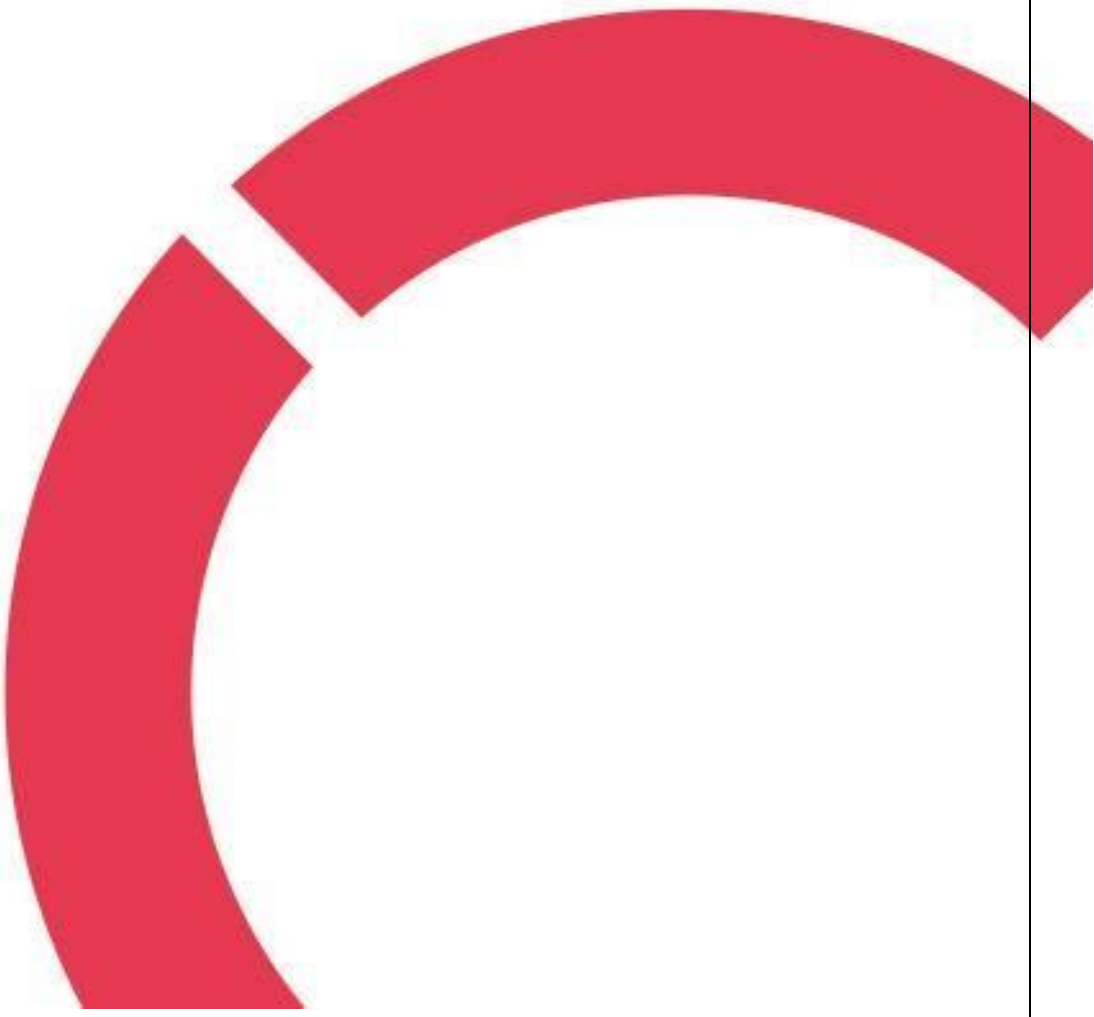
Utada Nikhitha

FACE MASK DETECTION IN REAL-TIME USING PYTHON

Thesis
MLR Institute of Technology
B.Tech
April 2023



ABSTRACT

MLR Institute of Technology	Date April 2023	Author Nikhitha Utada
Degree programme Computer Science and Engineering		
Name of thesis FACE MASK DETECTION IN REAL TIME USING PYTHON		
Centria supervisor M Kalpana		**Pages** 36
Instructor representing commissioning institution or company M kalpana		

The aim of the thesis was to develop a Face Mask Detection system. For face mask identification, the thesis examines the use of Python programming with Deep Learning, TensorFlow, Keras, and OpenCV. The classifier uses the MobileNetV2 architecture as a foundation to do real-time mask detection. This system can be used in real-time applications which require face-mask detection for safety purpose due to the outbreak of coronavirus pandemic.

The system's method is set up in such a way that it uses a video camera to capture people's images and apply detecting algorithms. After the successful implementation of face mask detection with a video camera that helps in the detection of people wearing and not wearing a face mask. Using the visualisation algorithms, it is possible to show the detection percentage of calculation in various ways.

The study is divided into two sections including theoretical and practical sections. The theoretical part of the studies will cover the basics of python programming, deep learning, and convolutional neural network. The practical part will demonstrate how to develop an object detection model for real-time face mask identification using Python programming language and an object detection technique.

Key words
Python, Computer Vision, Neural Networks, Deep Learning, Image, Object detection, Face mask.

CONCEPT DEFINITIONS

List of Abbreviations

AI	Artificial Intelligence
CV	Computer Vision
ML	Machine Learning
OpenCV	Open-Source Computer Vision Library
Accuracy	Percentage of correct classified images
Matrix	Multi-dimensional
Optimizer	A function used to compute gradient
Relu	Rectifiled linear unit
Scalar	Single value
Softmax	Activation function
Test data	Data that the network has not seen
Training data	Data used for training
CNN	Convolutional neural network
OS	Operating System
NP	NumPy
Flatten	A Utility layer
Validation data	Data used for validating algorithm
LB	Labels
Split	Data dividation
DNN	Deep neural network
Locs	Location
PREDS	Prediction
RGB	Red Green Blue

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

FIGURES

1 INTRODUCTION

Object detection is a computer vision approach for detecting and locating objects in images and video. Object detection can detect instances of visual objects of specific classes such as persons, animals, cars or building in digital pictures like photos or video frames. When an image is shown to the eyes, the brain immediately recognizes the objects. On the other hand, a machine needs long time and lot of training data to recognize the images. However, due to the development of recent technology in machine learning, deep learning and the field of computer vision has become a lot easier and more intuitive. Object detection technology has exploded in popularity across a wide range of sectors. It enables self-driving cars in safely navigating traffic, detecting violent behaviour in crowded areas, monitoring object through video surveillance, recognising face mask through object detection module, ensuring adequate quality control of pars in production, among many other things. And this is only the tip of the iceberg in terms of what object detection technology can achieve! In modern video surveillance systems, detecting face in video streams is critical task. Deep learning algorithms have recently been developed that deliver face identifying results.

Artificial intelligence enables image and video-based detection algorithms that can accurately detect an object and determine whether the human is wearing or not wearing a mask. Face mask identification can be done with diverse datasets utilizing deep learning and machine learning approaches such as support vector machines and decision trees. The main purpose of this thesis is to develop a Face mask detection model. The model was trained using the MobileNetV2 architecture. Using two separate image datasets, the model was trained and tested. A pretrained model of the MobileNetV2 architecture was used to distinguish faces from video streams. In addition to the OpenCV framework, a variety of packages of machine learning, deep learning approaches and image processing techniques were used. To provide effective monitoring and enable proactively, the following processes will be used: data augmentation, loading the classifier, establishing the fully connected layer, pre-processing, and loading the picture data, applying the classifier, training phase, validation and testing phase.

2 DEEP LEARNING

Deep learning is a subfield of machine learning in artificial intelligence that works with algorithms that are inspired by the biological structure and function of the brain to help computers with intelligence. A computer model learns to execute categorization tasks directly from images, text, or sound using deep learning. Deep learning models can attain cutting-edge accuracy, sometimes even surpassing human performance. Models are trained utilizing a vast quantity of labeled data and multi-layer neural network architectures. Data science, which covers statistics and predictive modelling, contains deep learning as a component. Deep learning improves the process of gathering, analysing, and interpreting massive amounts of data much faster and easier for data scientists. Deep learning is a vital component of driver-less cars, allowing them to discriminate between a stop sign and a lamppost. Voice control in consumer products such as phones, tablets, tvs, and hands-free speakers is enabled by this feature.

2.1 Artificial neural networks and biological brains

Artificial neural networks have been around since the 1940s, and deep learning emerges from them. Artificial neurons from neural networks, which are interconnected networks of processing units that resemble axons in a biological brain. Dendrites receive input signals from multiple nearby neurons in a biological neuron, which can number in the thousands. These changed signals are subsequently sent to the neuron's cell body, or soma, where they are combined and sent to the axon. The axon will send a signal to the adjacent dendrites of other neurons if the received input signal exceeds a certain threshold. For comparison, Figure 1 shows the structure of a biological neuron. (Pattanayak 2017.)

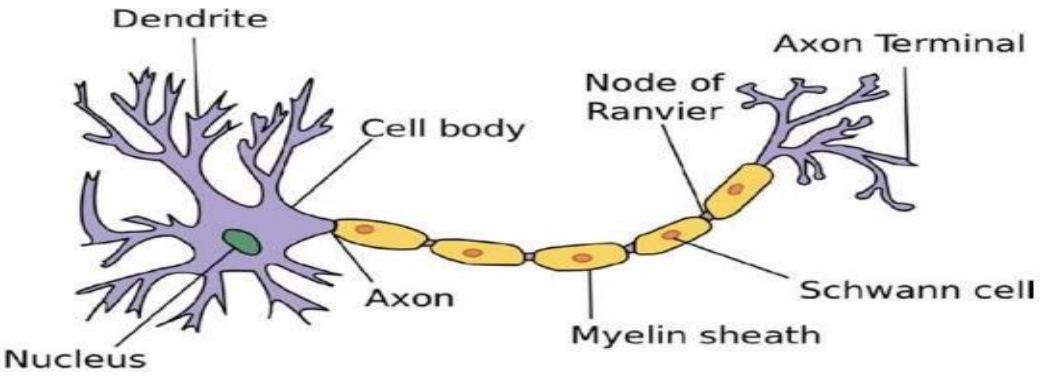


FIGURE 1. Structure of a biological neuron (Pro Deep Learning with TensorFlow 2017.)

