Lebanese University
Faculty of Technology

Communications and Computer Networks Engineering

Senior Project

Fighting Covid-19:

detection of facemask wearing

Submitted By

Ali Al Hadi Ayache

Supervisor

Dr. Rola El Osta

Academic Year: 2020-2021

Acknowledgements

I dedicate my graduation project to my dear family, may God keep it to my heart, and to those who contributed to raise up my educational and cultural level.

My thanks also go to Dr. Rola El Osta who assisted me and supervised my project, as she provided me with advice and instructions. Last but not least, I express my thanks and deep appreciation to all my instructors who accompanied me along the university years.

Contents

List of Figures iv

1 Introduction 1

2 Literature Review 5
- 2.1 Introduction . . . 5
- 2.2 Deep Learning . . . 5
- 2.3 Object Detection . . . 6
  - 2.3.1 CNN-Based Object Detectors . . . 6
  - 2.3.2 Face Detection . . . 8

3 YOLO: the real-time mask detector 11
- 3.1 Introduction . . . 11
- 3.2 The YOLO Network . . . 13
- 3.3 The Loss Function . . . 13
- 3.4 Comparing YOLO Versions . . . 13
- 3.5 YOLOv2 (YOLO9000) . . . 14
- 3.6 YOLOv3 . . . 14
- 3.7 YOLOv4 . . . 15
- 3.8 YOLOv5 . . . 15

4 Methodology and Implementation 17
- 4.1 Introduction . . . 17
- 4.2 Experimental Platform . . . 17
  - 4.2.1 Frameworks and Libraries . . . 17
  - 4.2.2 Arduino . . . 18
  - 4.2.3 Temperature Sensors . . . 19
  - 4.2.4 Arduino Buzzer . . . 21
  - 4.2.5 Camera Types . . . 21
  - 4.2.6 Arduino LCD I2C . . . 23
- 4.3 Methodology . . . 23
- 4.4 System Requirement . . . 24
- 4.5 Implementation: Software phase . . . 25
  - 4.5.1 Collecting data . . . 26
  - 4.5.2 Train face mask detector . . . 26
  - 4.5.3 Apply face mask detector . . . 28
- 4.6 Implementation: Hardware Phase . . . 29
  - 4.6.1 Preparing Circuit of Arduino . . . 29
- 4.7 Results . . . 30
- 4.8 Conclusion . . . 30

5 Conclusions and Future Work **33**

Appendices **35**

A Pyhthon Code **37**

B Arduino Code **41**

List of Figures

Figure 1.1 Blog Diagram of the work. 3

Figure 2.1 Artificial Intelligence subfields 6
Figure 2.2 Summary of R-CNNs 8
Figure 2.3 Face Detection 10

Figure 3.1 The YOLO Algorithm 12
Figure 3.2 yolo design 13
Figure 3.3 YOLO 16

Figure 4.1 Arduino Uno 19
Figure 4.2 dht-111 19
Figure 4.3 lm35 20
Figure 4.4 MLX90614 20
Figure 4.5 Buzzer 21
Figure 4.6 Logitech C920 22
Figure 4.7 Microsoft LifeCam HD-3000 22
Figure 4.8 IP Camera 23
Figure 4.9 IP webacm 23
Figure 4.10 LCD 24
Figure 4.11 Real Time Face detection 24
Figure 4.12 Flowchart 25
Figure 4.13 VoTT 27
Figure 4.14 Roboflow 27
Figure 4.15 ROI 28
Figure 4.16 circuit:1 29
Figure 4.17 Arduino circuit 30
Figure 4.18 Real-Time face Detection 31

Chapter 1

Introduction

According to World Health Organization (WHO), since December 2019 more than 114 countries suffered from COVID-19 pandemic which has declared as a deadly diseases that has globally infected over 110 million people causing more than 2.43 million deaths in the worldwide as on Feb 18, 2021. Wearing a face mask during this pandemic is a critical defensive in times when social distancing is hard to maintain. Therefore, many face mask detection and monitoring systems have been developed to provide effective supervision for hospitals, public transportations, airports, retail locations, and sports venues. Over years, in the field of image processing, computer vision and pattern recognition, face detection is the very first step for various applications that depends facial analysis algorithms for identifying, recognizing human faces and also to capturing facial motions in digital images, including the face recognition, face alignment, face verification, age recognition, face modelling, face authentication, face relighting, facial expression tracking, head pose tracking, facial expression recognition, gender recognition, and other face-detection based applications.

After the arise of Covid-19, the Face-Mask detection has widely considered problem in the image processing field. Face-mask detection represents both a detection as well as a classification problem because it requires first the location of faces of people in digital images and then the decision of whether they are wearing a mask or not. The first part of this problem has been studied extensively in the computer vision literature, due to the broad applicability of face-detection technology []. The second part, on the other hand (i.e., predicting whether a face is masked or not), has only gained interest recently, in the context of the COVID-19 pandemic. Consequently, mask detection has become a vital computer vision task to help the global society.

In this work, we will create a automation process for detecting the faces and distinguishing between a naked face and a face wearing a mask from an image. We will introduce a face-mask detection model that is based on computer vision and deep learning. The model can be integrated with surveillance cameras to impede the COVID-19 transmission by allowing the detection of people who are wearing masks not wearing face masks. This will be done by retrieving the video stream from a camera in real time, and then providing images to the detection system. The system will automatically classify faces as masked or not using an existing neural network. If there is no mask, trigger a preventive alert.

Creating a system for detecting the face-mask will provide a way for controlling the people who enters any places. The proposed system in this paper uses deep learning, TensorFlow, Keras, and OpenCV which are used as an image classifier to detect face-mask and sends a signal to Arduino device that control the open and close of door.

The system is an integration of a new deep learning object detection technique, called "YOLO" and libraries like OpenCV and Numpy which are used as an image classifier to detect face-mask and send a signal to Arduino device to control the Buzzer alert. As it achieves the highest accuracy and consumes the least time in the process of training and detection, we extended our work to measure the human body temperature and display it on LED with an Infrared temperature sensor (see Figure 1.1). The sensor also provides accurate results. Our model is implemented on an embedded system.

The remainder of this report is organized as follows: In Chapter 2, we briefly review related work in object detection and face mask detection. The proposed mask detection method is introduced in Chapter 3, and we present the experimental results in Chapter 4. Finally, we conclude the report in Chapter 5.

