Real Time Face Mask Recognition Using Haar Cascades

[1]Dr. SafiaNaveed.S, Associate Professor, Department of Computer Science and Engineering, KCG College Of Technology, Chennai.

[2] Dr.Nabeena Ameen AP (Sel Gr), Department of Information Technology, B.S. Abdur Rahman Crescent Institute of Science and Technology, Chennai,

[3] SRIRAM V, Student, Department of Computer Science and Engineering, KCG College Of Technology, Chennai.

Abstract: The COVID-19 pandemic, caused by a novel coronavirus, has been continuously spreading all over the world. The impact of COVID-19 has been felt in almost all sectors of development. The healthcare system is going through a crisis. Many precautionary measures have been taken to reduce the spread of this disease, and wearing a mask is one of them. We propose a system that restricts the growth of COVID-19 by finding people who are not wearing any facial masks in a smart city network where all the public places are monitored with Closed-Circuit Television (CCTV) cameras. When a person without a mask is detected, the corresponding authority is informed through the city network. A deep learning architecture is trained on a dataset that consists of images of people with and without masks collected from various sources. The trained architecture distinguished people with and without a facial mask for previously unseen test data. It is hoped that our study will be a useful tool to reduce the spread of this communicable disease in many countries around the world.

The present scenario of COVID-19 demands an efficient face mask detection application. The main goal of the project is to implement this system at the entrances of colleges, airports, hospitals, and offices where the chances of spreading COVID-19 through contagion are relatively higher. Reports indicate that wearing face masks while at work clearly reduces the risk of transmission. It is an object detection and classification problem with two different classes (Mask and Without Mask). A hybrid model using deep and classical machine learning for detecting face masks will be presented. While entering the place, everyone should scan their face and then enter, ensuring they have a mask with them. If anyone is found to be without a face mask, a beep alert will be generated as every office starts to open. Throughout the nation, more COVID-19 instances are continually being reported. It can end if everyone adheres to the safety precautions. Therefore, we expect that this module will assist in spotting masks when people arrive at work.

Keywords: Machine Learning, Deep Learning, Image Processing, Image detection, Haar Cascades.

1. INTRODUCTION

The spread of COVID-19 is increasingly worrying for everyone in the world. This virus can be transmitted from human to human through droplets and airborne.

To reduce the spread of COVID-19, everyone needs to wear a face mask, do social distancing, evade crowd areas, and always maintain their immune systems.

Face Recognition is a technique that matches stored models of each human face in a group of people to identify a person based on certain features of that person's face. Face recognition is a natural method of recognizing and authenticating people. Face recognition is an integral part of people's everyday contact and lives. The security and authentication of an individual are critical in every industry or institution. As a result, there is a great deal of interest in automated face recognition using computers or devices for identity verification around the clock and even remotely in today's world. Face recognition has emerged as one of the most difficult and intriguing problems in pattern recognition and image processing. With the aid of such technology, one can easily detect a person's face by using a dataset of identically matched appearances. The most effective approach for detecting a person's face is to use Python in deep learning. This method is useful in a variety of fields, including the military, defense, schools, colleges, and universities, airlines, banks, online web apps, gaming, and so on. Face masks are now widely used as part of standard virus prevention measures, especially during the COVID-19 virus outbreak. Many individuals or organizations must be able to distinguish whether people are wearing face masks at a given location or time. This data's requirements should be very real-time and automated. The challenging issue that can be mentioned in face detection is inherent diversity in faces such as shape, texture, color, having a beard, moustache, and/or glasses, and even masks. From the experiments, the proposed algorithm is very efficient and accurate in determining the facial recognition and detection of individuals.

The main aim is to detect violations such as not wearing a mask or not following social distancing in a workplace and notify the officials. Technology has advanced tremendously over the past century, from the Internet of Things (IoT) to machine learning and deep learning. CNN is used in various fields like medical, marine science, and many other

2. LITERATURE SURVEY

Mamata et al. (2014) defined face detection as a procedure that has many applications like face tracking, pose estimation, or compression. Face detection is a two-class problem where we must decide if there is a face or not in a

picture. This approach can be seen as a simplified solution to the face recognition problem.

Adaboost is adaptive in the sense that subsequent classifiers are tweaked in favor of instances misclassified by previous classifiers. Adaboost generates and calls a new weak classifier in each of a series of rounds from a set of training images. This method can be used for both face detection and face location. In this method, a standard face (such as frontal) can be used. The advantages of this method are that it is very simple to implement the algorithm and that it is easy to determine the face locations such as nose, eyes, mouth, etc. based on the correlation values.

The mask-face detection model is based on computer vision and deep learning. The model is an integration between deep learning and classical machine learning techniques with OpenCV, tensor flow, and Keira. We have used deep transfer learning for feature extraction and combined it with three classical machine learning algorithms. We introduced a comparison between them to find the most suitable algorithm that achieved the highest accuracy and consumed the least time in the process of training and detection.

The proposed system focuses on how to identify the person on image/video stream wearing face mask with the help of computer vision and deep learning algorithm

1. Train Deep learning model (MobileNetV2).
2. Apply mask detector over images / live video stream.

Most of the images were augmented by OpenCV. The set of images were already labeled mask and no mask.

The images that were present were of different sizes and resolutions, probably extracted from different sources or from machines (cameras) of different resolutions.

Yassin Kortli et al. (2020) suggested that developing biometric applications, such as facial recognition, has recently become important in smart cities. Besides, many scientists and engineers around the world have focused on establishing increasingly robust and accurate algorithms and methods for these types of systems and their application in everyday life.

These biometric factors make it possible to identify people's identities based on their physiological or behavioral characteristics. They also provide several advantages, for example, the presence of a person in front of the sensor is sufficient, and there is no need to remember several passwords or confidential codes anymore.

Face Recognition

Three basic steps are used to develop a robust face recognition system:

The face recognition system begins with the localization of human faces in a particular image. The purpose of this step is to determine if the input image contains human faces or not. The variations in illumination and facial expression can prevent proper face detection. To facilitate the design of a face recognition system and make it more robust, pre-processing steps are performed. Many techniques are used to detect and locate the human face image, for example, the Viola-Jones detector, the histogram of oriented gradient (HOG), and principal component analysis (PCA). Also, the face detection step can be used for video and image classification, object detection, region-of-interest detection, and so on.

Optimization Techniques

Naveed et al. [7] suggested few optimization techniques such as Particle Swarm Optimization (PSO), Discrete Particle Swarm Optimization (DPSO) and Fractional Order Discrete Particle Swarm Optimization (FODPSO) Techniques based on which the best or optimum features from the face of the chauffeur can be selected to denote his drowsiness.

