identity_future.result() == -1:
255                 self.logger.info(
256                     f"Undefined class: {self.predict_identity_future.result()}." )
257                 self.current_identity = None
258
259             else:
260                 self.logger.info(
261                     f"Person found: \"{self.identities[self.predict_identity_future.result()
                    ]['name']}\"." )
262                 self.current_identity = self.predict_identity_future.result()
263
264                 self.predict_identity_future = None
265
266             if self.current_identity is not None:
267                 # print(identities[current_identity]["name"])
268                 self.draw_banner_on_frame()
269
270             if self.log_identity_once:
271                 self.log_identity_once = False
272                 self.logger.info(
273                     f"{self.time_passed_human_readable():} Current identity: {self.
                    current_identity}." )
274                 print(
275                     f"{self.time_passed_human_readable():} Current identity: {self.
                    current_identity}." )
276             else:
277                 # TODO: handler unknown class identity
278                 pass
279
280     def time_passed(self):
281         return self.current_frame_index / self.fps
282
283     def time_passed_human_readable(self):
284         t = self.time_passed()
285         m = str(int(t//60)).zfill(2)
286         s = str(int(t % 60)).zfill(2)
287         ms = str(int((t * 100) % 100)).zfill(2)
288
289         return f"{m}:{s}:{ms}"
290
291     def generate_request_id(self):
292         t = self.time_passed()
293         m = str(int(t//60)).zfill(2)
294         s = str(int(t % 60)).zfill(2)
295         ms = str(int((t * 100) % 100)).zfill(2)

```

```

296
297     return f"{m}{s}{ms}"
298
299 def extract_audio(self):
300     '''
301     Extract single channel, 16k sampling rate, 16bit audio => br = 16k * 1 * 16 == 256k
bit rate
302     '''
303     # stringify pathlib object
304     video_path_str = str(self.video_path.resolve())
305     audio_path_str = str(self.audio_path.resolve())
306
307     self.logger.info(f"Extracting audio file from video...")
308     command = f"ffmpeg -i {video_path_str} -ab 256k -ac 1 -ar 16000 -vn {audio_path_str}"
309     subprocess.call(command, shell=True)
310
311     self.cleaner.add(self.audio_path)
312
313 def get_last_3s_audio(self):
314     '''
315     Get [duration] (or maximum) seconds of audio which ends at the same time as a
specified frame from a video, using [frame_index].
316     '''
317
318     audio_path_str = str(self.audio_path.resolve())
319
320     duration = 3
321     with wave.open(audio_path_str, 'rb') as f:
322
323         fs = f.getframerate()
324         sample_width = f.getsampwidth()
325         relative_loc = int(self.current_frame_index / self.fps)
326
327         # print(fs, sample_width, relative_loc)
328
329         nr_frames_in_duration = duration * fs
330         last_audio_frame = relative_loc * fs
331         first_audio_frame = max(
332             last_audio_frame - nr_frames_in_duration + 1, 0)
333
334         # print(first_audio_frame, nr_frames_in_duration, last_audio_frame)
335
336         f.setpos(first_audio_frame)
337         raw_data = f.readframes(nr_frames_in_duration)
338         audio_options = (
339             f.getnchannels(),
340             sample_width,
341             fs,
342             nr_frames_in_duration,
343             'NONE',
344             'not compressed'
345         )
346
347         # Convert buffer to float32 using NumPy
348         audio_as_np_int16 = np.frombuffer(raw_data, dtype=np.int16)
349         audio_as_np_float32 = audio_as_np_int16.astype(np.float32)
350
351         # Normalise float32 array so that values are between -1.0 and +1.0
352         max_int16 = 2**15
353         float_output = audio_as_np_float32 / max_int16
354
355     # if output_path:
356     #     with wave.open('./output.wav', 'wb') as g:
357     #         g.setnchannels(f.getnchannels())

```

```

358         #         g.setsampwidth(sample_width)
359         #         g.setframerate(fs)
360         #         g.writeframes(raw_data)
361
362         return float_output, raw_data, audio_options
363
364     def get_one_face(self):
365         '''
366         Return RGB face from a frame containing the plen
367         '''
368         return self.current_option.plen_classifier(self)
369
370     def draw_banner_on_frame(self):
371         '''
372         Draw a rectangle banner at the bottom of a photo/frame and save result to output_path
373         If text is provided, place it in the banner
374         If output_path is not provided, return the output
375         '''
376
377         unicode_dict = {
378             [U+FFFD] 'a',
379             [U+FFFD] 'A',
380             [U+FFFD] 'a',
381             [U+FFFD] 'a',
382             [U+FFFD] 'i',
383             [U+FFFD] 'o',
384             [U+FFFD] 'o',
385             [U+FFFD] 't',
386             [U+FFFD] 's',
387             [U+FFFD] 'S'
388         }
389
390         safe_text = f"{self.identities[self.current_identity]['name']}"
391
392         for old, new in unicode_dict.items():
393             safe_text = safe_text.replace(old, new)
394
395         new_frame = np.copy(self.current_frame)
396         height, width = self.current_frame.shape[:2]
397
398         width_offset = 0.25
399         top_offset = 0.25
400         bot_offset = 0.05
401
402         top_left = (
403             int(width_offset * width),
404             int((1 - top_offset) * height)
405         )
406
407         bot_right = (
408             int((1 - width_offset) * width),
409             int((1 - bot_offset) * height)
410         )
411
412         cv2.rectangle(new_frame, top_left, bot_right, (255, 0, 0), -1)
413
414         if safe_text:
415             font = cv2.FONT_HERSHEY_SIMPLEX
416             font_scale = 1
417             font_color = (0, 0, 255)
418             thickness = 2
419
420             (text_width, text_height), _ = cv2.getTextSize(
421                 safe_text, font, font_scale, thickness)

```