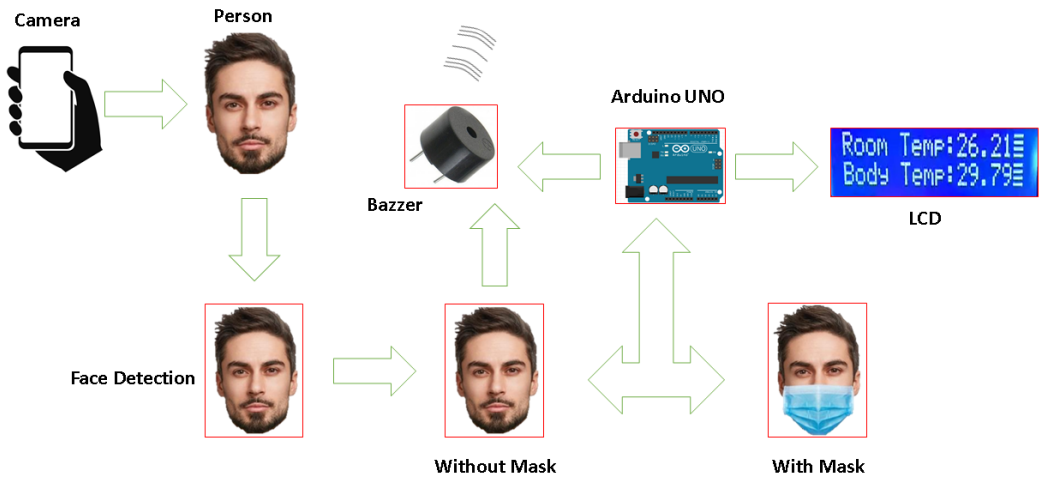

Figure 1.1: Blog Diagram of the work.

Chapter 2

Literature Review

2.1 Introduction

Object detection is a fundamental task in computer vision and widely applied in the applications of face detection , skeleton detection , pedestrian detection and sign detection. Face detection is always a hotspot in object detection. Due to the outbreak of the pandemic, face mask detection has attracted much attention and research. In this first chapter, we are going to present the main lines of our work. We will also present the famous detection methods used.

2.2 Deep Learning

Deep Learning (DL) is basically a subpart of Machine Learning (ML) model which involves algorithms that concerned with algorithms inspired by the structure and function of the brain and uses multilayer neural networks called artificial neural networks. Basically, both DL and ML belongs to the higher field Artificial Intelligence (AI) as shown in Fig.2.1. DL depends on Neural Network layers that transforms the input in some way to produce output to implement its functioning. In deep learning, Image can be called as "matrix of pixel values", therefore it can be an more easier to classify complex images matrix or images with similar forms of matrix or a very huge dataset of images with minimal changes in the matrix by depending on deep learning matrix classification. This may lead to clash in prediction scores and thereby affecting the accuracy and speed of classifier model [**?**].

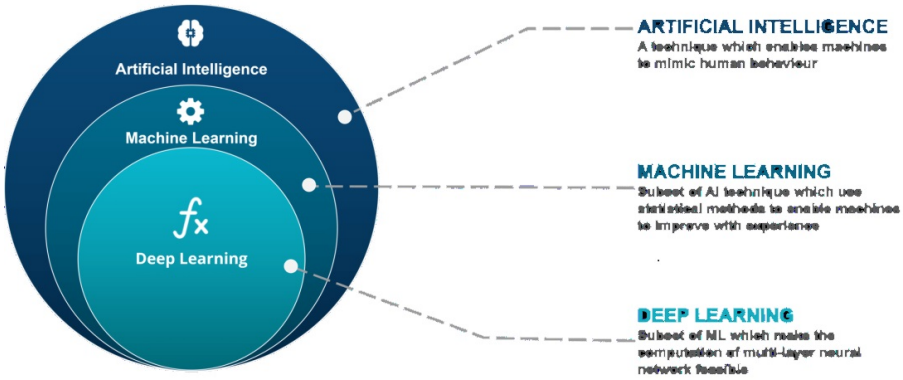

Figure 2.1: Artificial Intelligence subfields

2.3 Object Detection

2.3.1 CNN-Based Object Detectors

The Convolutional Neural Network (CNN) is a class of deep neural networks that are inspired by biological processes. A CNN consists of a series of building blocks, such as the convolutional layer, pooling layer, and fully connected layer, and is capable of learning automatically and adaptively learning the spatial hierarchies of features through a backpropagation algorithm. The kernels of a CNN are shared across all the image positions, which makes it highly efficient with respect to the parameters. These properties make the CNN an ideal solution for computer vision tasks. In recent years, with the great improvement of GPU computing power, deep learning technology has been booming. AlexNet, the champion model of the ImageNet Competition in 2012, utilizing a GPUfor computing acceleration, has brought a rebirth to deep learning. As an important task in computer vision, object detection has also been greatly promoted. According to the proposed recommendations and improvement strategies, the current object detection methods can be divided into two categories: two-stage detectors and one-stage detectors.

2.3.1.1 Two-Stage Detectors

The two-stage method first generates a large number of region proposals for each image through a heuristic algorithm or CNN and then classifies them and regresses these

candidate regions. As the earliest object detection method based on deep learning, R-CNN belongs to a two-stage detector. R-CNN firstly adopts selective search to generate a sparse set of candidate regions. Then, it extracts the features of the region through CNN and finally determines the class of each object by SVMand fine-tunes the bounding boxes using linear regression. Other common models are Fast R-CNN, Faster R-CNN, SPPNet, Feature Pyramid Network (FPN), Cascade R-CNN, etc. Although the two-stage detectors have good detection performances, their training phases are complex, and the testing speeds are usually slow, so they are not suitable for the real-time requirement.

2.3.1.2 One-Stage Detectors

In 2015, You Only Look Once (YOLO), proposed by R. Joseph et al., broke the domination of two-stage detectors in object detection. Unlike the two-stage methods, YOLO directly classifies and predicts the target at each location on the whole original image. The network divides the image into grids and synchronously predicts the bounding boxes and the probability of each region. Compared with the two-stage detectors, YOLO has extremely fast detection speed with the cost of a reduction in positioning accuracy. Since then, R. Joseph et al. made some incremental improvements and proposed YOLOv2 and YOLOv3. The anchor mechanism, a more powerful backbone network, multiscale trainings, etc., were introduced to further improve YOLO's accuracy and speed. In addition to the YOLO series, SSD, RetinaNet, EfficientDet, and RefineDet are representative models of one-stage object detection. Among them, RetinaNet was proposed by T.-y. Lin et al., in 2017. They discovered that the extreme foreground-background class imbalance during the training phase of dense detectors was the main reason why the one-stage detectors' accuracy was low. To address this issue, focal loss was proposed. Through the reconstruction of the standard cross entropy loss, the detector paid more attention to the samples that were difficult to classify in the training phase, enabling the one-stage detector to achieve comparable accuracy to two-stage detectors while maintaining a high detection speed.

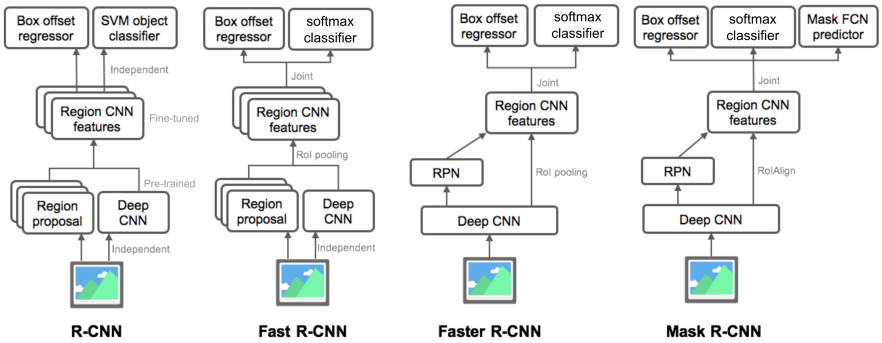

Figure 2.2: Summary of R-CNNs

2.3.2 Face Detection

Face detection is a technique for recognizing or confirming an individual's identity by looking at their face. Face recognition software can identify people in pictures, videos, or in real time.