Naveed et al. [8] also suggested that the region of interest can be captured and the same region can be inspected thoroughly in terms of pixel values, evaluating the degree of noise present, analyzing the position of boundaries to study the region of interest, analyze the fluctuating intensities across the forehead region of the face and consider the edge effects of the eyes

Face Recognition

This step considers the features extracted from the background during the feature extraction step and compares them with known faces stored in a specific database. There are two general applications of face recognition: one is called identification, and another is called verification. During the identification step, a test face is compared with a set of faces, aiming to find the most likely match. During the identity step, a check face is as compared with a recognized face withinside the database to be able to make the popularity or rejection decision.Correlation filters (CFs), convolutional neural networks (CNN), and k-nearest neighbor (K-NN) are known to effectively address this task.

UdayTheja et al. (2020) discussed Corona virus disease (COVID-19), an airborne infectious disease caused by a newly discovered Corona virus. The best way to prevent or slow down the transmissions is to have knowledge of the COVID-19 virus, the disease it can cause, and how it passes. There are many steps suggested by the WHO (World Health Organization) to prevent the spread. One of which is wearing medical masks, which is highly desirable even after the lockdown period until a vaccine or medicine is invented.

This system aims at classifying whether a person is wearing a mask or not by taking input from Images and real-time streaming Videos.

This system aims at classifying whether a person is wearing a mask or not by taking input from Images and Real time streaming Videos.

The classification of the images is done by training the model in 2 phases:

Phase 1: Face mask dataset is loaded into the system. Different classifiers like MobileNetV2, ResNet50, and VGG16 are used to generate a trained model.

Phase2: Load the face mask classifier model.

Detect faces in the images/video stream. Apply the classifier to each face Roi. Classify the images to be With Mask and Without Mask with Confidence.

This system may then be interfaced with

Case 1: Existing access control system so that violators can be restricted.

Case 2: There could be some scenarios in work places where people may forget or just put off the mask when it becomes uneasy for them to get accustomed to the new face masks.

3. METHODOLOGY

3.1 BlockDiagram

To predict whether a person has worn a mask correctly, the initial stage would be to train the model using a proper dataset. After training the classifier, an accurate face detection model is required to detect faces. The model can classify whether the person is wearing a mask or not.

The task here is to raise the accuracy of mask detection without being too resource-heavy. This approach helps in detecting faces in real-time, even on embedded devices like the Raspberry Pi. The following classifier uses a pre-trained model MobileNetV2 to predict whether the person is wearing a mask or not.

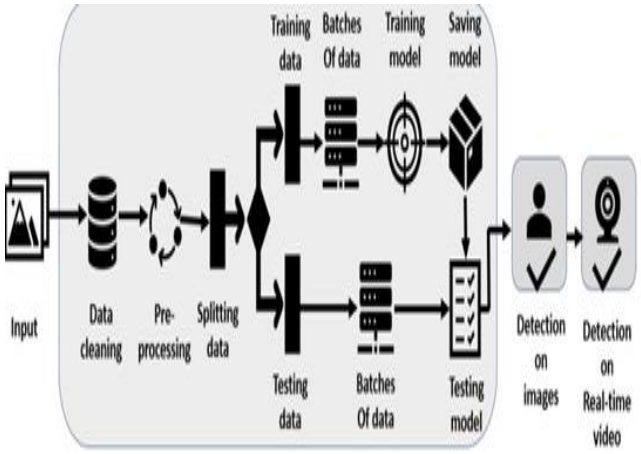

Fig 1

In order to train a custom face mask detector, we need to break our project into two distinct phases, each with its own respective sub-steps (as shown by Figure 1 above):

1. Training: Here we will focus on loading our face mask detection dataset from disk, training a model (using Kera's/TensorFlow) on this dataset, and then serializing the face mask detector to disk
2. Deployment: Once the face mask detector is trained, we can then move on to loading the mask detector, performing face detection, and then classifying each face as with maskor without mask

3.2UseCaseDiagram

Use case diagrams are typically developed in the early stage of development and people often apply use case modeling for the following purposes:

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts together with domain experts

Use cases share different kinds of relationships. Defining the relationship between two use cases is the decision of the software analysts of the use case diagram. A relationship between two use cases is basically modeling the dependency between the two use cases. The reuse of an existing use case by using different types of relationships reduces the overall effort required in developing a system.

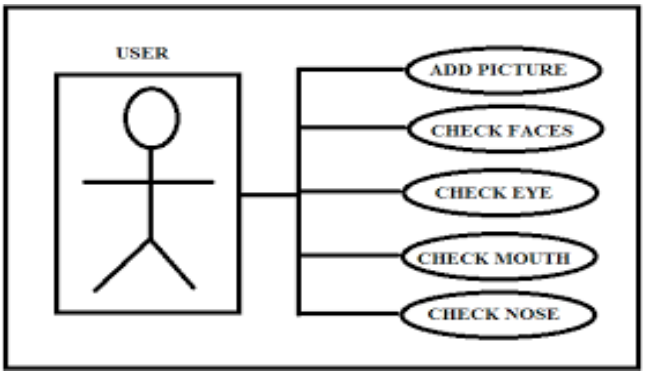

Fig 2

Over the above-mentioned figure 2 we have worked on the Receiving Inputs and checking them from the user which includes Add Picture, Check Faces, Check Eye, Check Mouth and Check Nose

3.3DataFlowDiagram

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles, and arrows, plus short text labels, to show data inputs, outputs, storage points, and the routes between each destination. Data flowcharts can range from simple, even hand-drawn process overviews, to in-depth, multi-level DFDs that dig progressively deeper into how the data is handled. They can be used to analyze an existing system or model a new one.

Like all the best diagrams and charts, a DFD can often visually "say" things that would be hard to explain in words, and they work for both technical and nontechnical audiences, from developers to CEOs. That is why DFDs remain so popular after all these years. While they work well for data flow software and systems, they are less applicable nowadays to visualizing interactive, real-time, or database-oriented software or systems.

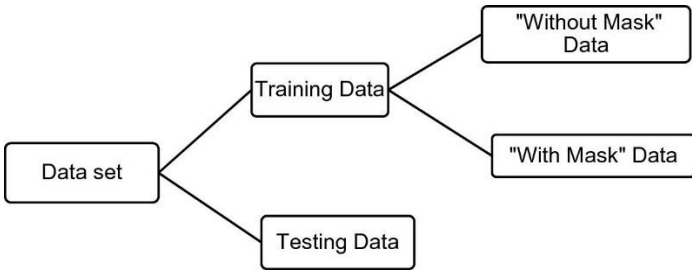

Fig 3

Here in figure 3, we are focusing on the Structure and Flow of Data set Flow, where Training Data and Testing Data are the Two branches of the data set and in Training Data, we have without mask data and With Mask data are Categorized

3.4ClassDiagram

The Unified Modeling Language (UML) can help you model systems in various ways. One of the more popular types in UML is the class diagram. Popular among software engineers to document software architecture, class diagrams are a type of structure diagram because they describe what must be present in the system being modeled. No matter your level of familiarity with UML or class diagrams, our UML software is designed to be simple and easy to use.