Artificial neuron units are based on biological neurons, with a few tweaks for ease of use. The input connection to the neuron, like the dendrites, transmit attenuated or amplified input impulses from neighbouring neurons. The signals are transmitted on to the neuron, which sums up the incoming signals before deciding what to output based on the overall input received. When the entire input crosses a predefined threshold, for example, a binary threshold neuron produces an output value of 1; otherwise, the output remains at 0. Artificial neural networks use a variety of different types of neurons, with the activation function on the entire input to produce the neuron output being the only difference. For simple analogy and comprehension, the different biological analogues are marked in the artificial neuron in Figure 2. (Pattanayak 2017.)

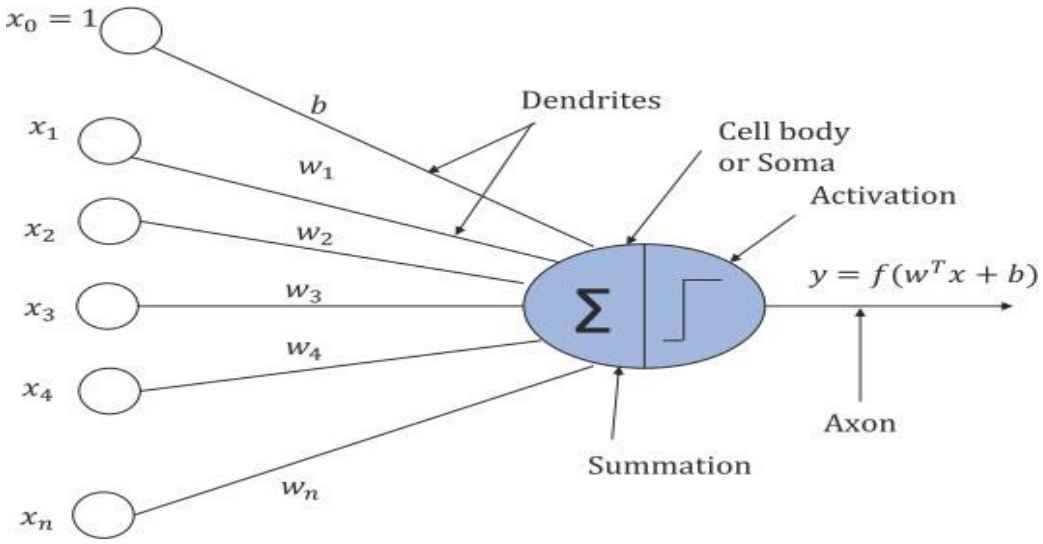


FIGURE 2. Structure of an artificial neuron (Pro Deep Learning with TensorFlow 2017.)

2.2 The process of deep learning

Deep learning models are often referred to as deep neural networks because most deep learning approaches use neural network designs. The number of hidden layers in a neural network is commonly referred to as “deep”. Deep learning refers to the numerous layers that a neural network collects over time, with performance improving as the network becomes more complex. Each layer of the network processes the data it receives in a unique way, which then informs the following layer. Deep neural networks can have up to 150 hidden layers, whereas traditional neural networks only have 2-3. (MathWorks, 2). Deep learning models are taught utilizing enormous quantities of labeled data and neural

network architectures that learn features directly from the data instead of requiring manual feature extraction. The structure of a neural network is shown in Figure 3.

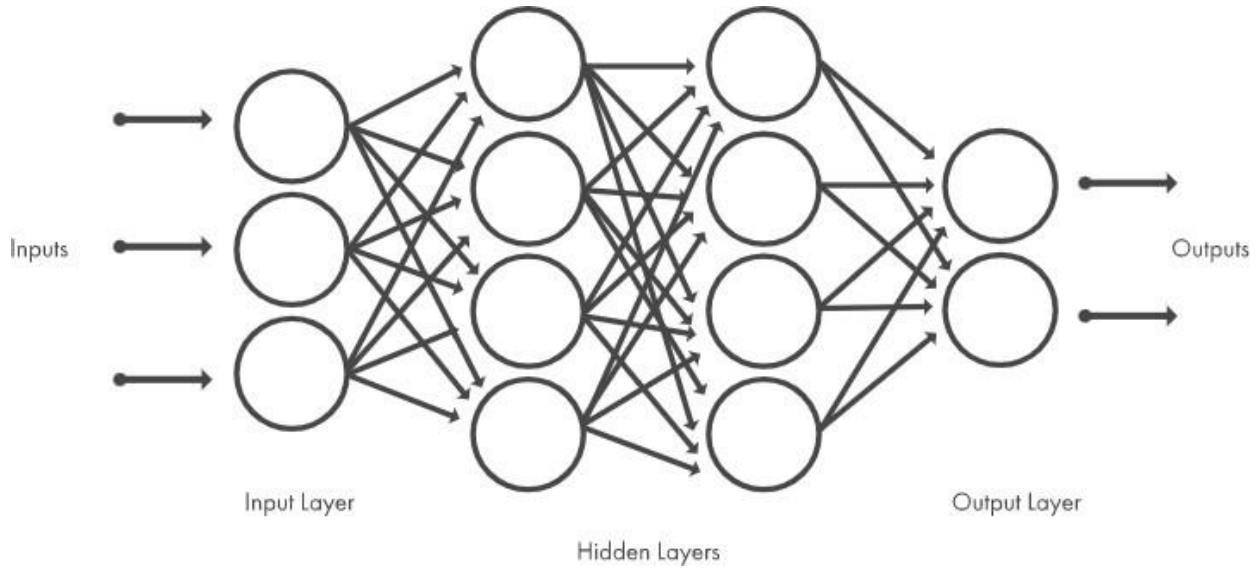


FIGURE 3. Layered neural network are made up of a series of interconnected nodes (MathWorks 2022.)

2.3 Deep learning methods

To create powerful deep learning models, a variety of approaches can be used. The loss of learning rate is one of them. The learning rate is a hyperparameter that determines how much the model changes when the weights are updated in response to the expected error. It is a factor that defines the system or provides operational conditions prior to the learning process. An unstable training process or the acquisition of a faulty set of weights might arise when learning rats are overly high. A slow learning rate could result in a lengthy and ineffective training process. Another strategy is transfer learning, which requires gaining access to a network's internals and fine-tuning a previously trained model. To begin, users add fresh data to an established network that already contains previously unknown classifications. After the network has been updated, new tasks can be completed with more specific classification skills. This method has the advantage of requiring significantly less data that other methods, reducing computation time to minutes or hours. (Burns 2021.)

A developer must collect a big data set and configure a network architecture capable of learning the features and model as part of the training approach. This approach is especially advantageous for new

applications and those with a high variety of output types. However, because it requires a big amount of data and might take days or weeks to train, it is a less used technique in general. The dropout method has been shown to increase neuronal performance on supervised learning tasks in domains including as speech recognition, document categorizations, and computation-al biology. By randomly deleting units and their connections from the neural network during training, this method seeks to alleviate the problem of overfitting in networks with many parameters. (Burns 2021.)

2.4 Examples of deep learning at work

Deep learning applications can be found in a wide range of fields. Automotive experts are employing deep learning to detect elements such as stop signs and traffic signals automatically. In addition, deep learning is used to recognize pedestrians, which helps to reduce accidents. It is used in the aerospace and defence industry to distinguish items from satellites that locate regions of interest and determine whether troops are in safe or dangerous zones. Cancer researchers are using it to detect cancer cells automatically in the medical field. Researchers improved a microscope that generates high-dimensional data that can be used to train a deep learning system to consistently identify cancer cells. By automatically recognizing if people or objects are within a harmful distance of heavy machinery, this technology aids in increasing worker safety near heavy machinery. Deep learning is also used in automatic hearing and voice translation. (MathWorks 2022.)

3 CONVOLUTIONAL NEURAL NETWORK(CNN)

Artificial neural networks are becoming increasingly used for processing unstructured data, such as images, text, audio, and speech. For such unstructured input, convolutional neural networks (CNNs) operate well. Convolutional neural networks find essential features from data when there is a topology connected with it. CNNs are inspired by multi-layer perceptron's in terms of architecture. CNN takes use of local spatial correlation by imposing local connection limitations between neurons in adjacent layers. The processing of data via the convolution operation is the heart of convolutional neural networks. When any signal is convolution with another signal, a third signal is produced that may provide more information about the signal than the original signal. (Pattanayak 2017.)

3.1 CNN architectures

CNN architectures occur in a variety of shapes and sizes, but they all have convolutional and pooling (or subsampling) layers that are organised into modules. These modules are followed by one or more fully connected layers, like a normal feedforward neural network. To create a deep model, modules are frequently stacked on top of one another. The network receives an image directly, which is then processed through various rounds of convolution and pooling. The results of these processes are then fed into one or more fully connected layers. The output layer receives the inputs from the layers above it, executes the calculations using its neurons, and then computes the output. Even though this is the most used base architecture in the literature, various architecture modifications have been proposed in recent years with the goal of boosting image classification accuracy or lowering computation costs. The relationship from input layer to output layer is shown in Figure 4. (Geron 2017.)