Over the past 60 years as shown in Fig-1, face detection methods widely used in various industries and have benefitted from the improvements in this technology and these include: law enforcement, border control, retail, mobile technology and banking and finance.

- 1964: Bledso did a facial programming experiment. They imagine a semi-automnmatic input method, in which the operators enter twenty different measures, such as the size of the mouth and eyes
- 1977: 21 new markers were added to the Bledso 1964 system to improved it (i.e. , width of lips, eyes color, hair color).
- 1988: Artificial intelligence was used to improve previously used computational methods that exposed multiple flaws. Mathematics ("linear algebra") are used to view symbols uniquely and to find a way to simplify and modify them independently of human markers.
- 1991: EIGENFACES which was the first successful techniques used in facial recognition technology, that depend on the statistical Principal component analysis (PCA) method, Was developed by Pentland and Turk of the Massachusetts Institute of Technology (MIT)
- 1998: Face recognition technology "FERET program" developed by the Defense

Advanced Research Projects Agency (DARPA), created a database of 2400 images for 850 persons of deferent age and gender.

- 2005: The Face Recognition Grand Challenge (FRGC) was created to promote and improve face recognition technologies that would complement existing facial recognition initiatives.
- 2011: using deep learning and machine learning techniques that depend on artificial neural networks, enables The system to selects a point for the comparison: in large databases.
- 2014: Facebook's internal algorithm, Deepface, allows it to identify faces. According to the social network, the process comes close to matching the output of the human eye in approximately 97% of images.
- 2017: Apple launched a facial recognition technology in its updates, and its use has expanded to retail and banking.
- 2017: Selfie Pay is a facial recognition system for online transactions developed by Mastercard.
- In 2018, Chinese police used a smart monitoring system focused on live facial recognition to arrest a suspect of "economic crime" at a concert where his face was recognized in a crowd of 50,000 people after being identified in a national database.
- From 2019, People who want to purchase a new phone in China will now agree to have their faces scanned by the operator.

The identification of face masks is a difficult task for the existing proposed face detector models. As in all normal face detections systems, face-mask detection system depends on evaluating and comparing the detected face with datasets for finding the face and then determining whether the person wear a mask on it or not. The data used in face mask detection model consists of categories: (1) without face mask, (2) with face mask.

We will describe in the next chapter deeply the object detection method that we use in our project.

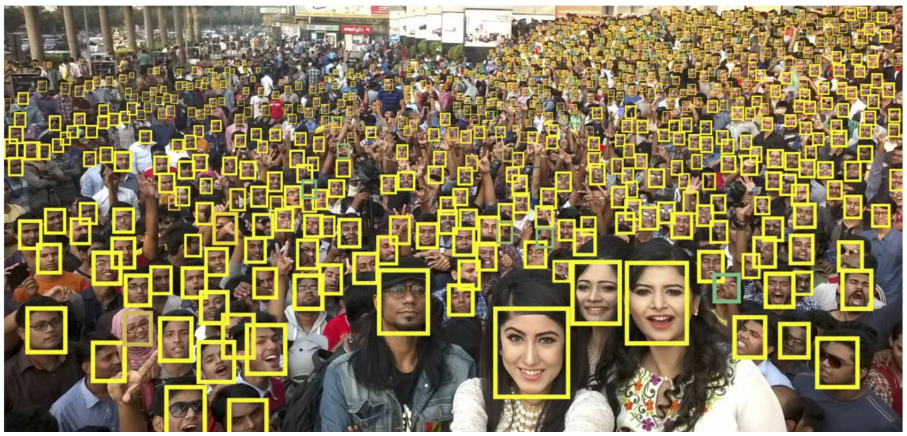

Figure 2.3: Face Detection

Chapter 3

YOLO: the real-time mask detector

3.1 Introduction

You Only Look Once (YOLO) is a network that uses Deep Learning (DL) algorithms for object detection [10,11]. YOLO performs object detection by classifying certain objects within the image and determining where they are located on it.For example, if you input an image of a herd of sheep into a YOLO network, it will generate an output of a vector of bounding boxes for each individual sheep and classify it as such.
How YOLO improves over previous object detection methods: Previous object detection methods like Region-Convolutional Neural Networks (R-CNN), including other variations of it like fast R-CNN, performed object detection tasks in a pipeline of multi-step series. R-CNN focuses on a specific region within the image and trains each individual component separately. This process requires the R-CNN to classify 2000 regions per image, which makes it very time-consuming (47 seconds per individual test image). Thus it, cannot be implemented in real-time. Additionally, R-CNN uses a fixed selective algorithm, which means no learning process occurs during this stage so the network might generate an inferior region proposal. This makes object detection networks such as R-CNN harder to optimize and slower compared to YOLO. YOLO is much faster (45 frames per second) and easier to optimize than previous algorithms, as it is based on an algorithm that uses only one neural network to run all components of the task.
YOLO Architecture Structure Design and Algorithm Operation: A YOLO network consists of three main parts:

1. The algorithm also known as the predictions vector

2. The network

3. The loss function

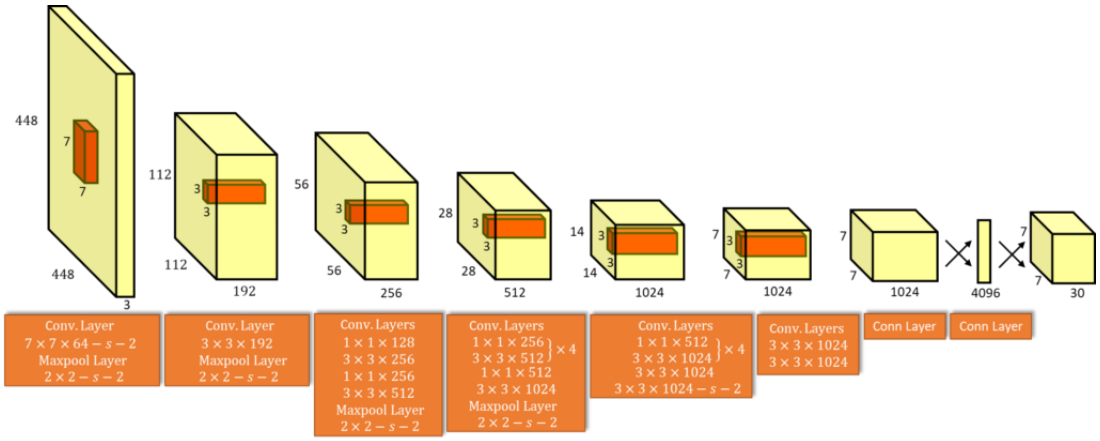

Figure 3.1: The YOLO Algorithm

Once you insert input an image into a YOLO algorithm, it splits the images into an SxS grid that it uses to predict whether the specific bounding box contains the object (or parts of it) and then uses this information to predict a class for the object [10,11]. Before we can go into details and explain how the algorithm functions, we need to understand how the algorithm builds and specifies each bounding box. The YOLO algorithm uses four components and additional value to predict an output.