```

422
423     # start with a decent font size, but decrease it for longer names
424
425     while text_width > (bot_right[0] - top_left[0]):
426         font_scale -= 0.1
427         (text_width, text_height), _ = cv2.getTextSize(
428             safe_text, font, font_scale, thickness)
429
430     text_position = (
431         (top_left[0] + bot_right[0] - text_width) // 2,
432         (top_left[1] + bot_right[1]) // 2,
433     )
434
435     cv2.putText(
436         new_frame,
437         safe_text,
438         text_position,
439         font,
440         font_scale,
441         font_color,
442         thickness
443     )
444
445     self.current_frame = new_frame
446
447
448 def create_identities_dict(path: Path):
449     obj = {}
450     with open(str(path.resolve()), 'r') as f:
451         for line in f.readlines():
452
453             [code, name, link] = map(str.strip, line.split(', '))
454
455             obj[code] = {
456                 "name": name,
457                 "link": link
458             }
459
460     return obj

```

Anexa D

VideoOptions class

```
1
2 class VideoOptions:
3     PLAY_VIDEO = 0
4     WRITE_VIDEO = 1
5
6     def __init__(self, mode, option) -> None:
7         self.mode = mode
8
9         if self.mode == VideoOptions.PLAY_VIDEO:
10            self.options = PlayerOption(option)
11        elif self.mode == VideoOptions.WRITE_VIDEO:
12            self.options = WriterOption(option)
13
14
15 class PlayerOption:
16
17     def __init__(self, obj: dict) -> None:
18         # TODO: validation schema
19
20         self.window_name = obj['window_name']
21         self.init_plen_classifier = obj['init_plen_classifier']
22         self.plen_classifier = obj['plen_classifier']
23         self.identity_model = obj['identity_model']
24         self.identity_labels = obj['identity_labels']
25         self.identity_predict = obj['identity_predict']
26
27
28 class WriterOption:
29
30     def __init__(self, obj: dict) -> None:
31         # TODO: validation schema
32
33         self.window_name = obj['window_name']
34         self.init_plen_classifier = obj['init_plen_classifier']
35         self.plen_classifier = obj['plen_classifier']
36         self.identity_model = obj['identity_model']
37         self.identity_labels = obj['identity_labels']
38         self.identity_predict = obj['identity_predict']
```

Anexa E

Logger class

```

1 from __future__ import annotations
2 import os
3 import logging
4 import re
5 from typing import Optional, TYPE_CHECKING
6 import wave
7 import cv2
8 import time
9 from .video_options import VideoOptions
10
11 if TYPE_CHECKING:
12     from .video import Video
13
14
15 class Logger:
16     '''
17     Class which logs based on the Video instance passed and saves debug data like face shots
18     and audio files
19     '''
20
21     def __init__(self, video_instance: Video, log=True, debug=True) -> None:
22         mode = 'play' if video_instance.options.mode == VideoOptions.PLAY_VIDEO else 'write'
23
24         self.logs_folder_path = video_instance.video_path.parent / \
25             (video_instance.video_path.stem + f'_logs_data_{mode}_{video_instance.options.
26             options.window_name}_{int(time.time())}')
27         logger = logging.getLogger(self.logs_folder_path.parts[-1])
28
29         self.log = log
30         self.debug = debug
31
32         if self.log:
33
34             if not os.path.isdir(str((self.logs_folder_path).resolve())):
35                 os.mkdir(str((self.logs_folder_path).resolve()))
36
37             logger.setLevel(logging.INFO)
38
39             # set new logger each time we call this function
40
41             fh = logging.FileHandler(
42                 str((self.logs_folder_path / 'logs.txt').resolve()), 'w')
43             fh.setLevel(logging.INFO)
44             fh.setFormatter(logging.Formatter('%(message)s'))
45
46             logger.addHandler(fh)
47         else:
48             logging.disable(logging.INFO)
49
50         if self.debug:
51             if not os.path.isdir(str((self.logs_folder_path / 'audio').resolve())):
52                 os.mkdir(str((self.logs_folder_path / 'audio').resolve()))

```