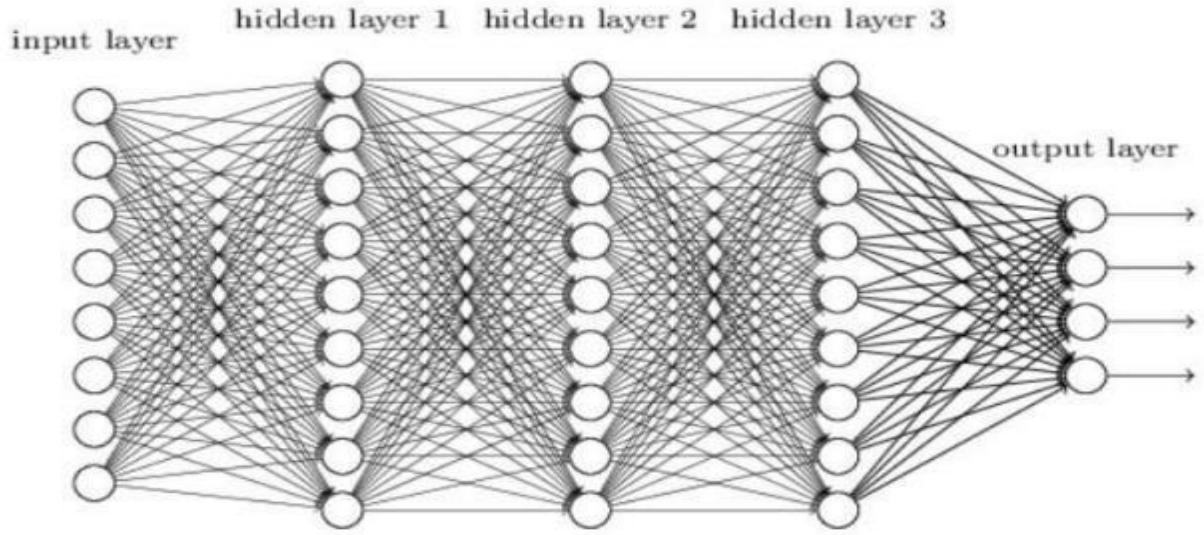


FIGURE 4. CNN architecture (Hands-On Machine Learning with Scikit-Learn and TensorFlow 2017.)

3.2 Workflow of CNN

The higher performance of convolutional neural networks with picture, speech, or audio signals inputs distinguishes them from other neural networks. Convolutional layer, pooling layer, and fully connected layer are the main three basic types of layers available. A convolutional network's first layer is the convolutional layer. While further convolutional layers or pooling layers can be added after convolutional layers, the fully connected layer is the last layer. The CNN becomes more complicated with each layer, detecting larger areas of the image. Earlier layers concentrate on basic elements like colors and borders. As the visual data travels through the cnn layers, it begins to distinguish larger elements or features of the item, eventually identifying the target object. (IBM Cloud Education 2020.)

3.2.1 Convolutional layer

The convolutional layer is the most important component of a CNN because it is where most of the computation takes place. It requires input data, a filter, and a feature map, among other things. For example, a color image made up of a 3D matrix of pixels is sent to the input layer. This means the input will have three dimensions: height, width, and depth, which match to the rgb color space of a picture. A feature detector, also known as a kernel or a filter, will traverse across the image's receptive fields, checking for the presence of the feature. Convolution is the term for this procedure. The feature detector

is a two-dimensional (2-D) weighted array that represents a portion of the image. The filter size, which can vary in size, is usually a 3x3 matrix, which also affects the size of the receptive field. After that, the filter is applied to a portion of the image, and a dot product between the input pixels and the filter then shifts by a stride, and the procedure is repeated until the kernel has swept across the entire image. A feature map, activation map, or convolved feature is the ultimate output of a series of dot products from the input and the filter. (IBM Cloud Education 2020.)

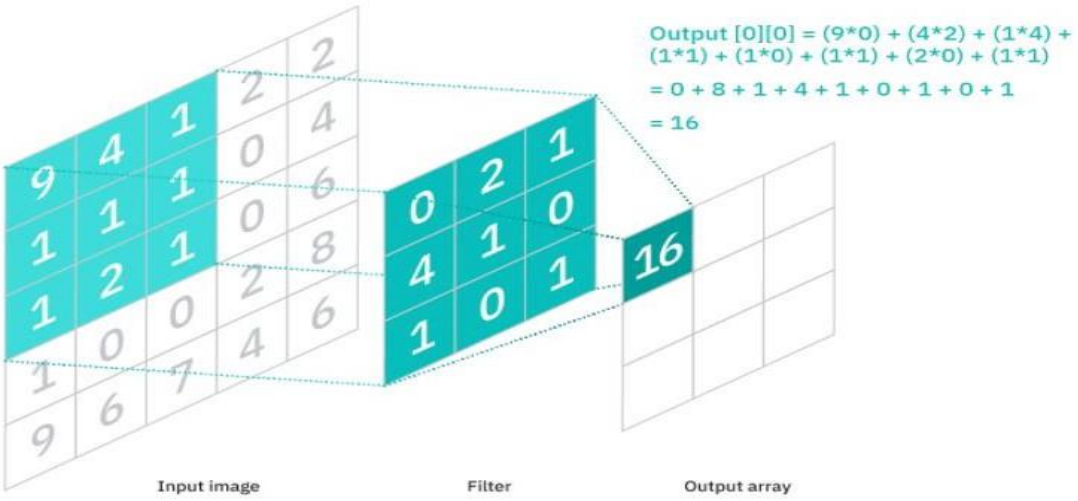


FIGURE 5. Visual representation of a convolutional layers (IBM Cloud Education 2020.)

Each output value in the feature map does not have to relate to each pixel value in the input image, as seen in the Figure 5. It simply must be connected to the receptive field, which is where the filter is applied. Convolutional and pooling layers are often referred to as “partially connected” layers because the output array does not have to map directly to each input value. The feature detector’s weights remain constant as it advances over the image, a technique known as parameter sharing. Through the process of backpropagation and gradient descent, some parameters, such as weight values, adjust during training. Before the neural network can be trained, three hyperparameters that determine the output volume size must be established. Number of filters, stride, and zero padding are among them. The depth of the output is influenced by the number of filters used. The kernel’s stride is the number of pixels it moves over the input matrix. Although stride values of two or more are uncommon, a larger stride produces a smaller output. When the filters do not fit the input image, zero-padding is utilized. All members outside of the input matrix are set to zero, resulting in a larger or similarly sized output. (IBM Cloud Education 2020.)

3.2.2 Pooling layer

Downsampling, often known as pooling layers, is a dimensionality reduction technique that reduces the number of factors in an input. The pooling process, like the convolutional layer, sweeps a filter across the entire input, but this filter has no weights. Instead, the kernel populates the output array by applying an aggregation function to the values in the receptive field. Pooling is divided into two categories: Max pooling and Average pooling. Max Pooling is a convolutional procedure in which the filter or Kernel takes the pixel with the highest value from the input and sends it to the output array. This approach is used more commonly as compared to average pooling. Average pooling, on the other hand, is a pooling technique that employs the average value for patches of a feature map to build a downsampled feature map. (IBM Cloud Education 2020.)

3.2.3 Fully connected layer

The fully connected layer's name is self-explanatory. In a neural network, fully connected layers are those where all the inputs from one layer are connected to each activation unit of the next layer. In partially connected layers, the pixel values of the input image are not directly connected to the output layer, as previously stated. Each node in the output layer is connected directly to a node in the previous layer in the fully connected layer. This layer performs categorization based on the features extracted by the preceding layers and the filters applied to them. While convolutional and pooling layers typically utilize relu functions to classify inputs, fully connected layers typically use a softmax activation function to produce a probability ranging from 0 to 1. (IBM Cloud Education 2020.)

3.3 Different architectures in CNN

CNN is the most well-known and widely used algorithm in deep learning. The fundamental advantage of cnn over its predecessors is that it detects relevant elements without the need for human intervention. CNNs have been widely used in a variety of domains, such as computer vision, audio processing, and facial recognition, among others. CNNs are like traditional neural networks in that their structure is inspired by neurons in human and animals' brains. There are variety of cnn architectures available, all of which have contributed to the development of algorithms that enable AI today and will continue to

do so in the future. This section covers the most well-known cnn architectures. A couple of them are listed below. (Kumar 2022.)

3.3.1 LetNet

The first CNN architecture is LetNet. Yann LeCun, Corina Cortes, and Christopher Burges created it in 1998 to solve problems with handwritten digit recognition. Five convolution layers are followed by two completely connected layers in the model. CNNs were first used in deep learning for computer vision problems with LetNet. However, because to the vanishing gradients problem, LetNet was unable to train effectively. To address this issue, between convolutional layers, a shortcut connection layer known as max-pooling is utilized to minimize the spatial dimension of images, preventing overfitting, and allowing cnn's to train more successfully. Figure 6 shows the architecture of LetNet-5. (Kumar 2022.)

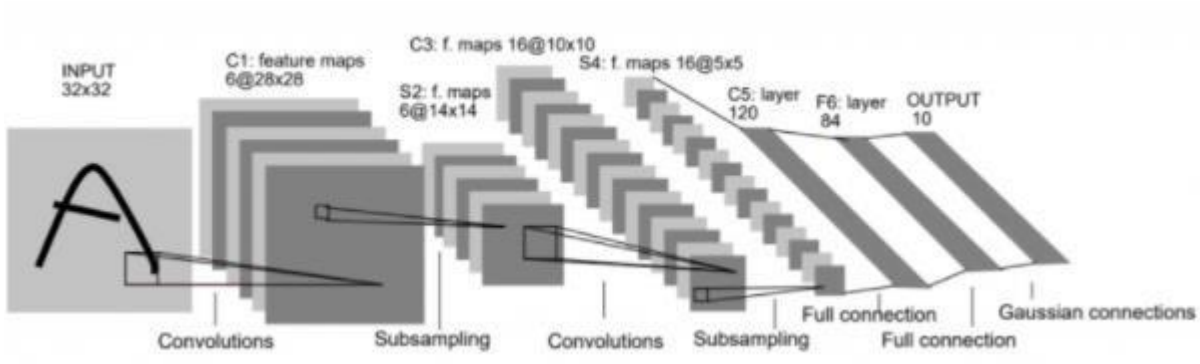


FIGURE 6. LetNet architecture (Vitaflux 2021.)