1. The center of a bounding box (bx by)

2. Width (bw

3. Height (bh)

4. The Class of the object (c)

5. The final predicted value is confidence (pc). It represents the probability of the existence of an object within the bounding box.

6. The (x,y) coordinates represent the center of the bounding box.

Typically, most of the bounding boxes will not contain an object, so we need to use the pc prediction. We can use a process called non-max suppression to remove unnecessary boxes with low probability to contain objects and those who share big areas with other boxes.

3.2 The YOLO Network

A YOLO network is structured like a regular CNN, it contains convolutional and max-pooling layers and then two fully connected CNN layers.

3.3 The Loss Function

We only want one of the bounding boxes to be responsible for the object within the image since the YOLO algorithm predicts multiple bounding boxes for each grid cell. To achieve this, we use the loss function to compute the loss for each true positive. To make the loss function more efficient, we need to select the bounding box with the highest Intersection over Union (IoU) with the ground truth. This method improves predictions by making specialized bounding boxes which improves the predictions for some aspect ratios and sizes.

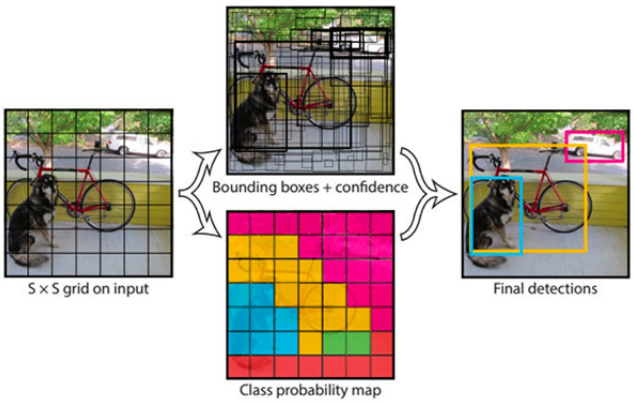

Figure 3.2: yolo design

3.4 Comparing YOLO Versions

The most current version of YOLO is the third iteration of the object detection network. The creators of YOLO designed new versions so to make improvements over previous versions, mostly focusing on improving the detection accuracy.

YOLO v1 was introduced in 2016 by Joseph Redmon et al with a research paper called "You Only Look Once: Unified, Real-Time Object Detection". This was the initial paper by Redmon that revolutionized the industry and changed the Real-Time Object detection methods totally. By just looking at the image once, it can detect the

objects with a speed of 45fps(frames per second), another YOLO v1 type, Fast YOLOv1 was able to achieve 155fps with little less accuracy. It used the Darknet framework that was trained on the ImageNet-1000 dataset. But YOLOv1 has many limitations like it can't detect the objects properly when the objects are smallit also can't generalize the objects if the image is of different dimensions.

3.5 YOLOv2 (YOLO9000)

The second version of YOLOv2 was released in 2017 by Ali Farhadi and Joseph Redmon. This time Joseph collaborated with Ali for major bug fixes and accuracy increment. The research they published was "YOLO9000: Better, Faster, Stronger." The name of the second version of YOLO was YOLO9000. The major competitor of YOLO9000 was Faster R-CNN, which was also an object detection algorithm that uses Region Proposal Network (SSD)Single-shot Multbox Detector to identify the multiple objects from an image. Some of the features of YOLOv2 are: YOLOv2 added Batch Normalization as an improvement that normalizes the input layer of the image by altering the activation functions. Higher-resolution input: input size has been increased from 224*224 to 448*448. Anchor boxes. Multi-Scale training. Darknet 19 architecture with 19 convolution layers and 5 Max Pooling layers.

3.6 YOLOv3

After one year, on March 25, Joseph Redmon and Ali Farhadi came up with another version of YOLO and a research paper called: "YOLOv3: An Incremental improvement." At 320×320, YOLOv3 runs with 22ms at 28.2 mAP with great accuracy, as shown in the above video. It is three times faster than the previous SSD and four times faster than Retina Net. New YOLOv3 followed the methodology of the previous YOLOv2 version: YOLO9000. In this approach, Redmond uses Darknet 53 architecture, which was a significantly improved version and had 53 convolution layers. Some of the new, improved features in YOLOv3 was:

- Class Predictions
- Feature Pyramid Networks(FPN)

- Darknet 53 architecture

3.7 YOLOv4

As Redmond was not currently working on the CV for a long time, a new team of three developers released YOLOv4. It was released by Alexey Bochoknovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Alexey is the one who developed the Windows version of YOLO back in the days. YOLOv4 runs twice faster than EfficientDet with comparable performance Some of the new features of YOLOv4 is:

- Anyone with a 1080 Ti or 2080 ti GPU can run the YOLOv4 model easily.
- YOLOv4 includes CBN(Cross-iteration batch normalization) and PAN(Pan aggregation network) methods.
- Weighted-Residual-Connections(WRC).
- Cross-Stage-Partial connections(CSP), a new backbone to enhance CNN(convolution neural network).
- Self-adversarial-training(SAT): A new data augmentation technique DropBlock regularization.

3.8 YOLOv5

After a few days of the release of the YOLOv4 model on 27 May 2020, YOLOv5 got released by Glenn Josher(Founder CEO of Utralytics). Glenn introduced the YOLOv5 Pytorch based approach, and Yes! YOLOv5 is written in the Pytorch framework. It is state of the art and newest version of the YOLO object detection series, and with the continuous effort and 58 open source contributors, YOLOv5 set the benchmark for object detection models very high; as shown below, it already beats the EfficientDet and its other previous YOLO versions.

Figure 3.3: YOLO

Chapter 4

Methodology and Implementation

4.1 Introduction

In this chapter, we will talk about the methodology of our project which consists of detecting the faces of people from a video sequence stream. The input to our app is video footage. First of all, we decompose the video into a set of images, and from this set we look for the best choice which corresponds to the image in which the search algorithm has detected the correspondence between the current image and the set of images recorded in the database therefore the recognition algorithm verifies the existence of the person sought in the database.

4.2 Experimental Platform

In what follows, we will explain the reasons for our choices as to the most appropriate development tools for our different needs, which will allow us to achieve our objectives as well as possible and as quickly as possible, as well as to carry out the development phase. It's the experimental platform.

4.2.1 Frameworks and Libraries

A deep learning framework is an interface, library or a tool which allows us to build deep learning models more easily and quickly, without getting into the details of underlying algorithms. They provide a clear and concise way for defining models using a collection of pre-built and optimized components [4]. In deep learning we have many libraries:

- OpenCV

- NumPy
- TensorFlow
- Keras

In our project we use **OpenCV** and **Numpy**:

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. ... The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. All of the new developments and algorithms appear in the C++ interface. There are bindings in Python, Java and MATLAB/OCTAVE. Later, OpenCV came with both cv and cv2 . Now, there in the latest releases, there is only the cv2 module, and cv is a subclass inside cv2 .

NumPy or Numerical Python is an open-source Python library that makes it easy to complex numerical operations. Working with machine learning and deep learning applications involve complex numerical operations with large datasets. ... In contrast, a NumPy array can only contain objects of the same data type.