UML was set up as a standardized model to describe an object-oriented programming approach. Since classes are the building block of objects, class diagrams are the building blocks of UML.

The various components in a class diagram can represent the classes that will be programmed, the main objects, or the interactions between classes and objects. The class shape itself consists of a rectangle with three rows. The top row contains the name of the class, the middle row contains the attributes of the class, and the bottom section expresses the methods or operations that the class may use. Classes and subclasses are grouped together to show the static relationship between each object.

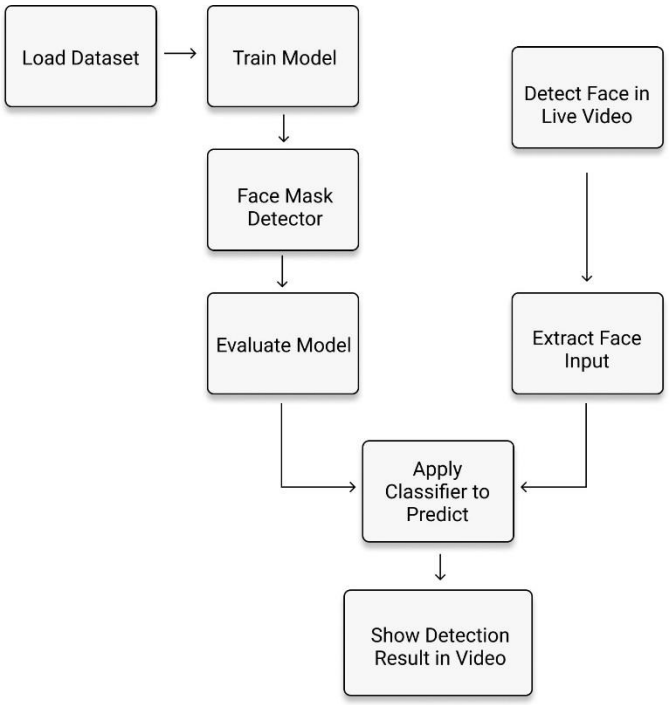

Fig 4

Over the Above Images stay as the origin for the solution which consist of Components such as Local Data set, Train Model, Face Mask Director, Evaluate Model, Apply Classifier to Predict,detect face in Live Video, Extract Face Input, Show Detection Result in video.

Where Train Model Receiving data from the data set and processed and directed Towards the Face Mask Detector for Further Moment. Which further moves to the Evaluate model and Reaches the Apply Classifier to Product block.

On the other hand, Were We have inputs from the, detect face in Live Video via, Extract Face Input to the, Apply Classifier to Predict. Finally, which set to the Show Detection Result in video.

3.5 Pre-processing

A compilation of data from the photos in this dataset for masked face identification and application contained a lot of repetitions and noise. Since a decent dataset determines how accurate a model would be after being trained on it, the data from the datasets were used. The repeats were then manually eliminated once they had been processed. The faulty photos that were discovered in the dataset were removed manually using data cleaning. Finding these pictures was an important step. As is widely known, dealing with corrupt photographs was challenging, but with the help of sincere efforts, we separated the work and cleaned the data set using mask images and those without. removing, spotting, and fixing mistakesin a dataset remove adverse effects from any predictive model.

This part explains the procedure of pre-processing the data and then training on data. First, we define a function name sorted alphanumerically to sort the list in lexicographical order. A function pre-processing is defined, which takes the folder to the dataset as input, then loads all the files from the folder and resizes the images according to the SSDMNV2 model. Then the list is sorted using sorted alphanumerically, and then the images are converted into tensors. Then the list is converted to NumPy array for faster calculation. After this, the process of data augmentation is applied to increase the accuracy after training the model.

3.6 Algorithm

Machine learning

Machine learning is software that learns to perform a task from a collection of examples rather than through a person explicitly defining rules and formulas. This learning software is called a model.

Teaching a model through examples is called training.

Train machine learning with Lobe

This version of Lobe learns to look at images using image classification - categorizing an image into a single label overall. We are working to expand to more types of problems and data in future versions.

Image classification is categorizing an image into a single label to represent its content. Apps using image classification could:

- Tell you when someone is coming up your front steps
- Send you photos of a new bird that just started showing up at your bird feeder
- Count the number of push-ups you have done in a workout
- Alert you when a shelf is empty
- Read signs in your environment

Lobe is not doing any reasoning or understanding of the content in your images. Image classification learns to find any patterns from your images - things like textures, colors, and shapes that can be used to separate your labels.

Haar Cascades Algorithm

A Haar classifier, or a Haar cascade classifier, is a machine learning object detection program that identifies objects in an image and video.

The algorithm can be explained in four stages:

- Calculating Haar Features
- Creating Integral Images
- Using AdaBoost
- Implementing Cascading Classifiers

It is important to remember that this algorithm requires a lot of **positive images** of faces and **negative images** of non-faces to train the classifier, like other machine learning models.

Calculating Haar Features

The first step is to collect the Haar features. A **Haar feature** is essentially calculations that are performed on adjacent rectangular regions at a specific location in a detection window. The calculation involves summing the pixel intensities in each region and calculating the differences between the sums. Here are some examples of Haar features below.

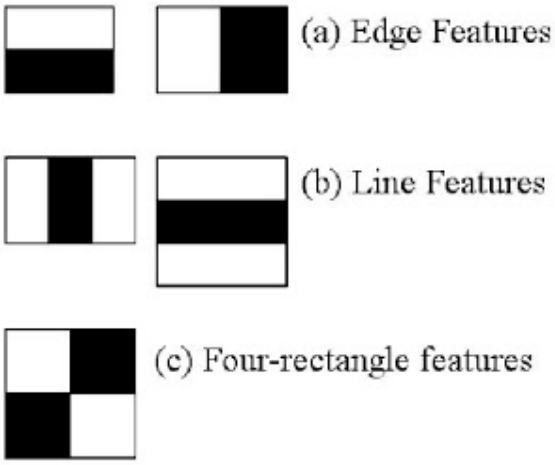

Fig 5

Where in figure 5 we have:
a) Edge Features
B) Line Features
c) Four - rectangle Features

Identifying these elements in a huge photograph can be challenging. Integral images come into play in this situation since they allow for a reduction in the number of operations.

Creating Integral Images

Instead of computing at every pixel, it instead creates sub-rectangles and creates array references for each of those sub-rectangles. These are then used to compute the Haar features.