3.3.2 ZFNet

ZFNet is a cnn architecture that combines fully connected layers with cnn's. Matthew Zeiler and Rob Fergus created ZFNet. It was the winner of the 2013 ILSVRC. Despite having fewer parameters than AlexNet, the network outperforms it in the ILSVRC 2012 classification test, obtaining top accuracy with only 1000 photos per class. It was better than AlexNet because the architectural hyperparameters were tweaked, especially the size of the middle convolutional layers and the stride and filter size on the first layer. The ZFNet cnn architecture has seven layers: convolutional layer, max-pooling layer (downscaling), concatenation layer, convolutional layer with linear activation function, and stride one, a dropout

applied before the completely connected output for regularization purposes. By introducing an approximate inference stage via deconvolutional layers in the middle of cnn's, this cnn model is computationally more efficient than AlexNet. The structure of ZFNet is shown in Figure 7. (Kumar 2022.)

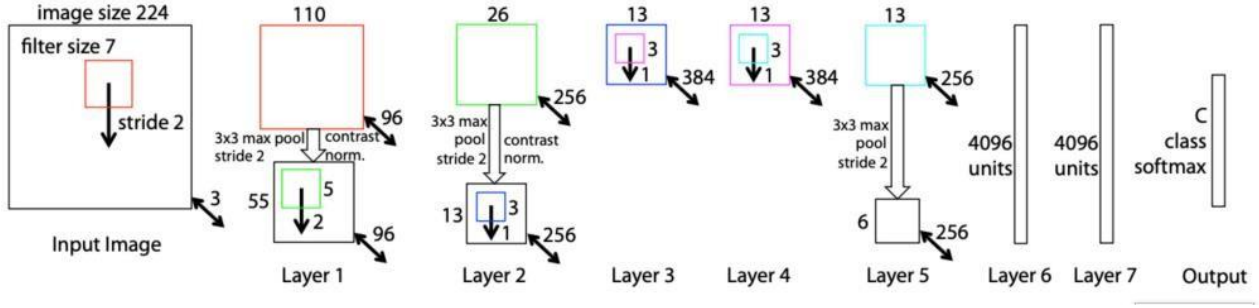


FIGURE 7. ZFNet architecture (Opengenus 2022.)

3.3.3 VGGNet

VGGNet is the cnn architecture developed at Oxford university by Karen Simonyan, Andrew Zisserman, and others. VGGNet is a 16-layer cnn that has been trained on over one billion images and has up to 95 million parameters (1000 classes). It has 4096 convolutional features and can handle 224 by 224 pixel input images. CNNs with such large filters are expensive to train and require a significant amount of data, which is why CNN architectures like GoogleNet (AlexNet architecture) outperform VGGNet for most image classification task with input images ranging from 100 x 100 pixels to 350 x 350 pixels. The ILSVRC 2014 classification challenge, which was also won by GoogleNet cnn architecture, is a real-world application / example of VGGNet cnn architecture. Due to its application on a variety of tasks, including object recognition, the VGG cnn model is computationally economical and serves as a good basis for many applications in computer vision. Its deep feature representation is employed in yolo, ssd, and other neural network architectures. Figure 8 shows a typical VGG16 network architecture. (Kumar 2022.)

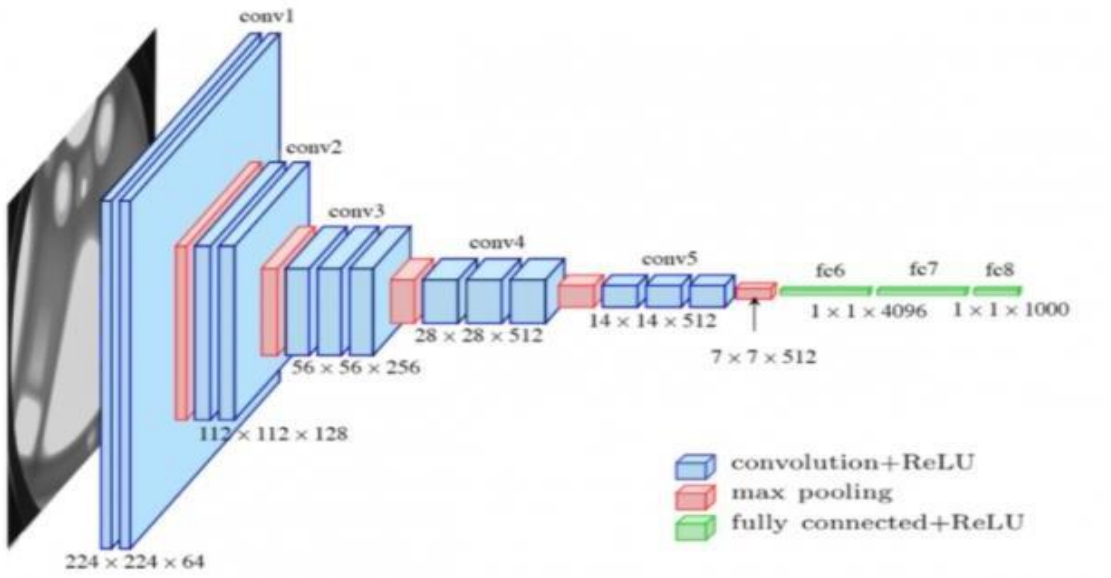


FIGURE 8. VGGNet architecture (Vitaflux 2021.)

3.3.4 MobileNets

MobileNets are a CNN design that is both efficient and portable in real world applications. CNNs that can fit on a mobile device to classify photos or detect objects with low latency are known as MobileNets. They are often very tiny CNN architectures, making them simple to execute in real-time on embedded devices such as smartphones and drones. The design is very adaptable, having been tested on CNNs with 100-300 layers and outperforming other architectures such as VGGNet. CNNs embedded into Android phones to run Google's Mobile Vision API, which can automatically recognize labels of popular objects in photos, are in real-world examples of MobileNets CNN architecture. Figure 9 shows MobileNet architecture. (Kumar 2022.)

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

FIGURE 9. MobileNet architecture (Srudeep PA 2020.)

4 RELATED TECHNOLOGY

Machine learning and deep learning have gained in popularity because of the recent push in the AI industry, and early adopters of this technology are starting to see benefits. Machine learning and deep learning can be implemented using a variety of programming languages, each of which focuses on a different aspect of the problem. Python is the most common language for machine learning and deep learning. For a long time, python has been preferred programming language for machine learning and deep learning researchers. Python provides developers with some of the most flexible and feature-rich tools that improve not only their productivity but also the quality of their code. It is one of the most developer-friendly programming languages, with a diverse set of libraries to serve every use case or projects. (Kumar 2021.)

4.1 Python

Python is a high-level and general-purpose programming language. Python is a programming language that may be used to create desktop gui apps, websites, web applications, mathematics, system scripting and so many. It has an autonomous memory management system and a dynamic type of system. It contains a large comprehensive library and supports different programming paradigms such as object-oriented, imperative, functional, and procedural. Python is predominantly a dynamic typed programming language that was released in 1991. Guido van Rossum invented it and after that the Python Software Foundation developed it. It has simple syntax that lets programmers to express concepts in fewer lines of code. Primarily it was built with code readability in mind. (GreeksforGeeks 2020.)

Python has a lot of advantages as a programming language. Python is easy to learn, which implies that anyone can learn to write code in a relatively short period of time. Python is one of the easiest object-oriented programming languages when compared to java, c, c++, and c#. It is an open-source programming language, which implies that anyone can develop it and contribute to it. Python has an online forum where thousands of programmers come together every day to enhance the language. Python is also free to download and use on any operating system, including windows, mac, and linux. Python has a large range of graphical user interfaces that can be readily integrated into the interpreter, making in one of the

most popular languages among programmers. It understands the concept of class and object encapsulation, allowing applications to be more efficient over time. Python comes with many inbuilt libraries that may be imported at any time and utilized in a specific program right out of the box. (Edureka 2021.)

4.2 TensorFlow

TensorFlow is an open-source machine learning software that focuses on deep neural networks from start to finish. TensorFlow is a collection of libraries, tools, and community resources that are diverse and comprehensive. It enables programmers to construct and deploy cutting-edge machine learning-based applications. The Google Brain team first designed the TensorFlow python deep-learning library for internal usage. The open-source platform's application in R&D and manufacturing systems has increased since then. There are a few key principles in TensorFlow. Tensors are TensorFlow's fundamental building blocks. In the TensorFlow python deep-learning framework, a tensor is an array that represents many forms of data. Unlike a one-dimensional vector or array or a two-dimensional matrix, a tensor can have n dimensions. The shape represents dimensionality. A one-dimensional tensor is a vector; a two-dimensional tensor is matrix; and a zero-dimensional tensor is a scalar. Figure 10 shows various tensor dimensions. (Sharma 2021.)

