ISSN 1870-4069

Implementation of an Embedded System to Detect the Use of Face Masks Using a CNN Model for the Networking Laboratory at the UQRoo Campus Cancun

María Kristell Flores Reyes, Daniel Sergio Martínez-Ramírez,
Julio Cesar Ramírez-Pacheco, Homero Toral-Cruz,
David Ernesto Troncoso-Romero,
Ismael Osuna-Galán, J. A. León-Borges

Universidad Autónoma del Estado de Quintana Roo,
División de Ciencias, Ingeniería y Tecnología,
Mexico

jleon@uqroo.edu.mx

Abstract. The access to face to face classes at the Autonomous University of the State of Quintana Roo requires the use of a face mask detector as a personal protection measure against any risk of COVID-19 infection or other respiratory diseases in enclosed spaces. As a proposed facemask detector, the present work consists of the implementation of an embedded instrument that performs the detection of face masks inside an enclosed space such as the networking laboratory so that students and teachers have a safer space against possible health risks. Therefore, the implementation of an object detection method was sought to identify whether any student or any staff member is wearing his or her face mask to have access to the laboratory facilities. Therefore, in this research work, we implement an embedded face mask detector system based on Convolutional Neural Networks (CCN) used over a Raspberry Pi 4 computer.

Keywords: Convolutional neural networks (CNN), embedded systems, face mask detector (FMD), real time face mask detector (RTFMD), raspberry Pi 4.

1 Introduction

In the Autonomous University of the State of Quintana Roo (UQRoo) at Cancun, the use face masks were a then mandatory requirement to protect the staff and the student community from COVID-19, and to keep a reasonable distance about 1.5 metres each one including the access to the campus facilities through sanitary filters [1].

In most of most public Mexican universities, there is no a usual monitoring of the entrance facilities using a face mask detection system. In the Networking Laboratory, there is an opportunity of deploying an embedded-system prototype to detect fase masks on faces through a comparison of two face detection models based on Convolutional Neural Networks (CNN) which are analyzed in terms of metrics to choose the metric with the best performance. Once, the best choice CNN-based model is implemented on a Raspberry Pi 4 computer.

Conceptually, Artificial Intelligence (AI) consists of systemised methods that mimic the human intelligence to perform tasks and to interactively improve upon the gathered information. AI consists of a variety of fields from bionic robotics, as well as video games, its logic and data processing speed mainly achieved through artificial neural networks [2].

On the other hand, Computational Intelligence (CI) comprises the extraction, learning, reasoning, and training methods to build new knowledge. Within CI, there are two subfields which are Deep Learning (DL) and Machine Learning (ML) that supports the statistical methods to allow machines to improve experiences [3].

On the other hand, ML automatically seeks to learn meaningful relationships and patterns from examples and observations. Advances in ML have enabled the emergence of intelligent systems with human like cognitive capabilities. The ability of such systems to solve problems is based on analytical models that generate predictions, rules, answers, recommendations or similar results [4]. The following six stages summaries the process for building a ML model:

1. Collecting data: Data can be collected from sources such as a website, using either an Application Programming Interface (API) or a database. It´s important to notice that this stage is highly time consuming because data feed the chosen ML model.

2. Data Preprocessing: Once data are available, data need to be processed to ensure the adequate format to feed the ML algorithm of interest. In practice, several pre-processing tasks have to be performed before the proper use of data properly.

3. Data Exploration: This stage performs a preliminary analysis to fix any missing values cases or to try to find, at first sight, any pattern on data to make easier the model construction. At this point, outliers should be detected; or find the most influential characteristics to make a prediction.

4. Algorithm Training: Processed data feed ML algorithms with data previously processed. The aim is to extract the useful information from the initial data and then to make predictions properly.

5. Algorithm Evaluation: The evaluation of an ML algorithm generates information which tests over the information generated by the knowledge of the previous training obtained through previous interactions.

6. Algorithm Implementation: Then, the ML algorithm is finally implemented by a programe on either a Personal Computer (PC) or an embedded system.

Summarizing these stages, we find a learning paradigm, based on supervised learning that means the algorithm is taught how to perform a job, having a classified dataset under a certain appreciation or idea to find patterns that can be applied in an analysis, and produce an output that is already known. In next section, we present some key required preliminaries about Convolutional Neural Networks(CNN) to deploy this proposed present work.

2 Face Detection Systems Preliminaries

2.1. Face Detection Systems

Face detection refers to the ability to identify the presence of human faces on digital images, using ML algorithms to determine whether one or more faces without regarding "whose face is whose" to only count the number of people on the analyzed image, then these analyzed images are stored into a searchable database. Moreover, face detection algorithms often start locating human eyes, which constitute what is known as a valley region and therefore one of the easiest features to be detected. Once the eyes are detected, the algorithm might attempt to detect facial regions, including eyebrows, mouth, nose, nostrils and iris. Once the algorithm assumes that has detected a facial region. Then the algorithm, can apply additional tests to validate whether it has detected a face [6].