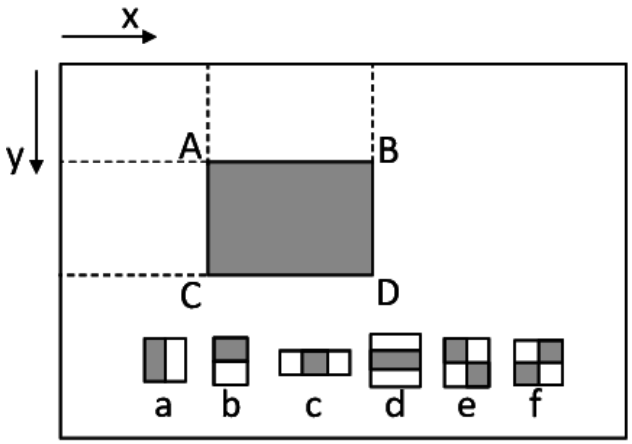

Fig 6

Here in Figure 6,It is important to note that nearly all the Haar features will be **irrelevant** when doing object detection, because the only features that are important are those of the object.

Ada boost Training

Ada boost essentially chooses the best features and trains the classifiers to use them. It uses a combination of **"weak classifiers"** to create a **"strong classifier"** that the algorithm can use to detect objects.

By sliding a window across the input image and computing Haar characteristics for each area of the image, weak learners are produced. This distinction is contrasted with a learnt threshold that distinguishes between non-objects and objects. These are "weak classifiers," whereas a strong classifier requires a lot of Haar characteristics to be accurate.

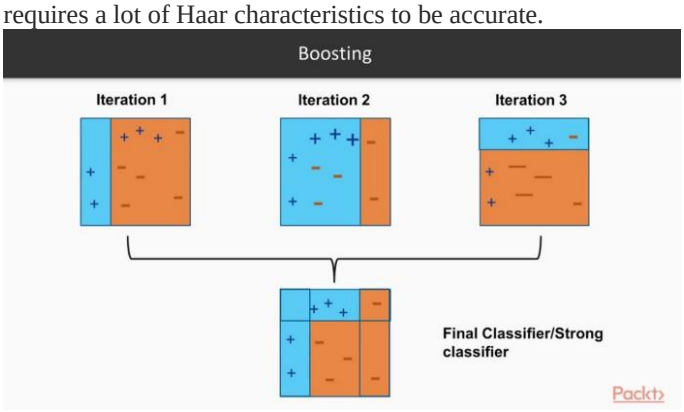

Fig 7

In the above Figure 7 we have iteration part1 ,2 and 3 which are Gone through the process of Boosting and Final Classifier / Strong Classifier is resulted. The last step combines these weak learners into a strong learner using **cascading classifiers**.

Implementing Cascading Classifiers

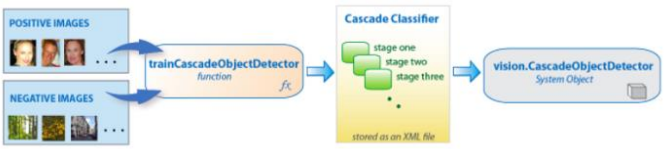

Fig 8

On the Figure 8 where the cascade classifier is composed of several stages, each of which contains a group of weak learners. Boosting is used to train weak learners, resulting in a highly accurate classifier from the average prediction of all weak learners.

Based on this prediction, the classifier either decides to go on to the next region (negative) or decides to report that an object was identified (positive).

Because the bulk of the windows do not contain anything of interest, stages are created to reject negative samples as quickly as feasible.

Because classifying an object as a non-object would significantly hurt your object detection system, it is crucial to have a low false negative rate. You may watch a demonstration of Haar cascades below. The red boxes represent "positives "Model usage

A model is a piece of code. You can export your model into a variety of industry-standard formats to run on Mac and Windows, the web, or mobile and edge devices.

Labeling

Labeling is assigning categories to your images to create examples that teach Lobe. These examples are commonly known as a dataset. From this dataset, Lobe will learn to automatically predict a label for a given image.

3.7Dataset

Import and label your images in Lobe

Images - import common image files directly from your computer. Lobe supports JPEG, PNG, BMP, and Web formats.

Camera - use any connected camera source to capture images directly in Lobe. You can optionally provide a label for these images. Hold down the camera button to capture a burst of images.

The dataset consists of images containing images of people wearing masks and images with people without masks.

Sample Dataset

No Mask

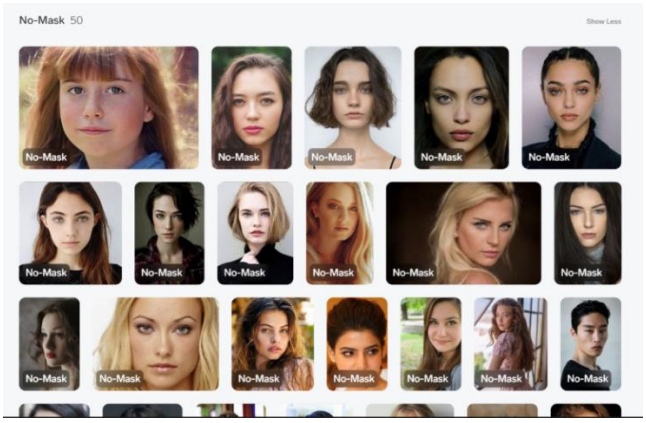

Fig 9

We have figure 9 has a collection of data where people are without mask.

Mask

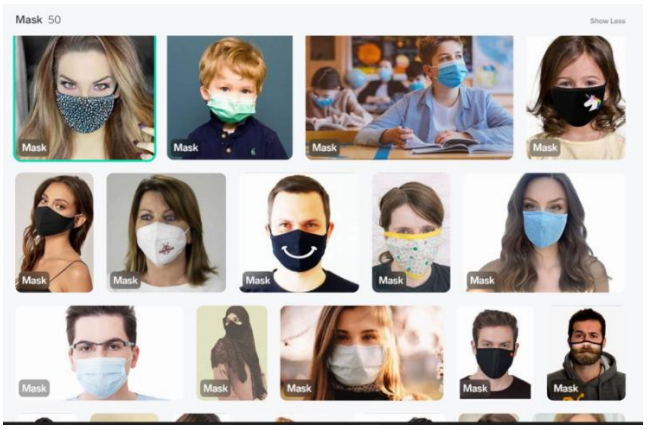

Fig 10

Here in Figure 10, we have a collection of data people with mask

Import an existing dataset

Folders- import existing labeled images by using folder names as the labels.

You can create new labels or edit existing ones by using the text box in the bottom cornerof eachimage.

The max image size Lobe can process is 178,956,970 pixels. For a square image, that is about 13,300 x 13,300 pixels. We recommend staying lower resolution for faster processing because Lobe will resize and crop your image to a 224 x 224 square.