```

51
52     if not os.path.isdir(str((self.logs_folder_path / 'photos').resolve())):
53         os.mkdir(
54             str((self.logs_folder_path / 'photos').resolve()))
55
56         self.audio_debug_folder_path = self.logs_folder_path / 'audio'
57         self.photos_debug_folder_path = self.logs_folder_path / 'photos'
58
59     self.logger = logger
60
61     def info(self, text):
62         self.logger.info(text)
63
64     def debug_data(self, face, audio_raw, audio_options, req_id):
65         if self.debug:
66             # save audio and face with frame_req's id
67             img_debug_path = str((self.photos_debug_folder_path /
68                                 (req_id + '.jpg')).resolve())
69
70             audio_debug_path = str((self.audio_debug_folder_path /
71                                    (req_id + '.wav')).resolve())
72
73             # print(img_debug_path)
74             # print(audio_debug_path)
75
76             with wave.open(audio_debug_path, 'wb') as audio_file:
77                 audio_file.setparams(audio_options)
78                 audio_file.writeframesraw(audio_raw)
79
80             cv2.imwrite(img_debug_path, face)
81
82     def create_markdown_log(self):
83         if not self.debug:
84             return
85
86         logs_md_path_str = str((self.logs_folder_path / 'logs.md').resolve())
87         logs_txt_path_str = str((self.logs_folder_path / 'logs.txt').resolve())
88
89         with open(logs_md_path_str, 'w') as f:
90             f.write(f'# Markdown for {self.logs_folder_path.stem}\n')
91
92         with open(logs_txt_path_str, 'r') as g:
93             for line in g.readlines():
94                 f.write(line + '\n')
95
96                 regex = re.search(
97                     r'\d{2}:\d{2}:\d{2}: Inference for request with id (\d{6}) ==> result:
(\d{5}).', line)
98
99                 if regex and regex.group(1) and regex.group(2):
100                     f.write(
101                         f"![[regex.group(1)]](./photos/{regex.group(1)}.jpg)" + '\n\n')

```

Anexa F

YOLO predictor

```

1 from __future__ import annotations
2 from numpy.lib import index_tricks
3 import cv2
4 from typing import TYPE_CHECKING
5 import numpy as np
6 from pathlib import Path
7 import time
8
9
10 if TYPE_CHECKING:
11     from identity_banner.services.video import Video
12
13 CONF_THRESHOLD = 0.5
14 NMS_THRESHOLD = 0.4
15
16 # Get the names of the output layers
17
18
19 def get_outputs_names(net):
20     # Get the names of all the layers in the network
21     layers_names = net.getLayerNames()
22
23     # Get the names of the output layers, i.e. the layers with unconnected
24     # outputs
25     return [layers_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
26
27
28 def get_one_face(frame, frame_height, frame_width, outs, conf_threshold, nms_threshold):
29
30     print('started yolo face detection')
31     # Scan through all the bounding boxes output from the network and keep only
32     # the ones with high confidence scores. Assign the box's class label as the
33     # class with the highest score.
34     confidences = []
35     boxes = []
36
37     for out in outs:
38         for detection in out:
39             scores = detection[5:]
40             class_id = np.argmax(scores)
41             confidence = scores[class_id]
42             if confidence > conf_threshold:
43                 center_x = int(detection[0] * frame_width)
44                 center_y = int(detection[1] * frame_height)
45                 width = int(detection[2] * frame_width)
46                 height = int(detection[3] * frame_height)
47                 left = int(center_x - width / 2)
48                 top = int(center_y - height / 2)
49                 confidences.append(float(confidence))
50                 boxes.append([left, top, width, height])
51
52     # Perform non maximum suppression to eliminate redundant

```

```

53     # overlapping boxes with lower confidences.
54     indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold,
55                               nms_threshold)
56
57     print('finished yolo face detection')
58
59     if len(indices) == 1:
60         i = indices[0][0]
61         box = boxes[i]
62
63         left = box[0]
64         top = box[1]
65         width = box[2]
66         height = box[3]
67
68         return frame[top:top+height, left:left+width]
69     return None
70
71
72 def load_yolo_net(video_instance: Video):
73     parent_folder = Path(__file__).parent
74
75     model_cfg = str((parent_folder / 'cfg' / 'yolov3-face.cfg').resolve())
76     model_weights = str((parent_folder / 'model-weights' /
77                          'yolov3-wider_16000.weights').resolve())
78
79     net = cv2.dnn.readNetFromDarknet(model_cfg, model_weights)
80     net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
81     net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
82
83     video_instance.yolo_net = net
84
85     return net
86
87
88 def yolo_wrapper(video_instance: Video):
89
90     frame = video_instance.current_frame
91     frame_height = video_instance.frame_height
92     frame_width = video_instance.frame_width
93     net = video_instance.yolo_net
94
95     # print(frame_width)
96     # print(frame_height)
97
98     blob = cv2.dnn.blobFromImage(frame, 1 / 255, (frame_width, frame_height),
99                                  [0, 0, 0], 1, crop=False)
100
101     # Sets the input to the network
102     net.setInput(blob)
103
104     t = time.time()
105     print('start yolo fwd')
106     # Runs the forward pass to get output of the output layers
107     outs = net.forward(get_outputs_names(net))
108     print(f"end yolo fwd: {time.time() - t}")
109
110     return get_one_face(frame, frame_height, frame_width, outs, CONF_THRESHOLD, NMS_THRESHOLD)

```