2.1.1. Object Detection Architectures Based on CNN Face Detection Algorithms

As CNNs are deep learning algorithms to analyze and learn visual features from large amounts of data [7]. Object detection allows computer systems to "see" their environments by detecting objects on images or visual videos. Through the use of supervised learning as seen from ML algorithms [8].

Supervised learning is when we train a ML algorithm by giving it questions (features) and answers (labels). The algorithm can make a prediction knowing the features. In this type of learning there are two training algorithms: classification and regression [8].

2.2. A Face Detection Model Using CNNs

The structure of CNNs is inspired by neurons in the human and animal brain. CNNs consists of numerous convolutional layers preceding the subsampling (clustering) layers, while the final layers are Fully-Connected (FC) [5].

Object detection allows computer systems to "see" within their views by detecting objects on either images or visual videos. This is made through the use of supervised learning as seen from ML algorithms [8].

As usual, supervised learning is when a ML algorithm is trained by avoiding the questions (features) and answers (labels). Then, the algorithm can make a prediction by knowingthe keyimage features. In this type of learning, there are two algorithms (training): Classification and regression. In [8], classification techniques expect the algorithm to say to which group under study the element of interest belongs, Sandoval mentions that the regression algorithm is a method on which a number is expected. In context, this number does not have a place in a group, but returns a specific value.

2.2.1. CNN Model Training

The classification of a supervised learning CNN-based model is performed after training the model to classify the trained images into their respective classes by

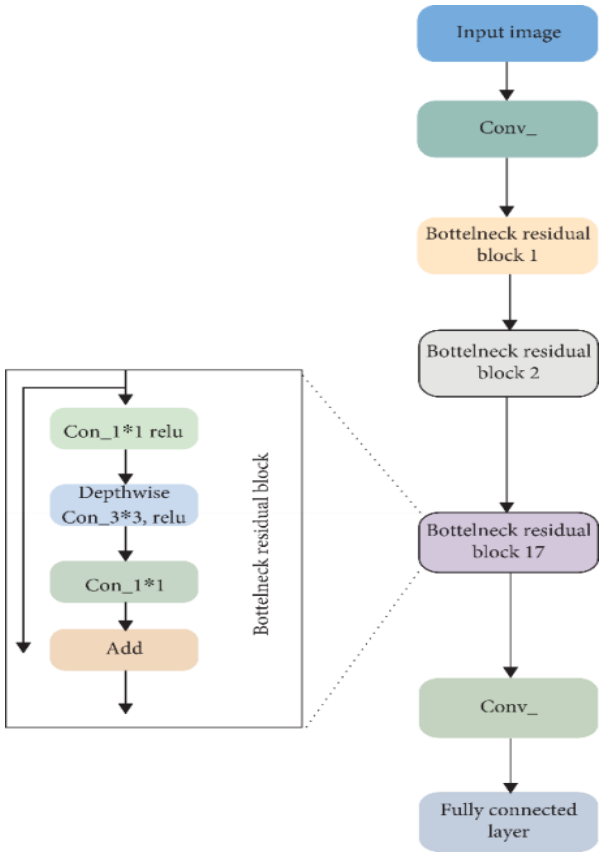

Fig. 1. MobileNet V2 architecture, reference [13-16].

learning important visual patterns. Typically, Tensorflow and Keras are the most commonly used Python-based libraries to be deployed on special computers such as embedded systems [10].

2.2.2. CNN Model for Face/Mask Detection

When the CNN-based model is trained, the model can be implemented so that any image detects the presence of a face mask as the object of interest. The gathered image is initially sent to the face detection model to detect all faces within the environment image. Then, these faces are passed as inputs to the face mask detection model. Next, the model would extract hidden patterns/features of interest from the image and finally classify them as "face masks" or "No face masks" [10].

Based on these preliminaries, we present a brief description of the architecture MobileNet V2 and used metrics, deployed on this proposed embedded system towards its definite deployment.

3 MobileNet V2 Architecture and Metrics

3.1. MobileNetV1/V2

To develop this embedded system, we choose MobilNets from Google, a CNN-based model whose core architecture is based on Depthwise Separable Convolutions including a width multiplier. For more details about MobilNets, see [12-16].

In addition, MobilNet V1/V2 are designed for mobile devices to support classification, detection. The ability to run deep networks enhances the user experience, in terms of accessibility anytime and anywhere as well as security, privacy, and energy consumption saving facilities [9].