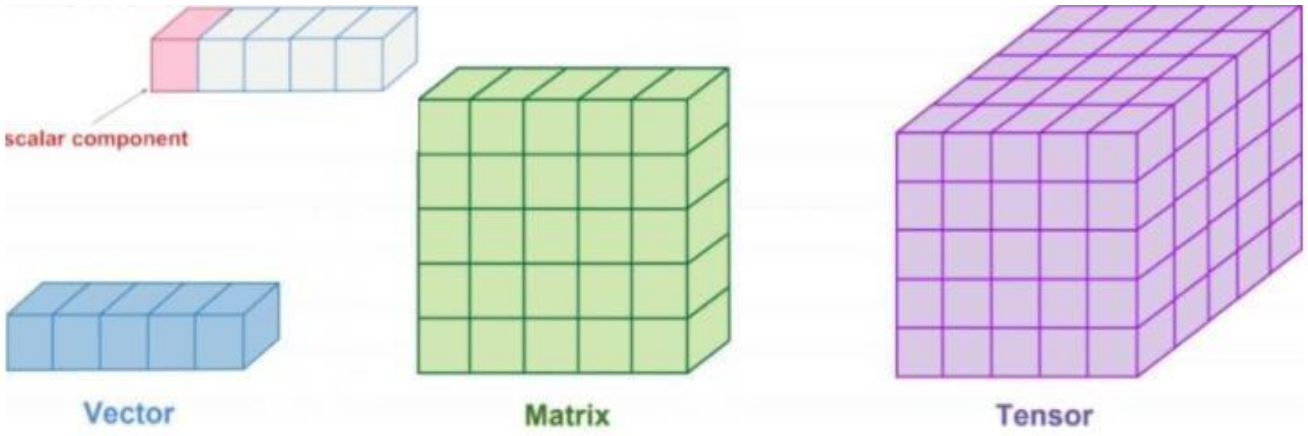


FIGURE 10. Tensor's architecture (Analyticsvidhya 2021.)

4.3 Keras

Google developed Keras, a high-level deep learning API for building neural networks. It is written in python and helps with neural network development. It is modular, quick, and simple to use. It was created by Google developer Francois Chollet. Low-level computation is not handled by Keras. Instead, it makes use of a library known as the "Backend". Keras provides the ability to swap between different back ends. TensorFlow, Theano, PlaidML, MXNet, and CNTK (Microsoft Cognitive Toolkit) are among the frameworks support by Keras. TensorFlow is the only one of these five frameworks that has accepted Keras as its official high-level API. Keras is a deep learning framework that is built on top of TensorFlow and has built-in modules for all neural network computations. Simultaneously, the TensorFlow core API may be used to build custom computations with tensors, computation graphs, sessions, and so on. It gives users complete control and flexibility over their applications, as well as the ability to quickly implement ideas. (Simplilearn 2021.)

4.4 OpenCV-python

OpenCV is a large open-source library for image processing, machine learning, and computer vision. python, c++, java, and other programming languages are among the languages supported by OpenCV. It can recognize objects, faces, and even human writing by analysing efficient library for numerical operations. One of OpenCV's goals is to provide a simple-to-use computer vision infrastructure that allows individuals to quickly create rather complex vision applications. Over 500 functions in the OpenCV library cover a wide range of vision topics, including factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Because computer vision and machine learning are frequently used together, OpenCV also includes a comprehensive machine learning library. (GeeksforGeeks 2021.)

4.5 NumPy and SciPy

NumPy is the most important python package for scientific computing. It is a library that includes a multidimensional array object, derived objects (such as masked arrays and matrices), and a variety of routines for performing fast array operations, such as mathematical, logical, shape manipulation, sorting, selecting, basic linear algebra, basic statistical operations, random simulation, and more. Many other

popular python packages, like as pandas and matplotlib, are compatible with NumPy. (NumPy 2022.). On the other hand, SciPy is another important python library for scientific and technical computing that is free and open source. It is a set of mathematical algorithms and utility functions based on the python numpy extension. It gives the user a lot of power by providing high-level commands and classes for manipulating and displaying data in an interactive python session. SciPy builds on NumPy, developers do not need to import NumPy if SciPy has already been imported. (Great Learning Team 2020.)

4.6 Imutils and Matplotlib

Imutils are a set of convenience functions for OpenCV and python 2.7 and python 3 that make basic image processing functions including translation, rotation, scaling, skeletonization, and presenting matplotlib pictures easier. Matplotlib is an important python visualization library for 2D array plots. Matplotlib is a multi-platform data visualization package based on numpy arrays and intended to operate with scipy stack. It was first introduced in 2022 by John Hunter. One of the most important advantages of visualization is that it provides visual access to large volumes of data in simple images. Matplotlib has a variety of plots such as line, bar, scatter, histogram, and so on. It is a cross-platform package that provides a variety of tools for creating in python from data stored in lists or arrays, and it is one of the most capable libraries available in python. (Imutils & Matplotlib 2022.)

5 IMPLEMENTATION OF PROBLEM

A Face Mask Detection model is the subject of this project. A model has been developed to determine whether people are wearing masks. The primary camera on the computer was used in this system, and the video was sent as input to the model that was deployed. This system is built using OpenCV libraries, as well as images that are supplied into the system during the learning process. The system was identified using the MobileNet algorithm. FIGURE 11 shows the overall system design.

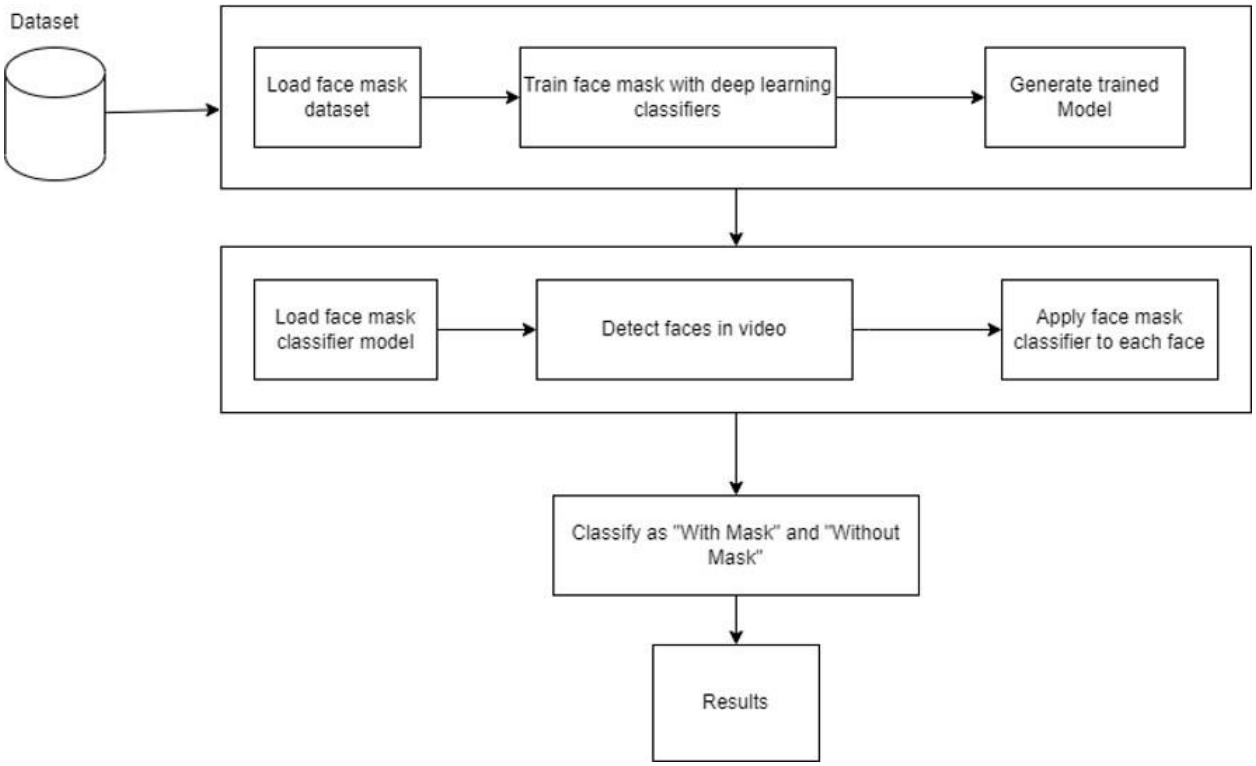


FIGURE 11. System architecture

This project was completed using jupyter notebook. Face mask detection folder was created as a project folder. Inside the jupyter notebook, an environment with a file named main.py was generated. Within the project, two folders have been created like dataset and Face Mask project. In the dataset folder, two Image datasets have been created and are ready to be used for mask training. Two new files have been added to the face mask project: train_mask.py and detect_mask.py. The python programming language was used to create the programs in these two files, which can be seen in FIGURE 12.