Collect images that you expect to see in the real world

Lobe can only learn the patterns that exist in the images you provide as examples. Collect images from the same source you expect to use with your exported model.

Capture as many variations as possible

By gathering photographs under various circumstances, try to capture all the differences that naturally occur. Check out various backgrounds, lighting, orientations, and zoom levels. This teaches Lobe what noise is and what portions of the image are important for making predictions.

Make sure your content can be seen in the image's central square.

Lobe crops the central square of your photographs while training your model.

Label usage

Label for each desired prediction

Create a label for each category you want the model to predict.

Catch-all-label

For each image, Lobe will always suggest one of the labels you provide. Create a label called "None" to serve as a catch-all bucket for photographs that do not fit any of your preferred categories if you anticipate the model seeing images that do not.

Gather as many different images as you can

For lesser situations, a general guideline is 100–1,000 photos per label. Lobe has a minimal requirement of 5 photos per label in order to get your project going quickly.

Have roughly equal number of images per label

The quantity of photos for each label should be equal. If the data is unbalanced, Lobe will predict labels with more photographs than others and discard labels with fewer images**.**

External cameras

Yes! The default camera attached to your computer will be the main source used by Lobe. Use the source selector in the Label or Use views to change cameras.

Make that the chosen camera is operational and connected to your computer, and that the Lobe app has authorization to utilize cameras, if you do not see an image from the camera.

Speed up labeling

- Use shift + click or cod/ctrl + click to multi-select many images and label them at the same time.
- Capture a burst of labeled images from the webcam.
- Use folder names as labels to import existing datasets.
- Use your arrow keys to move the image selection and label entirely with your keyboard

Import images from a CSV

Direct CSV use with Lobe is not possible. But you may download images and run your exported model on a CSV of image URLs using a Python command-line tool that we designed in addition to an external desktop application.

Training

Your model gains the ability to anticipate the right labels based on your samples during training. Your examples can be compared to a set of flashcards. As it trains, your model will repeatedly scan the flashcards for patterns that will help it identify the correct answers. Lobe automatically starts training when your examples meet the minimum requirements. To start training, you need:

Imported images to label as examples

At least two labels

At least five images per label

Lobe will also follow best practices to continue training when you make changes to your examples. If you make large changes or add/remove labels, Lobe will reset training and start training a new model.

After automatic training has completed, you can manually optimize your model to train for longer for better real-world performance **(File > Optimize Model)**.

Training time

Depending on how challenging it is to discriminate between the labels in your examples and how many examples you have, training time can vary greatly. It can take anywhere from minutes to hours, and for very big situations, it can even take days.

To get an estimated time, simply hover your cursor over the training progress. This estimated training time is changed every few seconds based on your progress and the processing speed of your machine. If you are using your computer for other tasks, you can see fluctuations in the CPU and memory that are accessible.

Train the model

history = model.fit_generator(train_generator,
epochs=10,
validation_data=validation_generator,
callbacks=[checkpoint])

Train for longer

When training has completed, you can optimize your model by selecting File > Optimize Model. Optimizing a model performs additional training and can take much longer to complete, but will generally help find a better version of your model.

While optimizing, Lobe will keep training for as long as your model is improving and does not have a set end-time.

Change themodel architecture

Without any setup or configuration, Lobe chooses the ideal model architecture for your situation. Project Accuracy: a bigger model with generally better accuracy on more challenging issues but longer prediction periods and memory use.

• Speed: A smaller model may have a lesser accuracy but a faster prediction speed and use of memory. The Raspberry Pi or other edge devices can also benefit from this model's optimization.

Changing your project will automatically train a new model and reset any prior training that has already been done.

Training results

- Your results let you know which photos your model accurately and erroneously predicts. Red colors indicate inaccurate predictions, whereas green markings indicate accurate ones. How confident the model was in its prediction is shown by the width of the label bar.Whether your model is successfully learning all the labels with View > All Images selected.
- Approximately how well it will work on new images with View > Test Images selected. Learn more about test images.
- Which images and labels confuse your model by selecting your individual labels in the sidebar.

Test images

Test images are a random subset of your examples that Lobe hides from your model during training. Lobe automatically splits your examples into two parts:

- Random 80% is used to train your model.
- Random 20% is held out and used to test your model.

Play

Play with the trained model

To test your model on fresh images in real time, use photographs from your PC or the webcam as a video feed. Look for trends where your model is producing inaccurate images and actively try to fool it. By providing input on the model's predictions, you may help it get better.

Correct an incorrect prediction

You can directly update the prediction text box and add the image and label as an example to provide feedback to your model.

Alternately, you can provide an example by clicking the image's proper or incorrect buttons.

To keep getting better, Lobe will automatically train with these fresh examples.

View multiple images at once

You can only view one image at a time or stream video using your camera right now export

Using the model

Your model is a set of data files that other applications can use to make forecasts. Both the model's structure and the weights that come from training are stored in these files.

To construct an API, you may utilize the model files either locally in your app or in the majority of the major cloud platforms. To aid in the early stages of your app development, Lobe additionally offers your model as a local API.

Export option

For leveraging models, Lobe offers a several workflows: utilizing a local API, customizing beginning projects, or interacting directly with model files.

Integrate with an external app

8

Check whether the other app you are using with Lobe can load and run model files directly or if it needs to call an API to obtain predictions via a network.

Using an API for predictions

You can create a network request to obtain predictions from your model if your intended app can do POST requests, handle JSON, and base64 encode images.

You can utilize the Lobe Connect local API with no additional settings if the app is already operating locally on the same computer or network as Lobe, such as Origami Studio. For the Lobe Connect local API to work, Lobe must be running and your project open.

The 'Create an API for flexibility' section below describes how to host your model as an external API if your intended app is operating in a different environment than Lobe or you want to have predictions without the Lobe app. open.

Loading model files directly

Choose the right Model File export option for your app if the intended app can import and run external models.

Look for officially supported converters provided by the desired format if Lobe does not yet support the export, such as TensorRT or OpenVINO. In most cases, a converter from ONNX or TensorFlow model files into the final format is available.

Create an API for flexibility

One of the most flexible methods to connect apps to use your model is by developing a REST API. You can run your model on your own computer or an edge device like a Raspberry Pi, distribute it to a variety of cloud providers, or call it as a service from other products.

Check out Lobe Connect to prototype or experiment with using an API. Check out REST Server for an example using Python and Flask with deployment instructions to Azure App Service on how to build one up yourself to run in any location.

Use the TensorFlow.js export option and look at the sample code in the export package for loading and running the model on an image to construct an API using a Node server such as Express.