While, MobileNetV2 is an enhancement of MobileNetV1 and serves as the state of the art for mobile visual recognition, including classification, object detection and semantic segmentation. MobileNetV2 is released as part of the Tensor Flow-Slim image classification library [9].

Moreover, we select MobileNet V2 because this architecture deals with linear bottlenecks including inverted residuals, for more details about the MobileNet V2 architecture are presented in [16-18]. In Figure 1, we present the diagram of the MobileNet V2 architecture.

3.2. Metrics of Performance

For performance of classification models, we only consider the following metrics such as precision is a non-negative prediction number that indicates how correct the system is, as seen in Equation (1), recall, or sensitivity, implies how many confident instances the model identifies in Equation (2), F1 score represents the mean of recall and precision, yielded in Equation (3) including the accuracy which highlights the usefulness and reliability of the used method as presented in Equation (4).

They are presented from Equations (1) to (4) respectively:

$$Precision = \frac{TP}{FP+TP}, \tag{1}$$

$$Recall = \frac{TP}{FN+TP}, \tag{2}$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall}, \tag{3}$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}. \tag{4}$$

where FN, FP, and TP represent the False Negative, False Positive, and True Positive samples respectively, from an internal confusion matrix as originally presented in [13-15].

For this embedded system, we make use of the of the Face Mask Detector (FMD) method and the Real Time Face Mask Detector (RTFMD) method besides a performance comparative of metrics in terms of the previously mentioned metrics.

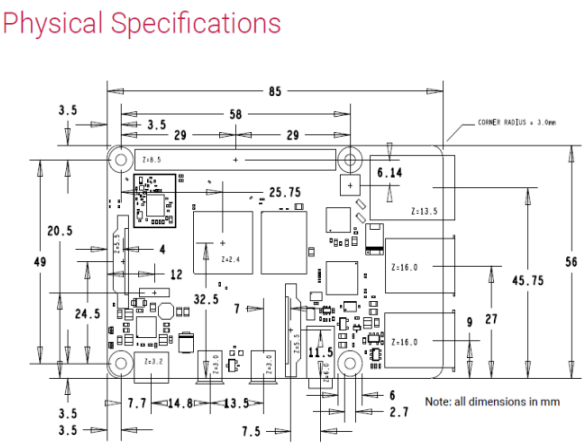

Fig. 2. Diagram of the physical specifications of the **Raspberry Pi 4** computer.

Fig. 3. The Raspberry Pi 4 model B computer.

4 Implementation and Technical Challenges

4.1. The Raspberry Pi 4 Computer Platform

For the implementation of this proposed embedded system, we require a platform to deploy the face detection models using the FDM metrics for CCNs. As the first technical challenge, we choose a portable computer which can load an mobile Operating System (OS) and we decide to use an open-source OS, a community developed one, such that the Raspbian OS, a light version based on Linux Debian OS, as the basis for the installation of the corresponding Python libraries to evaluate the FDM metrics.

As second technical challenge, we selected the Raspberry Pi 4 computer because this type of computer provides the possibility to connect input/output hardware such as monitors including some outstanding built in features such as USB connectivity,

Table 1. Technical specifications of the Raspberry Pi 4 Model B computer, see [11].

Specification	
Processor:	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory:	1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
Connectivity:	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet 2 × USB 3.0 ports 2 × USB 2.0 ports.
GPIO:	Standard 40-pin GPIO header (Fully backwards compatible with previous boards)
Video & sound:	2 × micro-HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port
Multimedia:	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 en-code); OpenGL ES, 3.0 graphics
SD card support:	32 GB Micro SD card slot for loading operating system and data storage
Input power:	5V DC via USB-C connector (minimum 3A) 5V DC via GPIO header (minimum 3A) Power over Ethernet (PoE)–enabled (Requires separate PoE HAT)
Environment:	Operating temperature 0–50ºC
Compliance:	For a full list of local and regional prod-uct approvals, please visit https://www.raspberrypi.org/documeta-tion/hardware/raspberrypi/conform-ity.md
Production lifetime:	The Raspberry Pi 4 Model B will remain in production until at least January 2026

Gigabit Ethernet including Wireless LAN connectivity. In Figure 2, we yield the physical specifications of the Raspberry Pi 4 computer [11].

Based on its physical specifications, this computer was selected because of its credit card size, providing to our embedded system a compact size to provide an easier installation. According to the Raspberry site, the model used in this implementation is the Raspberry Pi 4 Model B. An illustration of the Model B is shown in Figure 3 below.