4.2.2 Arduino

The Arduino UNO is the best board to get started with electronics and coding. If this is your first experience tinkering with the platform, the UNO is the most robust board you can start playing with. The UNO is the most used and documented board of the whole Arduino family.

Arduino Uno is a microcontroller board based on the ATmega328P .It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.. You can tinker with your Uno without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few

dollars and start over again. "Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards.

Figure 4.1: Arduino Uno

4.2.3 Temperature Sensors

DHT11

The DHT11 is a digital temperature sensor that measures temperature and relative humidity. These sensors contain a chip that does analog to digital conversion and spit out a digital signal with the temperature and humidity. This makes them very easy to use with any microcontroller, including the Arduino [14].

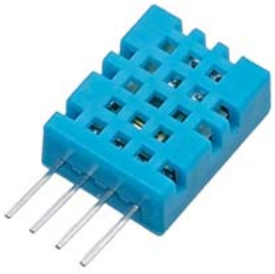

Figure 4.2: dht-111

LM35DZ, LM335, LM34

The LM35DZ is a linear temperature sensor that comes directly calibrated in Celsius. The analog output is directly proportional to the temperature in Celsius: 10 mV per degrees Celsius rise in temperature. This sensor is very similar with the LM335 (calibrated in Kelvin) and with the LM34 (calibrated in Fahrenheit).

Figure 4.3: lm35

MLX90614ESF

The MLX90614 is an infrared thermometer for non-contact temperature measurements. Both the IR sensitive thermopile detector chip and the signal conditioning ASIC are integrated in the same TO-39 can. Integrated into the MLX90614 are a low noise amplifier, 17-bit ADC and powerful DSP unit thus achieving high accuracy and resolution of the thermometer.

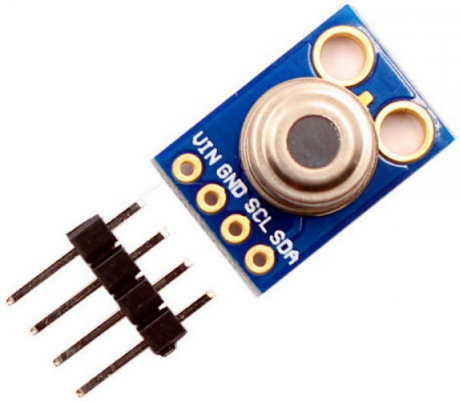

Figure 4.4: MLX90614

4.2.4 Arduino Buzzer

A buzzer or piezo speaker with Arduino. Buzzers can be found in alarm devices, computers, timers and confirmation of user input such as a mouse click or keystroke. You will also learn how to use tone() and noTone() function.

Figure 4.5: Buzzer

4.2.5 Camera Types

There are two types of webcams [15,16]:
(1) basic webcams, made by Microsoft, Logitech, HP, etc.
(2) IP Network Webcams, also known as network cameras

- A USB webcam is a camera that connects to a computer, usually through plugging it in to a USB port on the machine. The video is fed to the computer where a software application lets you view the pictures and also transfer them to the Internet. The downside to any USB type of webcam is the fact that you must have a PC switched on at the location of the camera to enable the images or video feed to be sent to a website.The first webcam was up and running at Cambridge University's Computer Science Department In 1991.

- An IP camera connects directly to your network, Broadband router or Modem. This type of webcam does not need a computer to connect to the Internet as it has all the electronics needed for this built in. This makes the camera very reliable as there is no PC involved or separate operating system. An IP webcam can connect to your router in the conventional wired way using CAT-5 cable or some models also have a wireless connection too. Because there is no requirement for a PC to enable the camera to work on the Internet, IP Webcams are more

reliable and far more economical to run than USB webcams.The first IP camera was released in 1996. It was made by AXIS who are still big in IP cameras today.

IP Camera Types

- Fixed IP Cameras
- Dome IP Cameras
- Wireless IP Cameras
- Indoor IP Cameras / Outdoor IP Cameras
- PTZ IP Cameras (Pan/Tilt/Zoom)

Some of basic webcam

Figure 4.6: Logitech C920

Figure 4.7: Microsoft LifeCam HD-3000

Some of IP webcam

Figure 4.8: IP Camera

Figure 4.9: IP webacm

4.2.6 Arduino LCD I2C

It is great blue back-light LCD display. As the pin resources of Arduino controller is limited, your project may be not able to use normal LCD shield after connected with a certain quantity of sensors or SD card. However, with this I2C interface LCD module, you will be able to realize data display via only 2 wires. If you already has I2C devices in your project, this LCD module actually cost no more resources at all. It is fantastic for Arduino based project.

4.3 Methodology

This project basically consists of a camera placed in front of door, the camera sends video frames to OpenCV running on a Windows PC. If OpenCV detects a face it will capture the image of the people entering public places, then the detected face image is

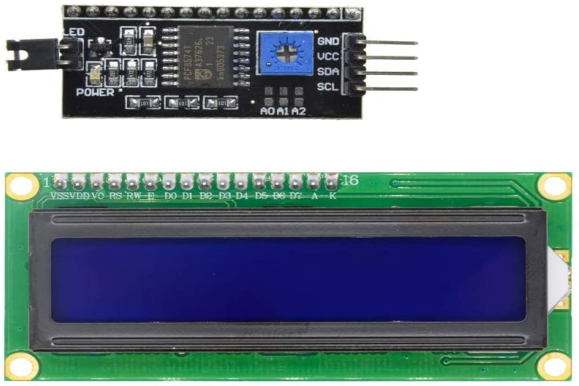

Figure 4.10: LCD

sent to a the system to be compared (tested) whether the person wears a face mask or not using . After the data processing, OpenCV will send the result to Arduino whether to messure a body temperature and alert if not masked.

Figure 4.11: Real Time Face detection

4.4 System Requirement

The system consists of software and hardware phases.

Software phase is responsible for collecting data, loading face mask dataset, training face mask, loading face mask classifier, detecting face in video stream, and extracting face in ROI.

This phase requires the following:

- Python: 3.4 and newer
- PySerial
- numpy==1.18.2

- opencv-python==4.2.0.

Hardware phase is responsible for preparing Arduino-based circuitry. It includes:

- Camera
- Computer
- Buzzer
- LED
- Arduino UNO
- Sensor mlx90614
- preparing circuits of Arduino
- Python/Arduino code