Use or customize a starter project

Check out the Web Sample beginning project, which uses TensorFlow.js without requiring a backend server, if you wish to make predictions in the browser. For quicker load times and prediction speed at the risk of possibly lesser accuracy when using TensorFlow.js in the browser, we advise choosing the Speed model under File > Project Settings.

For a sample utilizing CoreML for iPhone or iPad apps, see iOS App if you wish to execute your model in a mobile environment. For a sample utilizing TensorFlow Lite to run on an Android device, see Android App. For mobile apps, we advise choosing the Speed model from File > Project Settings because it can make predictions more quickly but at the risk of less accuracy.

Classifying images on your computer

You can export your model as TensorFlow and use the ImageTools desktop app to run your model on a folder of images or a spreadsheet of image URLs.

Write an app from scratch

If you have more experience as a developer, you can export model files directly and use their underlying frameworks such as TensorFlow, CoreML, or ONNX.

Where applicable, we recommend using our libraries inPython and .NET for working with Lobe exports as they provide useful helper functions for loading and processing data, as well as formatting return values and even working with the local API for quickly prototyping.

Local Python app or hosted on Azure, Google Cloud, or AWS

TensorFlow is the model export format. TensorFlow's Saved Model is a commonly used file format in Python programmers that use TensorFlow 1.x or 2.x. TensorFlow web services can deploy TensorFlow Saved Model files to execute inference as an API on the cloud. For an example of how to run the TensorFlow export, see our Python SDK.

Apple iOS

In order to create apps for iOS, iPad, and Mac, export your model as Core ML. If you require low latency and a tiny memory footprint on iOS, use the Speed model from File > Project Settings.
Android or Raspberry Pi

For use in mobile and IoT applications, export your model as TensorFlow Lite. If you require minimal latency and a smaller memory footprint on the edge, choose the Speed model from File > Project Settings.

ONNX

Export your model as ONNX for cross-compatible applications, including edge devices and .NET applications.

Web Applications

Export your model as TensorFlow.js for browser-based JavaScript or server-side Node applications. Use the Speed model from File > Project Settings if you need low latency and a smaller memory footprint in the browser.

Lobe Connect

Lobe will host a local API to call your model via a REST endpoint. Use this option to mock a service that runs predictions while developing your app.

To run the local API

Capture an input image as a base64 string. Make sure the base64string doesn't include the 'data: image/jpeg;base64,' prefix that sometimes is added.

Spreadsheets and local images

Export your model as a TensorFlowSavedModel to use with our helper code for running predictions on spreadsheets of image URLs or local image files.

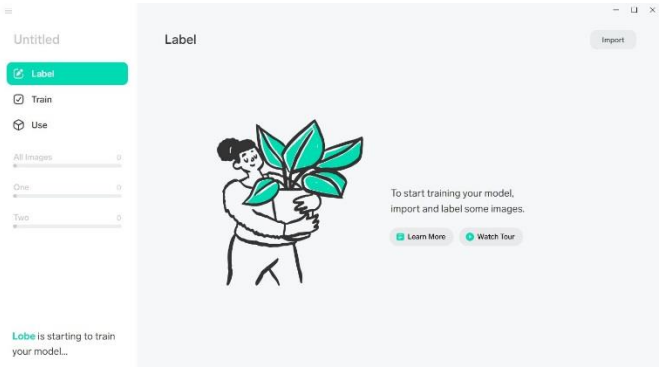

Fig 11

Here Figure 11 are the base of display of Label, Train and Use to start training your model import and label some images and clarification.