Next, we present the full technical specifications of the Raspberry Pi 4 Model B computer below. Based in Table 1, the model B was selected because of its specific features, RAM capacity, processor, and SD card support which provides the storage to install the OS to be loaded in the system.

4.2. CNN-Based Face Mask Detector Method

The implementation of CNNs using the FMD-based method starts with the installation of the operating system that is obtained on the Raspberry Pi Imager,

Table 2. Installed software dependencies for the FMD methods [18].

Dependencies	Name of the architecture or version
System architecture	Aarch64
Python	3.9.2
TensorFlow	2.11.0
Keras	2.11.0
Time-Python	0.0.15
OpenCv-Python	3.1.0
Numpy	1.23.5
Picamera2	0.3.6
Imutils	0.5.4

which can be downloadeddirectlyfromtheRaspberry web site. Afterdownloading, theuserexecutes the Raspberry Pi image installation file on the systems in order to install the OS. Then, the OS is ready to be implemented on the 32 GB microSD memory. Then, the Raspberry Pi OS (64-bit) Desktop (compatible with Raspberry Pi 4) is installed.

Once the operating system has been installed on the Raspberry Pi 4 microSD memory and the OS starts to be configured. Once the Graphic User Interface (GUI) started, the installation of the dependencies for the FDM method starts installing on the system.

Installation of the FMD Software Dependencies

Once the FDM is running, the user installs the main software dependencies to operate the FDM method from the terminal of the Raspberry Pi OS (Raspbian). Table 2 presents the required key software dependencies to be installed and configured on our embedded system [18].

4.3. Installation and Activation of the Raspberry Pi 4 Camera

To activate the Raspberry Pi camera, the user types the following command: sudo raspiconfig, on the terminal. Once the camera is activated, the installation process starts takes place. The next configuration windows appear as shown in Figure 4a), by selecting option 5 which shows of the interface options, where the peripherals can be configured.

When the Interfacing Option menu is chosen, the user accesses to either activating or deactivating the menu choices. Next, Figure 4b) shows the first option to activate the Raspberry Pi camera connection.

Once the Interfacing Option is selected. Subsequently 5c) shows the first option to activate the connection to the Raspberry Pi camera. Finally, Figure 5d) indicates the option to reboot the system to save the configuration data, to make effect configuration changes.

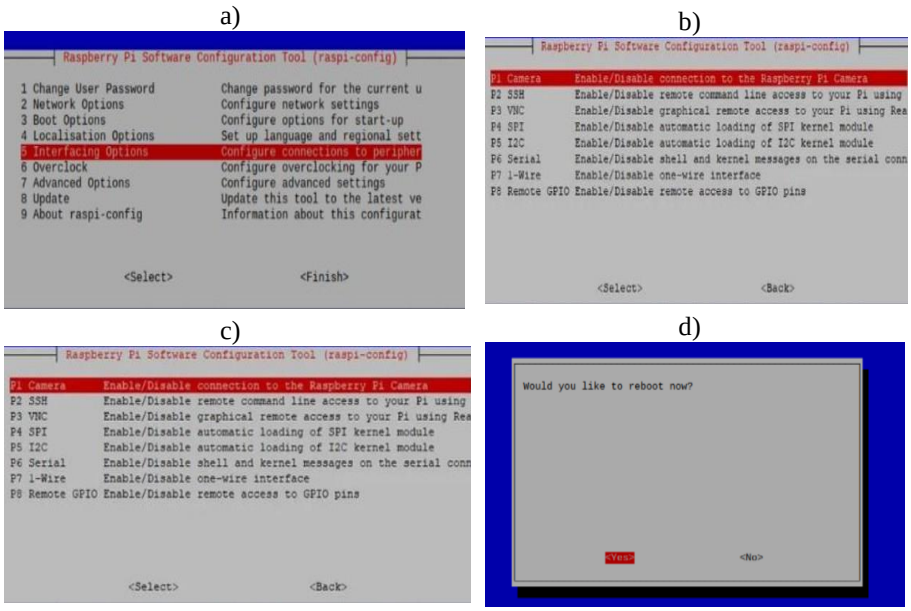

Fig. 4. a) The Raspberry Pi software configuration tool interface. b) The Raspberry Pi Software Configuration Tool menu.c) The Raspberry Pi Software Configuration Tool menu, the user selects the "P1 Camera" option and then choosing the enable option. d) The system reboot window to activate the camera [18].