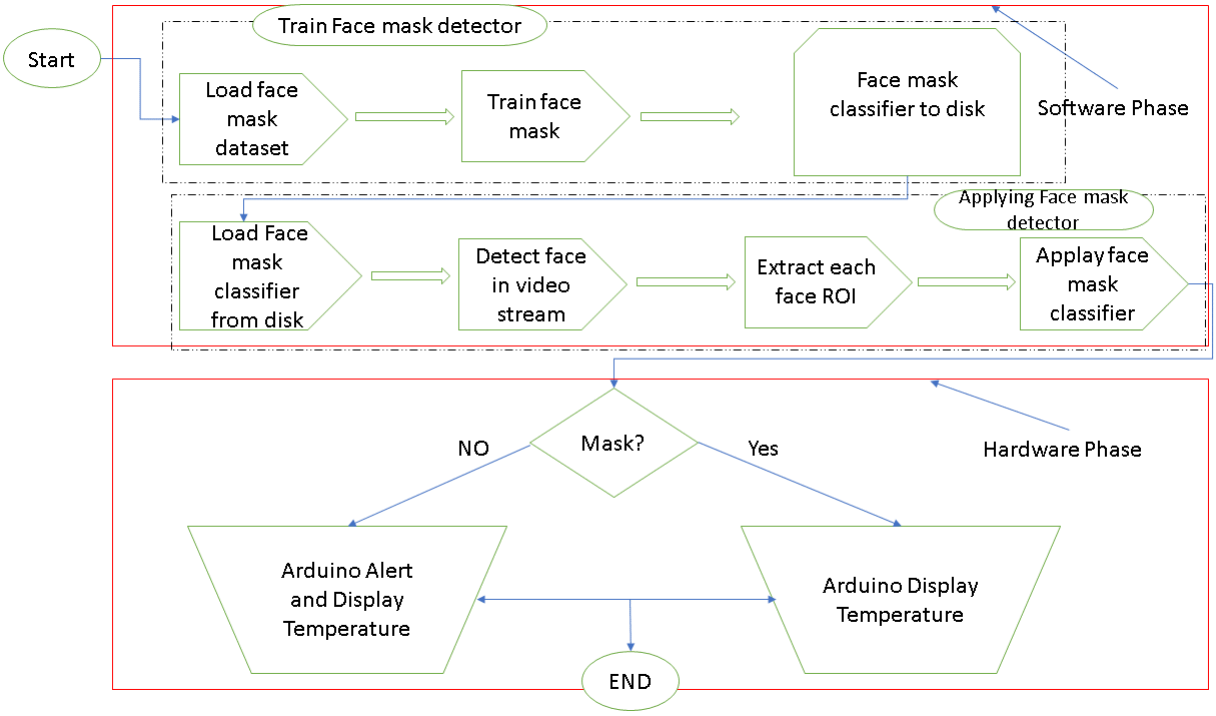

Figure 4.12: Flowchart

4.5 Implementation: Software phase

In this section, we present the implementation and experiments conducted on the OpenCV deep learning framework.

4.5.1 Collecting data

Taking photos and annotating probably one of the most famous quotes defending the power of data is that of Google's Research Director Peter Norvig claiming that "We don't have better algorithms". We collect a large number of data sets with face mask and without masks. Huge is the number of images, higher is the accuracy we get. For that we need a relatively large data set for optimum performance simply because they are able to "learn" more features (almost like gaining experience).so we took 1000 images using the camera of the other one in different positions and either in different distances and backgrounds to be more precise under the training of our Dataset. In order for our detector to learn to detect objects in images, it needs to be fed with labeled training data. So we have to:

- Install Microsoft's Visual Object Tagging Tool (VoTT)
- Annotate the objects inside the photos inserted
- Once labeled images are sufficient, press "CRTL+E" to export the project.

We see now a folder called "vott-csv-export". Within that folder, you also notice a "*.csv file" called "Annotations-export.csv" which contains file names and bounding box coordinates.

4.5.2 Train face mask detector

4.5.2.1 Load face mask dataset

We download the application from https://app.roboflow.com/datasets to be able to create our own Dataset which will be used in training our detector. Then we import our images already annotated and the "*Annotationsexport.csv*" file. We apply the resize preprocessing option $--> $ scratch to 416 x 416, we upload Train/Test to split the dataset and choose "Train/Test Split" of 70%, 20%, 10%. Finally, our Dataset is generated.

4.5.2.2 Train face mask

From the following website https://colab.research.google.com/drive/1pwowg038eognddf6sxdg5asc8pica
scrollto=gnvu7eu9cqj3, we use "$YOLOv4-tiny-DarknetRoboflow.ipynb$" where

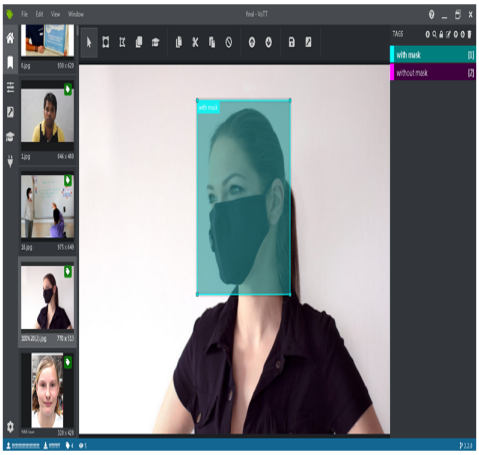

Figure 4.13: VoTT

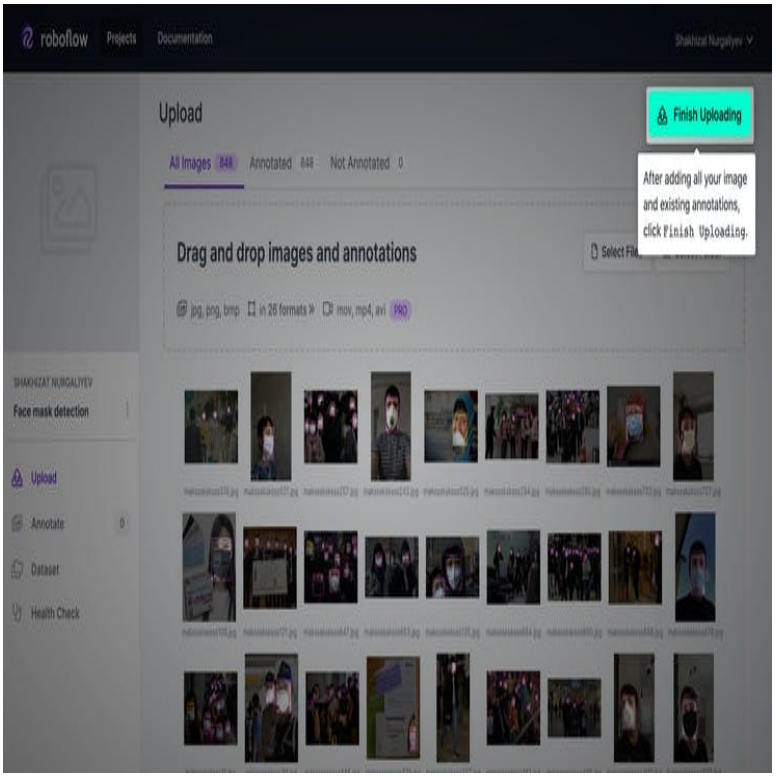

Figure 4.14: Roboflow

we implement the tiny version of YOLOv4 for training on our own dataset, YOLOv4 tiny. We could make the following modifications:
We replace 60 by 30 in env compute capability=60, then we put compute capability=30. In $-- >!sed - i"s/ARCH = -gencodearch = compute_60, code = sm_60/ARCH = -gencodearch = compute_\$computecapability, code = sm$compute capability/g" Make file (replace here 60 by 30: compute_30,code=sm_30). In the part of "SET UP CUSTOM DATASET FOR YOLOV4" we put the link of the export of your dataset. NOTE: YOLO Darknet format.