3.9 Sample Screenshots

With Mask

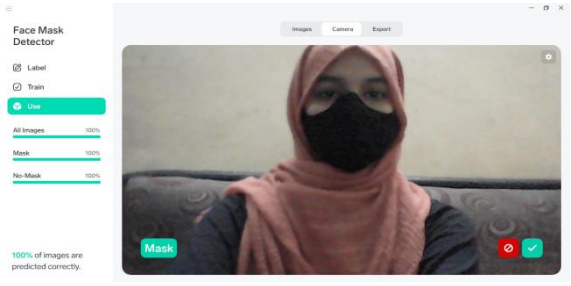

Fig 12

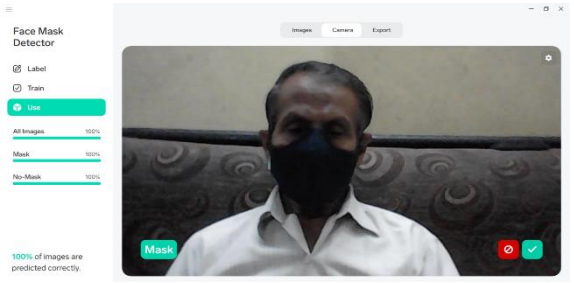

Fig 13

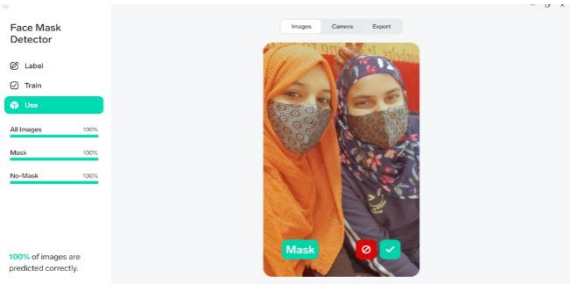

Fig 14

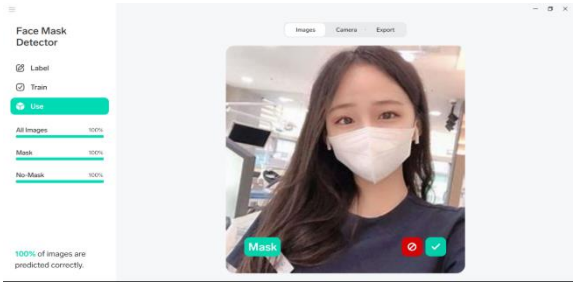

Fig 15

Without Mask

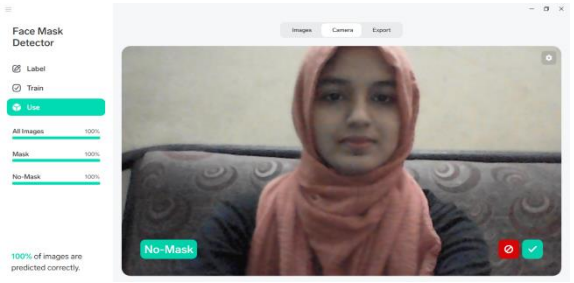

Figure 16

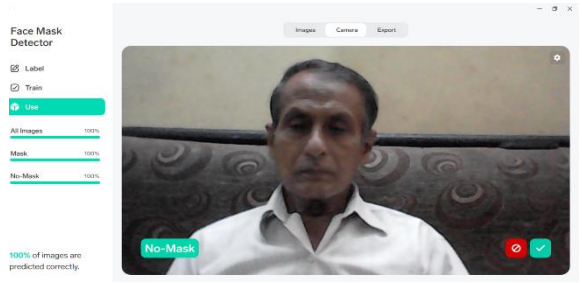

Fig 17

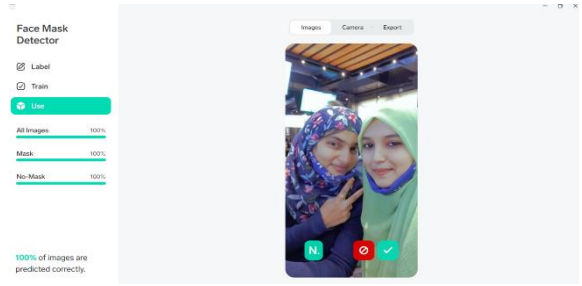

Fig 18

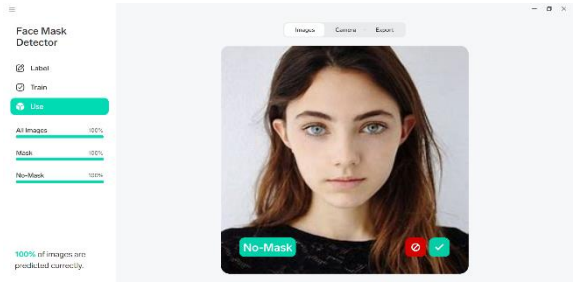

Fig 19

4. RESULTS

4.1 Confusion Matrix

The tests are conducted using the recall metrics, Precision, F1-score, and the corresponding macro average and weighted avg, with the goal of illustrating the potential utility of the system. Utilizing these measures has as its goal evaluating the system from many angles. Recall and precision show how well the model can identify genuine positives. Recall considers both the model's accuracy in identifying false positives and false negatives. False positives happen when an object is classified as a face in this case of face detection using masks. Since it is untrue that a plant is a positive face, the system, for instance, frames a plant as a face with a face mask. There are several reasons why this might happen in our system, which is why it takes a lot of effort to gather a sizable database so that the model being trained can better identify the target classes (faces). False negatives happen when a face is missed in the initial step of recognition because it has covered parts that make categorization challenging; in this approach, the initial classifier is a pre-made tool. The F1-score, on the other hand, gives an overall assessment of the system's performance. It combines precision and recall (in a single value), with 0 denoting poor performance and 1 denoting the best performance (all cases properly detected).

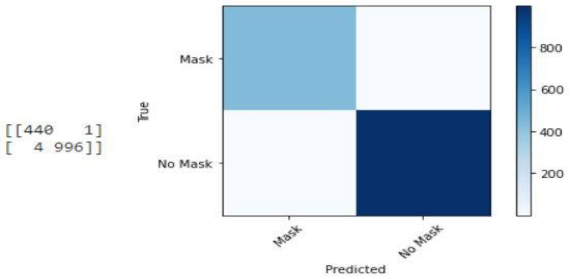

Fig 20

Figure 20 - Confusion matrix classifier

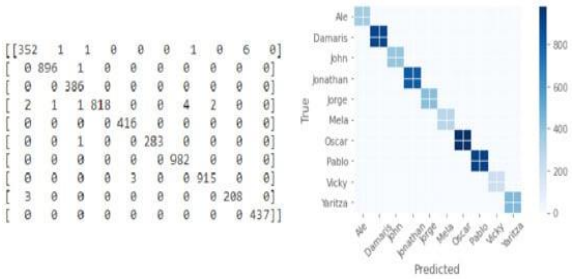

Fig 21

Figure 21 - Confusion matrix with a face mask

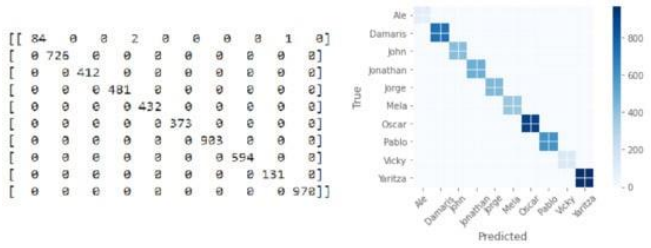

Fig 22

Figure 22- Confusion matrix without mask

4.2 Performance Metrics

20% of the data are used for testing and the remaining data are used to train the face classification model. For training data, a 99.6% accuracy is achieved at about 10 epochs after the model reaches convergence. Finally, after comparing the test data and trained model, the confusion matrix had the following values: 440 true positives (VP), 1 false negative, 4 false positives, and 996 true negatives. When comparing the two graphs, there is a correlation between the high accuracy attained and the minimal FN and FP detection during training. Equations are used to describe specificity, sensitivity,

accuracy, and precision. Aside from thatit has an accuracy of 99.65%, precision of 99.09%, sensitivity of 99.77%, and specificity of 99.6%.

Accuracy = (VP + VN)/Total
Precision = VP/(VP + FP)
Sensitivity = VP/(VP + FN)
Specificity = VN/(VN + FP)

Lobeusestwo popularmodelsforimageclassification:

1. *Accuracy: ResNet-50V2*

Arguments

- include_top: whether to include the fully-connected layer at the top of the network.
- weights: one of None (random initialization), 'ImageNet' (pre-training on ImageNet), or the path to the weights file to be loaded.
- input_tensor: optional Keras tensor (i.e., output of layers. Input ()) to use as image input for the model.
- input_shape: optional shape tuple,
- pooling: Optional pooling mode for feature extraction when include_top is False.
- None means that the output of the model will be the 4D tensor output of the last convolutional block.
- avg means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor.
- max means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if include top is True, and if no weights argument is specified.
- classifier_activation: A str or callable. The activation function to use on the "top" layer. Ignored unless include_top=True. Set classifier_activation=None to return the logits of the "top" layer. When loading pretrained weights, classifier_activation can only be None or "softmax".

Arguments

- num_classes (int): Number of classes
- width_mult (float): Width multiplier - adjusts number of channels in each layer by this amount
- inverted_residual_setting: Network structure
- round_nearest (int): Round the number of channels in each layer to be a multiple of this number
  - Set to 1 to turn off rounding
  - block: Module specifying inverted residual building block for mobilenet
  - norm_layer: Module specifying the normalization layer to use

Both models utilize transfer learning with pretrained weights from the ImageNetdataset. Transfer learning lets you train better models with less data and gives a better starting point for training on larger data.