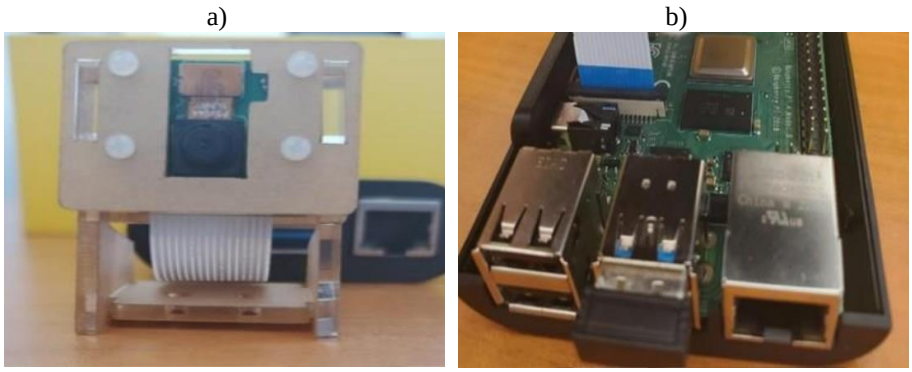

Fig. 5. a) The used Raspberry pi camera. b) The connection between the camera and the Raspberry Pi computer: The camera cable is plugged into the internal port [18].

4.4. Implementation of the Raspberry Pi

Figure 5a) yields the built-up camera for the Raspberry Pi 4 computer while Figure 5b) shows the connection of the flexible cable from the camera to the Raspberry Pi 4 Model B [18].

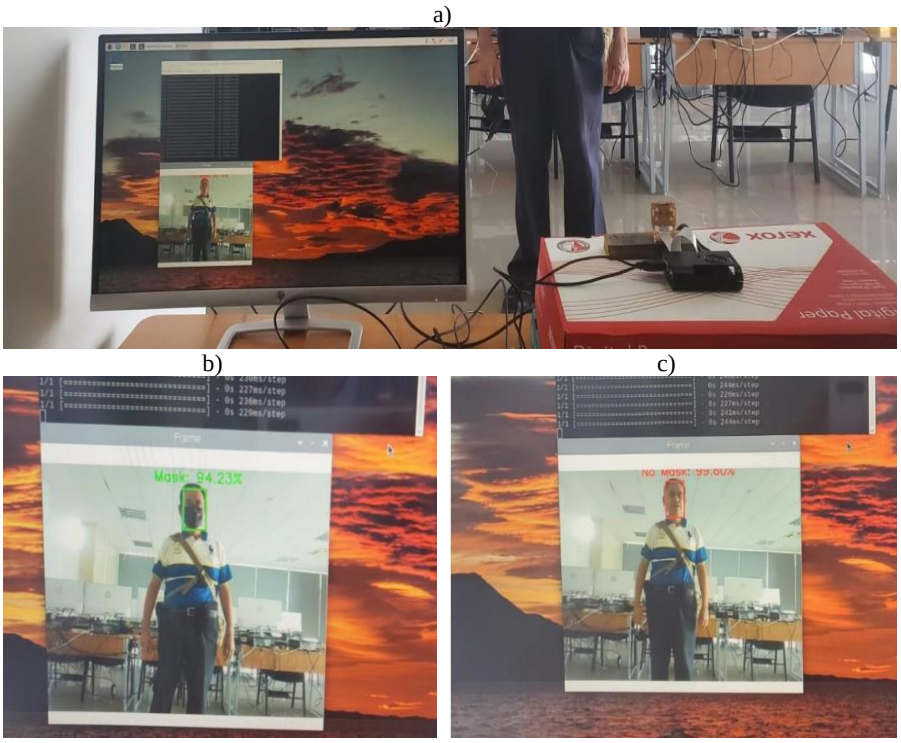

Fig. 6. a) The peripheral connection deployment: monitor with a micro-HDMI cable, a keyboard, a mouse, and a supply cable. b) Face detection using a face mask. c) Face detection without a face mask [18].

As physical implementation, Figure 6a) shows the Raspberry Pi 4 deployment at the Networking Lab. In the same Figure 6a), the monitor shows the output image and a green box appear for wearing a face mask.

In this deployment [18], the user opens the terminal over the directory of the CNN model "Face Mask Detector", made by the following command line; cd face-mask-detector-main. Subsequently, the main Python script is executed to start the face mask detection system, with the following python interface command; python3 detect_mask_video.py

After a few seconds, the camera frame opens to detect whether or not the face mask is present. Figure 6b) presents a zoom-in of the monitor which shows the detection of face mask on face in real time by a green-colored box with the state description "Mask".

Figure 6c) indicates the face detection without a face mask, showing a red-colored box with the description "No Mask". Figures 7b) and 7c) yield how the detection of the use of face masks on real time.