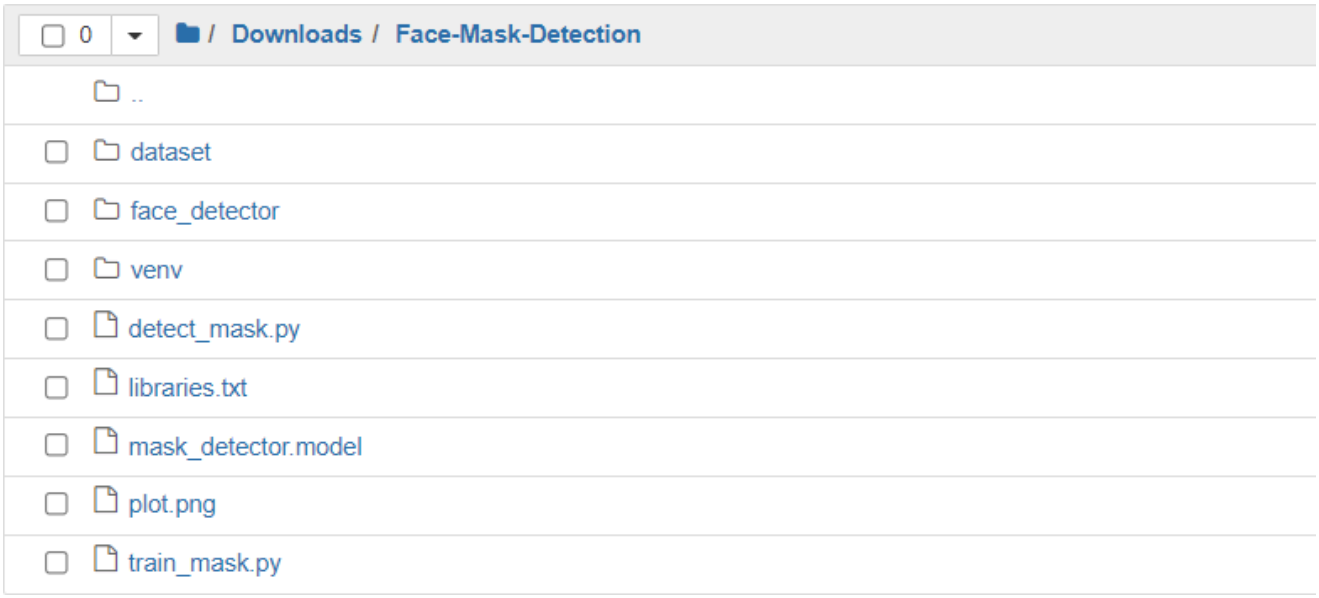


FIGURE 12. Project path

5.1 Preparing dataset

Two image datasets were collected from the kaggle website for the model's training and testing purposes. (Kaggle 2022.) Following that, image dataset was split into two groups: one Images_with_mask and one Images_ without_mask. More images have also been added to the dataset, which were taken with the laptop's default camera. Using these datasets, it is possible to develop a model that can distinguish between people wearing masks and people who are not wearing masks. These datasets contain around 1400 images from the folders "Images_with_mask" and "Images_without_mask". Images with mask and without mask can be seen in FIGURE 13 and 14.

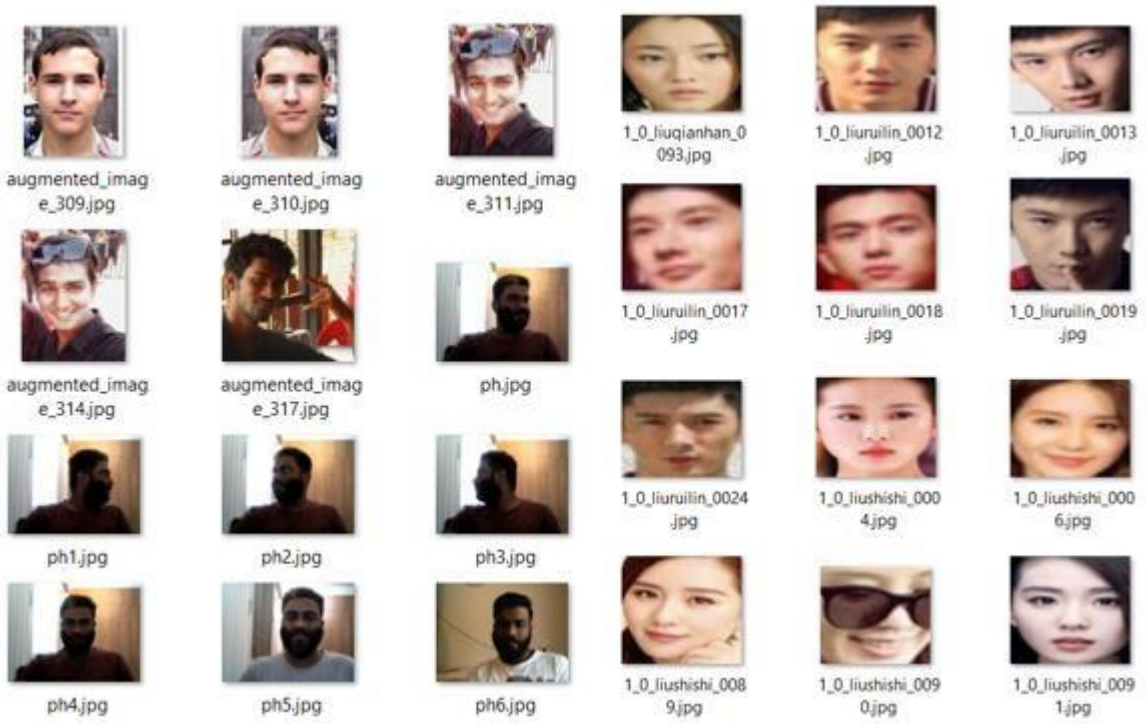


FIGURE 13. Images without mask (Kaggle 2022)

Different types of people's facial images are displayed above the image figure. These images were taken from kaggle's website. The dataset also contained images taken by author using his computer's default camera and added to th dataset. These images were utilized to achieve the purpose of model training.

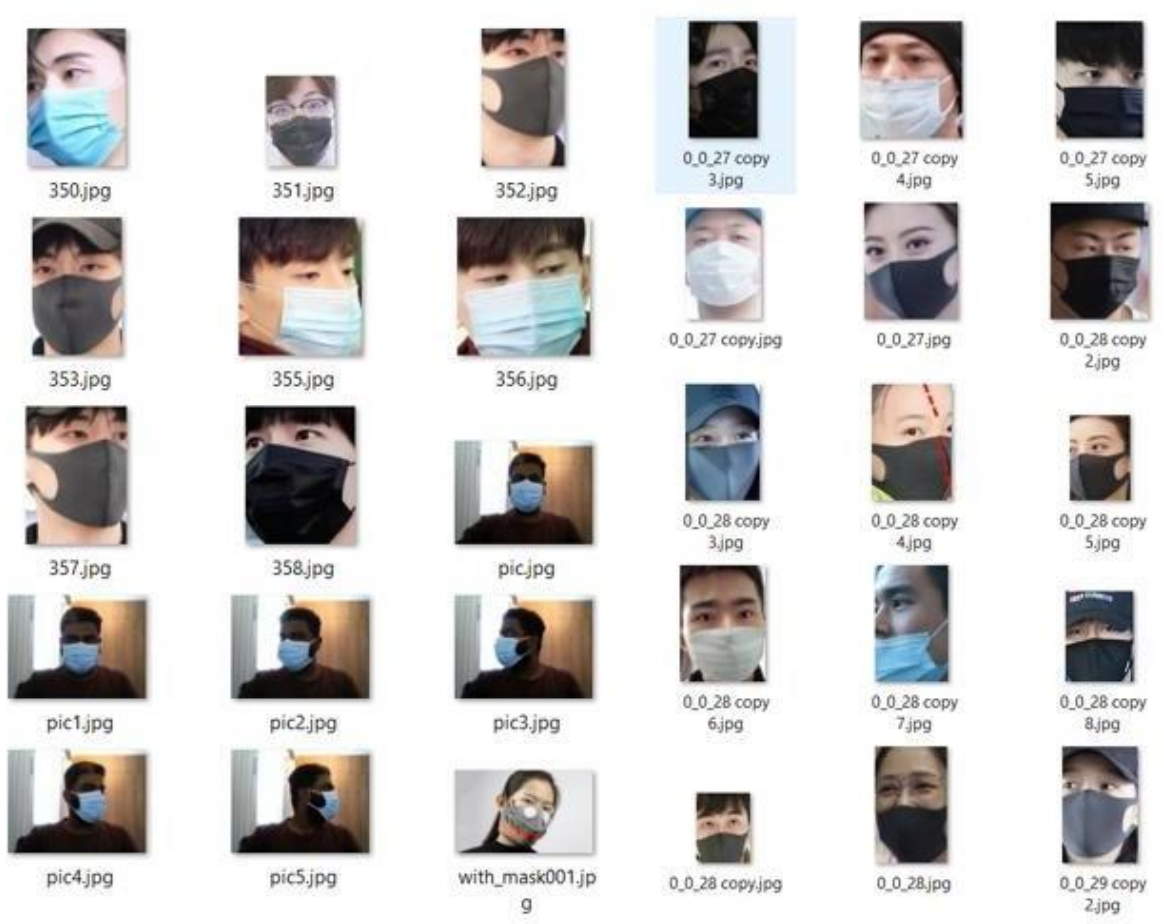


FIGURE 14. Images with mask (kaggle 2022)

The datasets also contain images with mask. These have also been collected from kaggle website. For data training, pictures were taken from different viewpoints and by different people. Author included a couple of his own photos in the image datasets, which he took with the computers default camera.

5.2 Importing necessary libraries

The python programming language, as well as machine learning and deep learning methods were used to identify the face masks. Inside the project path, several required libraries were installed. Libraries were installed using the cmd-command prompt. Packages like tensorflow, numpy, imutils, keras, opencv, scipy, and matplotlib were imported after the installation. These packages take care of their own functions as needed within the application. FIGURE 15 shows all the imported packages.

```python
#Libraries for train_mask.py files.

from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
```

```python
# Necessary libraries for detect_mask.py

from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
import numpy as np
import imutils
import time
import cv2
import os
```

FIGURE 15. Libraries were imported for the train_mask.py and detect_mask.py filles

5.3 Train neural network model

To begin, the libraries needed for the implementation process were imported, including tensorflow, numpy, imutils, keras, opencv, scipy, and matplotlib. Datasets were also imported using the import technique. Model's directory has been created and the path to the dataset folder in the directory has been shown in FIGURE 16. In this area, there is category. Within the category, there are two values: one with mask and the other without. Through these two categories, a loop function has been applied. FIGURE 16 also demonstrates that the program started with the initial learning rate and then switched to a lower learning rate. Because the learning rate has been used less than the loss, the loss has been appropriately estimated. As a result, the level of accuracy was high.

```
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

DIRECTORY = r"C:\Users\karim Suzon\Downloads\Face-Mask-Detection"
CATEGORIES = ["with_mask", "without_mask"]
```