4.5.3 Apply face mask detector

To apply the face mask detector, the following steps are done:

1. Load face mask classifier: we download the two files:
   - The detector file "custom-yolov4-tiny-detector.cfg" in the cfg directory
   - The weight file "custom-yolov4-tiny-detector_best.weights" in the backup directory
2. Detect face in video stream: we open the Camera in video stream to detect the face
3. Extract face in ROI: Take face in ROI means left and right eyes ROI and mouth ROI

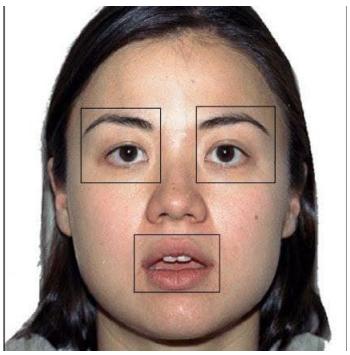

Figure 4.15: ROI

4. Apply face mask classifier: apply face to display it if masked or not to send them into Arduino

4.6 Implementation: Hardware Phase

4.6.1 Preparing Circuit of Arduino

In this circuit we use the following components:

- Arduino UNO
- Wires
- Buzzer
- LCD I2C
- MLX90614 infrared temperature sensor

We connect the SCL and SDA of the MLX sensor to A5 and A4 analog pins, VCC to 5V and GND to GND respectively. The buzzer is connected to pin 8 and GND. It is better to avoid pins 3 and 11 for the Buzzer.

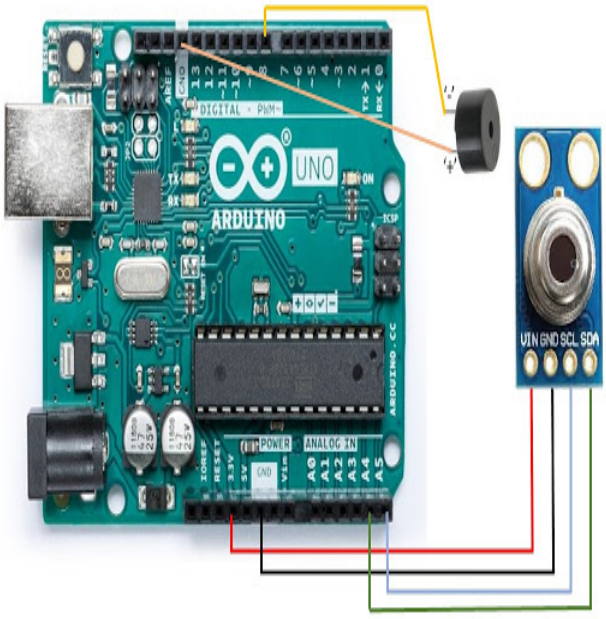

Figure 4.16: circuit:1

We add the LCD i2C to the circuit while respecting the same previous connection. The LCD should display the temperature if the face is masked. Otherwise, the buzzer

gives alert because the face is naked and the LCD display the temperature too by the temperature sensor. In this case, the person must wear him/her face mask. If the face masked only the temperature is displayed on LCD.

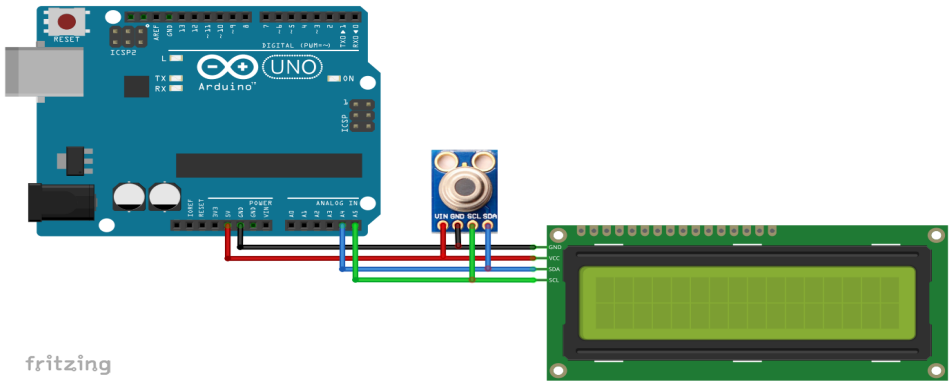

Figure 4.17: Arduino circuit

4.7 Results

As shown in Figure 3.8, the system would respond to people's temperature and the detection results. A buzzer would give alert if the person did not wear the mask properly or his temperature was out of the normal range. The average execution time of the system for processing one image frame was 0.13 s, which showed that our system could detect face masks in real-time and was of value in practical applications.

4.8 Conclusion

In this chapter, we present the methodology and the implementation we followed in our project. We also show the results successfully obtained by detecting if the face is wearing a face mask or no, and also by displaying the relevant body temperature.

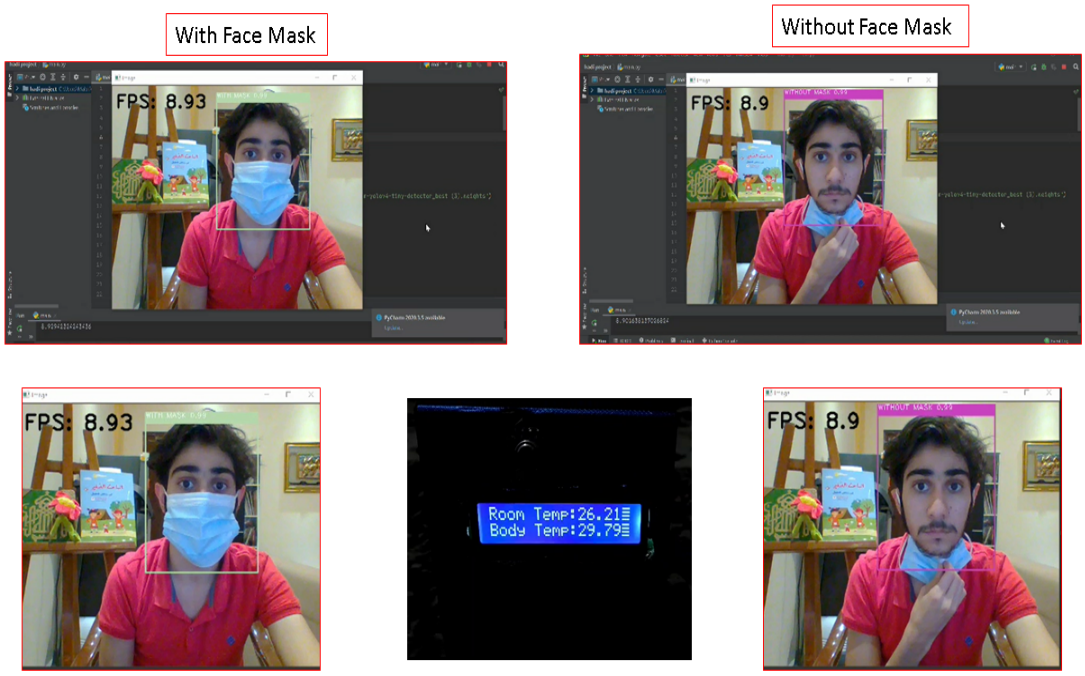

Figure 4.18: Real-Time face Detection

Chapter 5

Conclusions and Future Work

Due to the urgency of controlling COVID-19, the application value and importance of real-time mask detection are increasing. Deep Learning libraries(OpenCv and Others) has been used to detect face and determine whether the person wear a mask or not. Based on these approaches, our project helps the community to know if the person is masked or no with a buzzer alert and a body temperature sensor implemented on Arduino. This can also help in controlling the spreading of COVID-19 in public places.