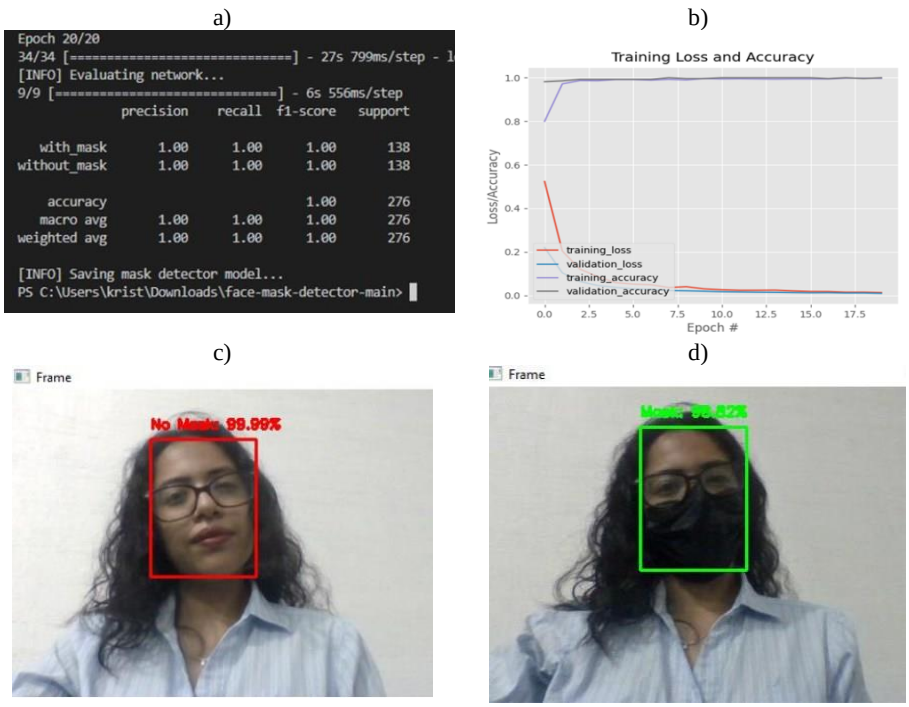

Fig. 7. Metrics of the *Face Mask Detector* **FMD** code. a) Performance metrics b) Graph of losses and accuracy for the **FMD** code. c) Face detection without face mask, **FMD** model. d) Face detection with face mask [18].

5 Results and Discussion

5.1. Results

5.1.1. Method 1: Face Mask Detection (FMD)

Based on the image dataset experiments in [5] which presents both conditions: with mask and without mask, as "with_mask" and "without_mask" respectively. We present the FMD model to obtain the previously mentioned performance metrics: recall, precision, and F1-score. Running the training model, showed in Figure 7, we yield the performance metrics of results [18].

Figure 7a) shows the classification of performance metrics for precision, recall, and F1-score by, sorted by columns, respectively while both with/without face mask conditions including the accuracy are yielded as rows up to 20 epochs with a running time of about 7 seconds. Figure 7b) presents the training/validation accuracy in terms of the number of epochs which tends to the unity from 2.5 epochs while the