FIGURE 16. Path directory, initial learning rate, epochs, and batch size

There have been two data and labels list generated, which can be seen in FIGURE 17. There were no items on either list. All image arrays were concatenated with the data. All the corresponding images were appended to the labels list whether they have a mask or not. They effectively served as a label for those images, indicating whether they had mask or not.

```
data = []
labels = []
```

FIGURE 17. Data and labels list

After that, the for-loop function was used to loop through the categories. A link to the route with the directory and category has been generated. 'Os.lisdir' produces a list of all the images paths inside the loop. Following that, the Load image function is utilized. The load image is handled by the imported library "keras.preprocessing.image". The image path has been loaded with load image, and the image size target is (224, 224) pixels. The dimensions of an image are its width and height. By loading the images, they were saved into an array. The "img_to_array" method was used to convert the images to an array. The "keras.preprocessing.image" package was used to import the image into the array module. After converting the images to an array, a library called "preprocess_input" function was utilized. This model was created with MobileNet. Following that, all the images were appended to the data list and labelled within the labels list. The "preprocess_input" method was used because of the MobileNet library. When utilizing the MobileNet library, the "preprocess_input" function is necessary. FIGURE 18 shows the entire operation of this part.

```
for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)

        data.append(image)
        labels.append(category)
```

FIGURE 18. For Loop function has applied through categories

For all the data, numerical values were provided. On the other hand, the labels remained alphabet values with mask and without mask. These text characters were transformed to Binary as a result. The "LabelBinarizer" function from the "sklearn.preprocessing" package was used to do this. Then "with_mask" and "without_mask" were used to create categorical variables. The data has been processed and labelled into numpy arrays after being divided into categorial variables, as the deep learning module only works with arrays. The list was converted to an array using the "np.array" methods. NP is the abbreviation for numpy. The labels list has also been transformed into a numpy array, which can be seen in FIGURE 19.

```
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

data = np.array(data, dtype="float32")
labels = np.array(labels)
```

FIGURE 19. Data labelling

Training and testing data were separated into trainX and testY, as well as trainY and testY. Within the classification process, there were two types of datasets: training and testing datasets. The training set was used to train a model, whereas the test set was used to test the model that have been trained. As shown in FIGURE 20, 30% of the testing sets were used, while the remaining 80% was used for additional training. In the labels stratify was utilized and in the random state value was used 42. Because the datasets were separated into training and testing, data preparation was carried out. The

“sklearn.model_selection import train_test_split” package was used to split the data into train and test. MobileNet was utilized for image processing in addition to the convolution neural network. After being processed, the images were sent to MobilNet. Then s fully connected layer was created via max-pooling. As a result, the results appeared.

```
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.30, stratify=labels, random_state=52)
```

FIGURE 20. Data splitting

The “ImageDataGenerator” has been generated, as seen in FIGURE 21. To provide data documentation, the function “ImageDataGenerator” was utilized. By applying numerous attributes to it, such as flipping, shifting, and rotating, it was able to create many different images. As a result, it enabled the creation of more datasets. The values have been included with each. Essentially, image generator was used to create several images from a single image by modifying its attributes.

```
aug = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.15,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.15,
    horizontal_flip=True,
    fill_mode="nearest")
```

FIGURE 21. Image data generator

Using MobileNetV2, a baseModel has been developed. Weights = “imagenet” is a parameter that was used to pre-train an image model. Following that, the boolean value “include_top = False” was mentioned. A fully connected layer was included at the top of the network. The image form as it passed through “input_tensor”. The image size (224, 224) was given there which was specified in the data preprocessing section. Three channels in the image were represented by three values such as 224, 224, 3.

These three channels represented red, blue, and green in rgb colour images. The baseModel is shown in FIGURE 22 along with the parameters that have been set for it.

```
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
```

FIGURE 22. Setting the parameters

Pooling was utilized to construct a completely connected layer after the baseModel was done. Head-Model object was created, and the output was sent to the baseModel as the first parameter, which can be seen in FIGURE 23. A pool was built, and the pool's size is (7,7). The layer was flattened and a dense layer of 128 neurons was applied. Relu was the activation layer, hence relu is the activation function in non-linear use cases. Dropout is just being used to prevent the model from being over-fitted. In final model had two layers because one was wearing mask and the other was not. The softmax activation function also used for the headModel.

```
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

FIGURE 23. Creating headModel

A model function has been developed, as seen in FIGURE 24. The model function has two inputs and outputs parameters. The inputs were in the baseModel, and the outputs were in the headModel. The layers in the base model were initially frozen to prevent them from being changed during the training operation, as shown in FIGURE 24. They were merely a stand-in for convolutional neural networks. They were kept for the purpose of training.

```
78 model = Model(inputs=baseModel.input, outputs=headModel)
79
80 for layer in baseModel.layers:
81     layer.trainable = False
```

FIGURE 24. Model function

The initial learning rate was specified as 1e-4, which was indicated at the start of the code, and thereafter it has changed to decay. The loss function was specified as “binary_crossentropy”. In the compilation adam optimizer was utilized as the optimizer. The adam optimizer was related to Relu, which was the optimizer of choice for any image prediction approach. The accuracy of the function was measured using metrics, and that was the only metric to keep track of. FIGURE 25 shows the explanation for this section.

```
83 print("compiling model...")
84 opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
85 model.compile(loss="binary_crossentropy", optimizer=opt,
86     metrics=["accuracy"])
```

FIGURE 25. Function for learning rate, loss, and accuracy

The model was in the process of being fitted, as shown in FIGURE 26. The data was being tracked to determine if there was a way to gain more image for training data. An image data generator was used for the purpose of large number of datasets. For small datasets, it is always advisable to employ an image data generator. For the data validation testX and testY has been used. The number of epochs passed to the model previously discussed, such as 20 epochs.

```
88 print("[INFO] training head...")
89 H = model.fit(
90     aug.flow(trainX, trainY, batch_size=BS),
91     steps_per_epoch=len(trainX) // BS,
92     validation_data=(testX, testY),
93     validation_steps=len(testX) // BS,
94     epochs=EPOCHS)
```

FIGURE 26. Model fit for training and testing

The “model.predict” method was used to evaluate the model network, which can be seen in FIGURE 27. The label’s index was determined using the highest predicted probability for each image in the testing set. As shown in FIGURE 27, the command “np.argmax” was used to do so. The classification report was formatted well formatted, and the model was saved at the end, as seen in FIGURE 27. FIGURE 27 also shows the accuracy and metrics plotted using Matplotlib. The image was also saved using mat-plotlib. As a result, two files appeared on the desktop. One was a model file, while the other was a matplotlib plotting file called “plot.png”.

```
print("evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

predIdxs = np.argmax(predIdxs, axis=1)

print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))

print("saving mask detector model...")
model.save("mask_detector.model", save_format="h5")

N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")
```

FIGURE 27. The network for evaluating

5.4 Test the model

To run the model, the computer’s command prompt was opened and navigated to the location where the relevant training file was located. The python keyword was typed, followed by the file name train_mask.py, and the enter key was hit to start it. After that, the training phase of this model started. All the images had to be trained, which took a lengthy time. The “plot.png” image of the findings was stored in the project folder after the mask detector model was trained. The accuracy and training loss

were plotted. The accuracy and loss have been precisely conveyed. The model appeared to be in good form. The epochs down were given up to 20 epochs. The model's functioning performance was excellent, as shown in FIGURE 28. The "plot.png" and mask detector filles have been saved on the local desk. Face detection was performed using two filles that were downloaded and saved in the face-detector folder. For the camera operation, OpenCV was used.

FIGURE 28. Visualization through matplotlib

5.5 Mask detection process

The "faceNet" program was the first to be loaded, as seen in FIGURE 29. The goal of utilizing "faceNet" to find the location of face detection pairs file. The file paths were connected and stored in a variable called "weightsPath". The "readNet" method from cv2 and the dnn model has been utilized to use them. The abbreviation dnn stands for deep neural network.

```
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
```

FIGURE 29. faceNet function

The model was loaded with “load_model”, and the camera was loaded with the “VideoStream” method, as shown in FIGURE 30. The load model was used to detect masks. There were two different models for detecting masks and faces. Within the “VideoStream” method, there was a function called src that represented the camera. And src=0 indicated the primary camera start function, which was used to start loading the camera.

```
maskNet = load_model("mask_detector.model")

print("[ starting video stream...")
vs = VideoStream(src=0).start()
```

FIGURE 30. load_model and VideoStream

A while loop was used, and the value was set to true, as shown in FIGURE 31. The read function read each frame by frame, and each frame was nothing more but an image. As a result, any image that follows the sequence correctly and has those frames per second seems to the eyes as a video. Following the reading of the frame, a new frame with a width of 400 pixels was created. There was a faceNet, a maskNet, and a frame accessible. FaceNet was used to predict faces, maskNet was used to recognize masks, and frame was used to display the video that was being loaded from the camera.

```
while True:
    frame = vs.read()
    frame = imutils.resize(frame, width=400)
```

FIGURE 31. Video frame reading through While loop

The frame, faceNet, and maskNet arguments were used to build a detect and predict mask function. After that a normal manipulation was then carried out, which can be seen in FIGURE 32. At the end, the method returned the user’s location and prediction. The x and y coordinates of the precise rectangular around the face were used to determine the location. Only the accuracy of the person wearing the mask was used to make the prediction. As a result, the prediction was that the mask would be worn 90% of the time and not worn 10% of the time.

```
def detect_and_predict_mask(frame, faceNet, maskNet):
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
        (104.0, 177.0, 123.0))

    faceNet.setInput(blob)
    detections = faceNet.forward()
    print(detections.shape)
```

FIGURE 32. Detect and predict mask function defined

The method was called and a tuple containing the location and prediction displayed, as shown in FIGURE 33. The value of the tuple was extracted and split by the X and Y coordinates. A rectangle was drawn within that spot for the mask prediction. The beginnings and endpoints of X and Y were X1, Y1 and X2, Y2. As a result, the first prediction was based on the mask, whereas the second prediction was based on the mask not being present.