The implemented system shows a high accuracy in giving the true results In the future, we can upgrade the application and merge it with many different systems, eg. open the door of a mall to permit people wearing mask and having convenient temperature entering or close the door in the contrary case. We can also get benifit of the application by implementing it in ATM machines, schools, universities and many other institutions with high population.

Bibliography

[1] E. Dong, H. Du and L. Gardner, "An interactive web-based dashboard to track COVID-19 in real time", The Lancet Infectious Diseases, vol. 20, no. 5, pp. 533-534, 2020. Available: 10.1016/s1473- 3099(20)30120-1 [Accessed 6 April 2021].

[2] P.S. Othman, R.R. Ihsan, R.B. Marqas, S.M. Almufti, "Image Processing Techniques for Identifying Impostor Documents Through Digital Forensic Examination". Image Process. Tech. 2020, 62, 1781– 1794.

[3] https://www.geospatialworld.net/blogs/difference-between-ai%EF%BB%BF-machine-learning-and-deep-learning/

[4] https://towardsdatascience.com/all-the-numpy-you-need-to-supercharge-your-deep-learni

[5] https://viso.ai/deep-learning/ann-and-cnn-analyzing-differences-and-similaritie

[6] https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-b

[7] https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529

[8] https://d2l.ai/chapter_computer-vision/rcnn.html

[9] https://www.analyticssteps.com/blogs/introduction-yolov4

[10] https://www.analyticssteps.com/blogs/introduction-yolov4

[11] https://analyticsindiamag.com/yolov5/

[12] Abdallah, A.S., Abbott, A.L. and El-Nasr, M.A. A New Face Detection Technique using 2D DCT and Self Organizing Feature Map, Proceedings of World Academy of Science, Engineering and Technology 21, 15-19.

[13] https://store.arduino.cc/usa/arduino-uno-rev3

[14] https://randomnerdtutorials.com/9-arduino-compatible-temperature-sensors-for-your-ele

[15] https://www.voipsupply.com/blog/voip-insider/ip-camera-types/

[16] https://www.techradar.com/news/computing-components/peripherals/what-webcam-5-reviewe

[17] https://www.digilearnings.com/deep-learning/

Appendices

Appendix A

Pyhthon Code

```
import cv2
import numpy as np
import time
import serial
# Load Yolo
arduino = serial.Serial(port='COM4', baudrate=57600, timeout=.1)

net = cv2.dnn.readNetFromDarknet("custom-yolov4-tiny-detector (3).cfg",r"custom-yolo

classes = ['WITH MASK','WITHOUT MASK']

layer_names = net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
colors = np.random.uniform(0, 255, size=(len(classes), 3))

# Loading camera

cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
#address= "http://192.168.43.18:8080/video"
#address= "http://192.168.1.104:8080/video"
#address= "http://192.168.1.101:8080/video"
#cap.open(address)
#cap = cv2.VideoCapture(0)
```

```
font = cv2.FONT_HERSHEY_PLAIN
starting_time = time.time()
frame_id = 0
while True:
_, frame = cap.read()

frame_id += 1

height, width, channels = frame.shape
# Detecting objects
blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)
outs = net.forward(output_layers)

# Showing informations on the screen
class_ids = []
confidences = []
boxes = []
for out in outs:
for detection in out:
scores = detection[5:]
class_id = np.argmax(scores)
confidence = scores[class_id]
if confidence > 0.2:
# Object detected
center_x = int(detection[0] * width)
center_y = int(detection[1] * height)
w = int(detection[2] * width)
h = int(detection[3] * height)

# Rectangle coordinates
x = int(center_x - w / 2)
```

```
y = int(center_y - h / 2)

boxes.append([x, y, w, h])
confidences.append(float(confidence))
class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.4, 0.3)

for i in range(len(boxes)):
if i in indexes:
x, y, w, h = boxes[i]
label = str(classes[class_ids[i]])

def write_read(x):
arduino.write(bytes(x, 'utf-8'))
time.sleep(0.05)
data = arduino.readline()
return data
if True:



#while (port.isOpen()):
if ((label == "WITHOUT MASK") or ((label == "WITHOUT MASK") and (label == "WITH MASK
#temp = '1'
#arduino.write(temp.encode())
#arduino.write('1')
arduino.write(str.encode('1'))
print(label)

else:
# temp = '0'
# arduino.write(temp.encode())
# arduino.write('0')
```

```
arduino.write(str.encode('0'))



#print(label)
#if label == "WITHOUT MASK"
confidence = confidences[i]
color = colors[class_ids[i]]
cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
cv2.rectangle(frame, (x, y), (x + w, y + 20), color, -1)
cv2.putText(frame, label + " " + str(round(confidence, 2)), (x, y + 10), font, 1, (255,255,2



elapsed_time: float = time.time()- starting_time
fps = frame_id / elapsed_time
cv2.putText(frame,"FPS: " + str(round(fps, 2)), (10, 50), font, 3, (0, 0, 0), 3)
print(fps)
cv2.imshow("Image", frame)
key = cv2.waitKey(1)
if key == 27:
break

cap.release()
cv2.destroyAllWindows()
```

Appendix B

Arduino Code

```
int x=0;
#include <Wire.h>
#include <Adafruit_MLX90614.h>
#include <LiquidCrystal_I2C.h>

Adafruit_MLX90614 mlx = Adafruit_MLX90614();

//Define I2C address of lcd, number of columns and rows
LiquidCrystal_I2C lcd(0x27, 16, 2);

double temp_amb;
double temp_obj;



void setup() {
// put your setup code here, to run once:
//pinMode(ledPin, OUTPUT);  // initialize the LED pin as an output
Serial.begin(57600);
//Serial.begin(9600);
Serial.setTimeout(1);
pinMode(3,OUTPUT);
digitalWrite(3,LOW);
```

```
lcd.init();
lcd.backlight();

//Initialize MLX90614
mlx.begin();

}

void loop() {
// put your main code here, to run repeatedly:
if (Serial.available()>0);{
x=Serial.read();
//x = Serial.readString().toInt();
if(x=='1'){
digitalWrite(3,HIGH);

}
else if(x=='0'){
digitalWrite(3,LOW);

}
else {
temp_amb = mlx.readAmbientTempC();
temp_obj = mlx.readObjectTempC();
if(temp_obj > 37) {

Serial.print("Warning...HIGH TEMP...");
lcd.clear();
lcd.print("HIGH TEMP...");
delay(50);}
else{
```

```
}

//lcd display
lcd.setCursor(0, 0);
lcd.print("Room Temp:");
lcd.setCursor(10, 0);
lcd.print(temp_amb);

lcd.setCursor(15, 0);
lcd.write(1);
lcd.setCursor(0, 1);
lcd.print("Body Temp:");
lcd.setCursor(10, 1);
lcd.print(temp_obj);
lcd.setCursor(15, 1);
lcd.write(1);


}
}
}
```