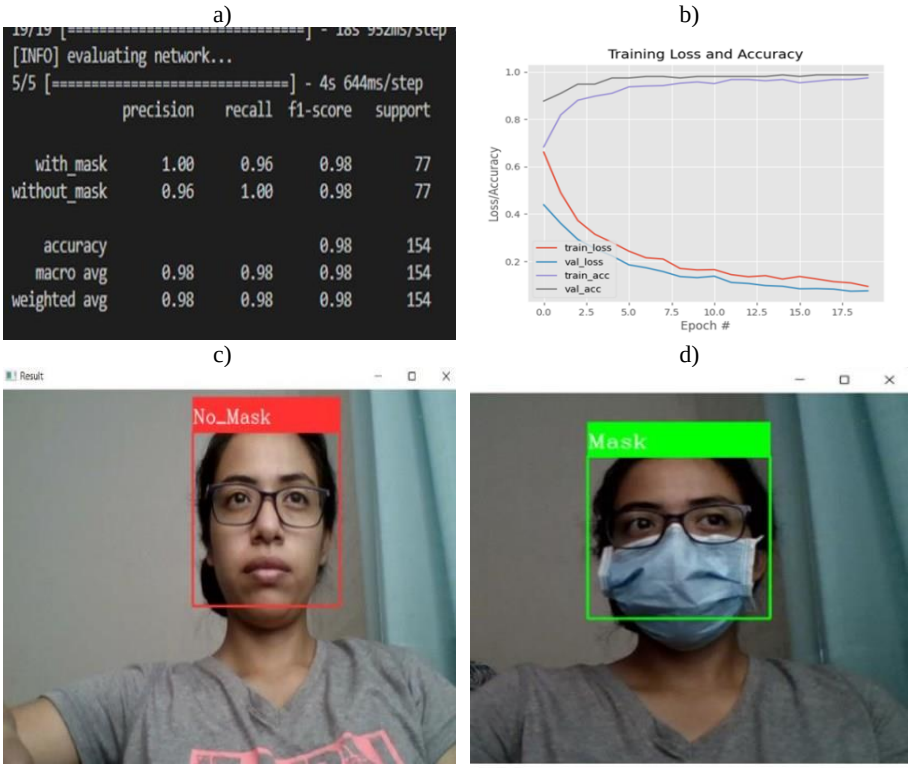

Fig. 8. Metrics of the *Face Mask Detector* **RTFMD** code. a) Performance metrics b) Graph of losses and accuracy for the **RTFMD** code. c) Face detection without face mask, **RTFMD** model. d) Face detection with face mask [18].

training/validation loss has a decreasing behavior. Meanwhile, Figures 1c) and 1d) show the results of the FMD method without and with face masks as detection results: a red frame for the "No Mask" status and a green frame "Mask", respectively [18].

5.1.2. Method 2: Real Time Face Mask Detection (RTFDM) Training

On the other hand, the Real Time Face Mask Detection RTFMD method project does not contain a dataset. However, the dataset was downloaded from [17], and attached to Naemazam's project (2022) [17]. Based on [19], we prepared a dataset having two folders named "with_mask" and "without_mask" image databases. By running the training model, as viewed in Figure 8, presents the measurement of metrics.

In this point, we present the RTFMD method to obtain the previously mentioned performance metrics: recall, precision, and F1-score. Running the training model, showed in Figure 8, we yield the performance metrics of results [18].

Analogously with the previous subsection, Figure 8a) shows the classification of performance metrics for precision, recall, and F1-score by, sorted by columns, respectively while both with/without face mask conditions including the accuracy are yielded as rows up to 20 epochs with a running time of about 5 seconds. Figure 7b)

Table 3. A comparison of metrics [18].

Name of the model	With or without face masks	Precision	Recall	F1-Score
Model 1 FMD	With face masks	1	1	1
	With no face masks	1	1	1
Model 2 RTFMD	With face maks	1	.96	.98
Accuracy = 0.98	With no face masks	.96	1	.98

presents the training/validation accuracy in terms of the number of epochs which carries out behaviors less than the unity while the training/validation loss have a decreasing be-haviour. Meanwhile, Figures 1c) and 1d) show the results of the RTFMD method with-out and with face masks as detection results: a red frame for the "No Mask" status and a green frame "Mask", respectively [18].

5.2. A Comparison of Metrics of the Face Mask Detector (FMD) and the Real Time Face Mask (RTFMD) Detection methods

5.2.1. Discussions

For discussions of results, we make a performance comparison between the FMD and the RTFMD models to choose the model to be deployed on the Raspberry Pi 4 computer. Table 3 yields this comparison in terms of precision, recall, and F1-score performance metrics for both with and without detection of face masks, tested on both academic staff and student members.

In addition, Table 3shows the precision, recall, andF1-scoremetrics for both the FMD and the RTFMD methods which cover performance metrics in a range from 90% to 100% [18].

Based on the results presented in Table 3, section four deals with the implementation of the FMD model on our home-made face detection embedded system using the Raspberry Pi 4 Model B.

Based on performance results, the FDM method was chosen because it showed the best accuracy in detecting face masks closely to the 100% even though the RTFDM method showed an accuracy of about 98%. It is worth mentioning that, although this embedded instrument is under its experimental deployment [18].

6 Conclusions

The implementation of an embedded face detection system using mouth covers based on a Raspberry Pi 4 platform allows for a reliable, programmable, expandable and upgradeable instrument that can implement face detection algorithms based on CNNs.

Then, it is possible to evaluate different face detection modelsover the same instrument to help preventing of infectious diseases not only the covid-19 but also to help implementing protection and prevention measures within a crowded space.

As based on Raspberry Pi 4, we can evaluate not only itsconfiguration and installation of input/output devices managed by a Linux-based OS and implemented in Python-based libraries to be able to perform and evaluate different face detection models.

As a practical evaluation, this implementation of the face cover detection system over a Raspberry Pi 4 computer. However, there is not still a storage method to keep image databases because of the storage limitations of this small computer. Nowadays, there is an ethical concern of image uses but the purpose of this embedded system is only to detect face masks on faces. For deployment places, it is originally implemented in the networking laboratory but this embedded computer would be applied in other parts of campus.