4.3 ROC Analysis

Fig 23

Figure 23- Training loss vs. accuracy classifier

Fig 24

Figure 24- Training loss vs. accuracy without a face mask

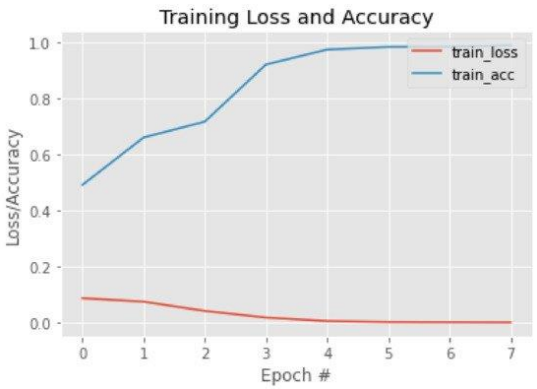

Fig 25

Figure 25-Training loss vs. accuracy without a face mask

5. Conclusion

There should be action made to slow the COVID-19 pandemic's spread. Our intention is to give users a pleasant learning and information-gathering experience. The computer models that were used in this project This method produced speedy and precise results for facial recognition. The COVID-19 mask detector we are making here today has the potential to be utilized to help make sure that you and others are safe (but I will leave that up to the medical professionals to decide on, implement, and distribute in the wild).

This prototype system enables face identification of wearers of masks as well as those who are not, and it might be applied as a low-computing-consumption suggestion for personnel access control.Images are used to test the two models of this system, improving precision, and optimizing each model. The name tag and likelihood of success are provided for the person's face that was successfully classified from the database. The system's three stages made it possible to extract the crucial features of a person's face and then utilizes a straightforward neural network to perform the classification task. In this regard, the studies conducted showed that using "Face Embeddings" as input to the neural network produced satisfactory results.

Because the database was created for this stage with a small number of participants, even though it is made up of numerous photos and has little variability, it is possible to notice an over-adjustment during training. The technique does, however, have the potential to be utilized in other facial recognition applications. It should be noted that if a face cannot be in the database, it will still be recognized, but a tag indicating whether the individual is wearing a mask, "mask" or "no mask," will be added.It should be remembered that the system uses a confidence level of 0.6 to determine whether a face belongs to a specific person. The accuracy is 99.65% when determining whether persons are wearing masks or not. With test data from persons who do not wear masks, the facial recognition model has an accuracy of 99.96%, and with test data from people who do, it has an accuracy of 99.52%. This lays the groundwork for future studies that may deepen the understanding of this area.

6. Future Work

Face mask human recognition has several uses in a variety of fields. A human face has several features that stand out and may be distinguished from many other objects. It locates faces by eliminating basic features like the eyes, nose, and mouth, among others, and then makes use of them to recognize a face. Usually, a fact classifier qualified and accommodating to distinguish between facial and non-facial parts is used. Human faces also have distinctive surfaces that can be used to distinguish them from other objects. Additionally, the edges of highlights can help to identify objects from faces the different techniques are dependent on the specific requirements of the application. Every strategy has advantages and disadvantages; thus, we must choose the optimal strategy based on the situation. Marketers are becoming more interested in face detection. It can be utilized in a variety of settings, including airports, where it can be crucial to identify whether travelers are wearing masks. Videos of travelers' data may be stored in the system at the entrance. a hospital This system can be combined with CCTV cameras, and the data from those cameras may be used to check whether their employees are wearing masks. The spectrum of this system's application includes security systems in many different public spaces, including malls, hospitals, IT firms, and others.

Therefore, it should be taken into consideration to combine the first and second stages into one model in the future and to develop a unique algorithm that searches for faces both with and without masks. This prevents identifying faces with and without mas after using the Open CV face detector first.

This will speed up the processing time and strengthen the model. Additionally, it is suggested that a comparison of the models used for learning transfer be conducted in the future in order to identify the best model and network trained under unfavorable evaluation conditions. The final models can be compacted and deployed after they have finished training.

References

[1] World Health Organization et al. Coronavirus disease 2019 (covid-19): situation report, 96. 2020. - Google Search. (n.d.).https://www.who.int/docs/default-source/coronaviruse/situation-reports/20200816-covid-19-sitrep-209.pdf?sfvrsn=5dde1ca2_2.

[2] Social distancing, surveillance, and stronger health systems as keys to controlling COVID-19 Pandemic,PAHODirector says-PAHO/WHO|PanAmericanHealthOrganization.(n.d.).https://www.paho.org/en/news/2-6-2020-social-distancing-surveillance-and-stronger-health-systems-keys-controlling-covid-19.

[3] V.C.C.Cheng,S.C.Wong,V.W.M.Chuang,S.Y.C.So,J.H.K. Chen,S.Sridhar,K.K.W.To,J.F.W.Chan, I.F.N.Hung,and P.L.Ho, et al. "The role of community-wide wearing of face mask for control of coronavirus disease 2019 (COVID-19) epidemic due to SARS-CoV-2", J. Infect. vol. 81, pp. 107–114, 2020, [GoogleScholar][CrossRef] [PubMed].

[4] K. Dang, andS. Sharma, "Review and comparison of face detection algorithms", In Proceedings of the 7thInternational Conference Confluence 2017 on Cloud Computing, Data Science and Engineering, Noida, India, January2017, vol. 12–13, pp. 629–633. [Google Scholar]

[5] G.J. Chowdary, N.S. Punn, S.K. Sonbhadra, and S. Agarwal, "Face Mask Detection using Transfer Learning ofInceptionV3, in: International Conference on Big Data Analytics, Springer", Cham, 2020, pp. 81-90, pp. 1–11, doi:10.1007/978-3-030-66665-1_6.

[6] N. Abbassi, R. Helaly, M. A. Hajjaji, and A. Mtibaa, "A deep learning facial emotion classification system: aVGGNet-19basedapproach," in Proceedings of the 202020thInternational ConferenceonSciencesand Techniques of Automatic Control and Computer Engineering (STA), IEEE, Monastir, Tunisia, December 2020, pp. 271–276.

[7] Early Diabetes Discovery from Tongue Images,Safi Naveed S, Geetha G and Leninisha S, The Computer Journal, 2020, https://doi.org/10.1093/comjnl/bxaa022

[8] Intelligent Diabetes Detection System based on Tongue Datasets, Safi Naveed S, GurunathanGeetha , Current Medical Imaging Reviews 2019;15(7):672-678, doi: 10.2174/1573405614666181009133414

[9] R. Helaly, M. A. Hajjaji, F. M'Sahli, and A. Mtibaa, "Deep convolution neural network implementation foremotionrecognitionsystem,"in Proceedingsofthe202020thInternationalConferenceonSciencesandTechniques of Automatic Control and Computer Engineering (STA), IEEE, Monastir, Tunisia,December 2020, pp. 261–265.

[10] "Mask Detect" Application. Available online:https://play.google.com/store/apps/details?id=es.upv.mastermoviles.intemasc.rec(accessedon14August2021)