```

    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    for (box, pred) in zip(locs, preds):
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred
```

FIGURE 33. Detect and predict mask function has called

For image labelling, a label has been inserted. The label has been successfully categorized in the image with or without mask, as seen in FIGURE 34. It was applied to the rectangle that was generated after the labelling step was completed. RGB has been utilized for colour, which was the colour coding standard on OpenCV2. As a result, the lowest and highest values were 0 and 255, respectively. Green was represented by b=0, g=255, and r=0 in rgb. The image with the mask was represented by the colour green, whereas the image without the mask was resented by the colour red. Red was the highest value for the second pair. After that, a format string was utilized for the label. The proportion of the prediction was displayed. As previously indicated, the mask was predicted to be 90% effective, whereas no mask was predicted to be 10% effective. As a result, the mask with the highest percentage was predicted.

```
        label = "Mask" if mask > withoutMask else "No Mask"
        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

```

FIGURE 34. Labelling of images and format function

The image frame included with text. The font was given a size of 0.45. The X value remained constant, while Y value was reduced by 10 pixels. The text was shown above the image box, with no overlap. On the frame, a rectangle was painted. Each frame started with a rectangle that had the X and Y coordinates as well as the colour code, as seen in FIGURE 35. The rectangle's thickness was 2.

```
        cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
```

FIGURE 35. Algorithms for image frames

The order in which the images were displayed was constructed here, as was the frame presentation. The while loop was finally broken here, as shown in FIGURE 36. A configuration was saved with the letter 'q' for breaking from the loop, and the loop was broken by hitting the q button. All the windows were eventually eliminated, and the video was turned off.

```
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    if key == ord("q"):
        break

cv2.destroyAllWindows()
vs.stop()
```

FIGURE 36. Image Frame, breaking loop, and turn off the windows

5.6 Examine the project

To check the result, the command prompt on the computer was opened and navigated to the exact location where the detect mask video file was located. The program was run by typing "python detect_mask.py" and pressing enter button. Finally, the machine ran a face detection algorithm. In a window named frame, a face has been detected. The detector categorized the face as mask and no mask with

an accuracy score of indicating that the face mask was identified. Face mask detection was performed in real time using the author's as well as his family member's facial images. FIGURE 37 shows a single frame containing three people. MobileNet algorithm was utilized to map facial features from the video. Furthermore, the system identifies the face mask whenever a new face image enters the camera.

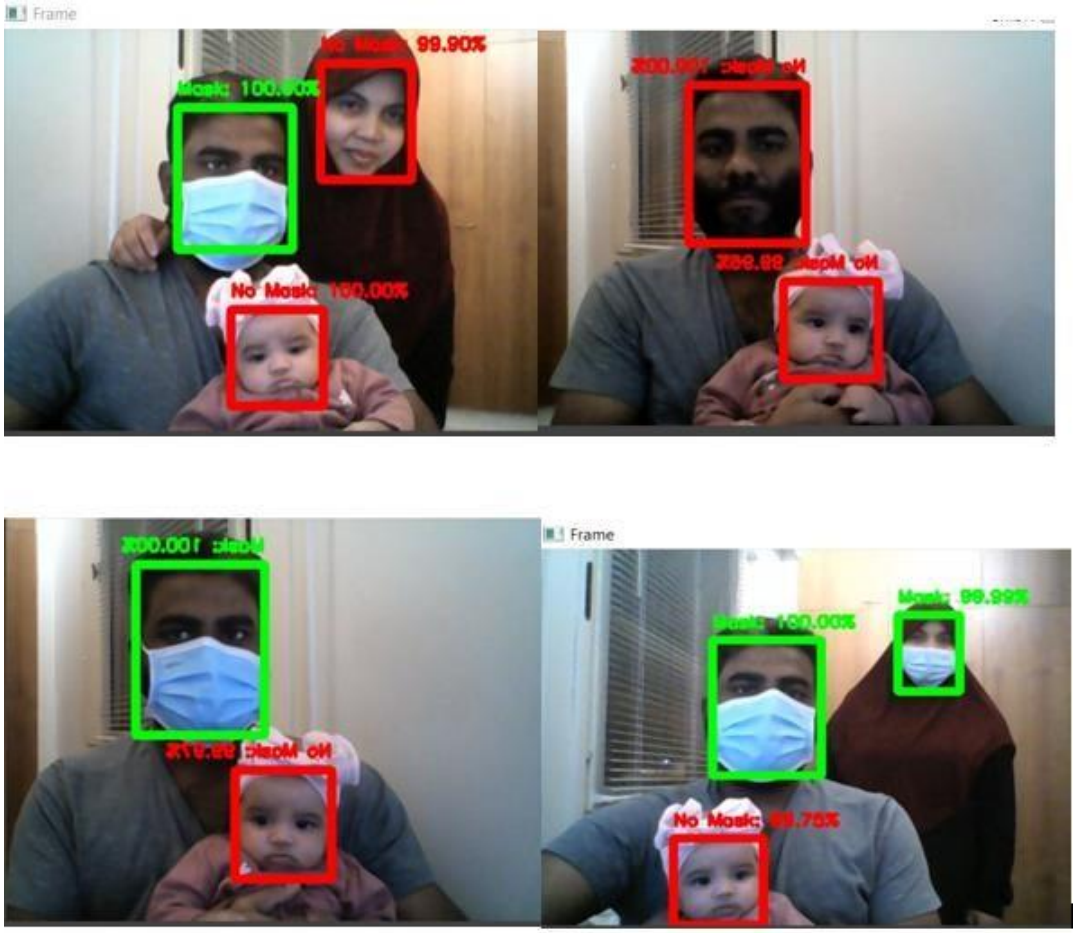


FIGURE 37. System output

6 CONCLUSION

Object detection is a technology that falls under the umbrella of computer vision. Objects in images and videos can be recognized and tracked using this technique. Object identification, also known as object detection, can be used for face recognition, vehicles recognition, self-driving vehicles, security systems, and a range of other applications. Object detection using deep learning and machine learning has become a study focus on recent years. The purpose of this thesis was to create a face mask detector the goal is achieved by putting the concept into practice using cutting-edge technologies like opencv, mobilenet, machine learning, and deep learning. Masks are becoming increasingly trendy these days. In the absence of immunization, masks are one of the few ways to protect against the corona virus, and they play an important role in protecting people's health from respiratory illnesses. To ensure human safety, this project can be integrated with embedded technology and deployed in a range of public venues, such as airports, train stations, offices, schools, and public spaces.

REFERENCES

Burns, E. 2021. *Deep learning*. Available at: What is Deep Learning and How Does It Work? (techtarget.com). Accessed 20 January 2022.

Geron, A. 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. First edition. USA: O'Reilly Media, Inc.

Great Learning Team. 2020. *SciPy Tutorial for Beginners | Overview of SciPy library*. Available at: https://www.mygreatlearning.com/blog/scipy-tutorial/. Accessed 9 March 2022.

IBM Cloud Education. 2020. *What are Convolutional Neural Networks?* Available at: https://www.ibm.com/cloud/learn/convolutional-neural-networks. Accessed 10 February 2022.

Imutils. *Image processing library*. Available at: https://pypi.org/project/imutils/. Accessed 10 March 2022.

Kumar, A. 2022. *Different Types of CNN Architectures Explained*. Available at: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/. Accessed 15 February 2022.

Kumar, S. 2021. *Why Python is Best for AI, ML, and Deep Learning*. Available at: https://www.rtinsights.com/why-python-is-best-for-ai-ml-and-deep-learning/#:~:text=The%20benefits%20of%20making%20Python,the%20popularity%20of%20the%20language. Accessed 28 February 2022.

Kaggle. Available at: https://www.kaggle.com/. Accessed 25 January 2022.

Matplotlib. Available at: https://matplotlib.org/. Accessed 12 March 2022.

MathWorks. *What is Deep Learning? 3 things you need to know.* Available at: https://www.mathworks.com/discovery/deep-learning.html. Accessed 13 January 2022.

MathWorks. *Examples of Deep Learning at Work.* Available at: https://www.mathworks.com/discovery/deep-learning.html. Accessed 22 January 2022.

NumPy. *What is NumPy.* Available at: https://numpy.org/doc/stable/user/whatisnumpy.html. Accessed 6 March 2022.

OpenCV Python Tutorial. 2021. *GeeksforGeeks.* Available at: https://www.geeksforgeeks.org/opencv-python-tutorial/. Accessed 4 March 2022.

Opengenus. 2022. *Evolution of CNN Architectures.* Available at: https://iq.opengenus.org/evolution-of-cnn-architectures/. Accessed 15 March 2022.

Pa, S. 2020. *An Overview on MobileNet.* Available at: https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d. Accessed 17 March 2022.

Pattanayak, S. 2017. *Pro Deep Learning with TensorFlow*. First edition. California: Apress media.

Python Features. 2020. *GeeksforGeeks.* Available at: https://www.geeksforgeeks.org/python-features/. Accessed 20 February 2022.

Python Features. 2020. *GeeksforGeeks.* Available at: https://www.geeksforgeeks.org/python-features/. Accessed 20 February 2022.

Python Features. 2021. *Edureka*. Available at: https://www.edureka.co/blog/python-features/. Accessed 21 February 2022.

Sharma, S. 2021. *TensorFlow for Beginners with Examples and Python Implementation.* Available at: https://www.analyticsvidhya.com/blog/2021/11/tensorflow-for-beginners-with-examples-and-python-implementation/. Accessed 25 February 2022.

Simplilearn. 2021. *The Best Introductory Guide To Keras*. Available at: https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras. Accessed 2 March 2022.