Viewing the RTFDM performance, the accuracy was less effective than the FDM model. However, this embedded system would use different detection models or better. For detection optics, this embedded system does not originally consider camera optics parameters such as lightning, viewing angles which would affect the performance in case of groups of students arriving to the laboratory. For design criteria, this embedded system is conceived to operate indoors instead of outdoor because we assumed lightning conditions as constant. To consider angle views, these views would depend on the camera location.

Although the Raspberry Pi 4 architecture allows connectivity to the Internet, the focus of this work is the comparison of face detection metrics using FDM and RTFDM algorithms based on CNNs using MobileNet V2. The scope of this instrument design is based on a functional embedded system that can be implemented at a very reasonable cost for applications in enclosed, confined and crowded spaces. Finally, we should mention that this face detection system is not currently an IoT device at this moment, but thanks to the versatility of the Raspberry Pi 4 platform, we can add the IoT functionality in future to ensure a cloud storage of images through a local campus server.

References

1. Universidad Autónoma de Quintana Roo: Protocolo y Plan de Seguridad Sanitaria (2020) https://www.uqroo.mx/imagen2020/COVID%2019/PROTOCOLO_DE_SEGURIDAD_SANITARIA-UQROO.pdf.
2. Corporación Universitaria Republicana: Artificial Intelligence at a Glance (2022) http://ojs.ure-publicana.edu.co/index.php/ingenieria/article/view/234/213.
3. Gonzalez, L.: Inteligencia Artificial vs Machine Learning vs Deep Learning. Learn AI (2021) https://aprendeia.com/inteligencia-artificial-vs-machine-learn-ing-vs-deep-learning.
4. Russell, S., Norvig, P.: Artificial Intelligence: A modern approach. Pearson Series In Artificial Intelligence (2021)
5. Alzubaidi, L., Zhang, J., Humaidi, A.J.: Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions, a Survey Paper. Journal of Big Data (2021) doi: 10.1186/s40537-021-00444-8.

6. Diferencia entre face detection y face recognition: Tvc. MX (2020) http://soporte.tvc.mx/Ingenieria/DAHUA/ARCHIVOS_COMUNES/Diferencia%20entre%20Face%20Detecti%C3%B3n%20y%20Face%20Recogniti%C3%B3n%20Dahua.pdf.
7. What Is Computer Vision?: Intel (2023) https://www.intel.com/content/www/us/en/internet-of-things/computer-vision/convolutional-neural-networks.html.
8. Sandoval, L.J.: Machine Learning Algorithms for Data Analysis and Prediction ITCA-FEPADE (2018) http://www.redicces.org.sv/jspui/bitstream/10972/3626/1/Art6_RT2018.pdf.
9. MobileNetV2: The Next Generation of on-Device Computer Vision Networks. Google-blog.com (2020) https://research.google/blog/mobilenetv2-the-next-generation-of-on-device-computer-vision-networks/.
10. Goyal, H., Sidana, K., Singh, C.: A Real Time Face Mask Detection System Using Convolutional Neural Network. Multimedia Tools and Applications, 81(11), pp. 14999–15015 (2022) doi: 10.1007/s11042-022-12166-x.
11. Model, B.: Raspberry Pi 4 Computer (2023) https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf.
12. Howard, A.G., Zhu, M., Chen, B.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017) doi: 10.48550/arXiv.1704.04861.
13. Hussain, D., Ismail, M., Hussain, I.: Face Mask Detection Using Deep Convolutional Neural Network and MobileNetV2-Based Transfer Learning. Wireless Communications and Mobile Computing, pp. 1–10 (2022) doi: 10.1155/2022/1536318.
14. Asghar, M. Z., Albogamy, F.R., Al-Rakhami, M.S.: Facial Mask Detection Using Depthwise Separable Convolutional Neural Network Model During COVID-19 Pandemic. Frontiers in Public Health, 10 (2022) doi: 10.3389/fpubh.2022.855254.
15. Farman, H., Khan, T., Khan, Z.: Real-Time Face Mask Detection to Ensure COVID-19 Precautionary Measures in the Developing Countries. Applied Sciences, (12)8, pp. 3879, (2022) doi: 10.3390/app12083879.
16. Sandler, M., Howard, A., Zhu, M.: MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition, (2018) doi: 10.1109/cvpr.2018.00474.
17. Solano, G.: Data set fase (2021) https://github.com/GabySol/OmesTutorials2021/tree/main/Mascarillas%20 dataset/Dataset_faces/Sin_mascarilla.
18. Flores-Reyes, M.K.: Implementación de un sistema de detección del uso de cubrebocas usando un modelo CNN en una Raspberry Pi para el Laboratorio de Redes en la UAEQRoo Campus Cancún (2022)
19. Azam, N.: Real Time Face Mask Detection (2022) https://github.com/naemazam/Real-Time-Face-Mask-Detection.