**CHAPTER 1**

**INTRODUCTION**

**1.1 ORGANIZATION PROFILE**

An Q Spiders (Chennai), we develop innovative software projects and solutions that drive business growth. Our team of experienced professionals provides exceptional software support to help you achieve your goals.

**Our Services:**

At Q Spiders (Chennai), we offer a range of services to help your business thrive. Our services include custom software development, web application development, mobile app development, and software integration. We also provide software testing and maintenance services.

• Cloud Services

• Managed IT Services

• Software Development

• Software Maintenance

• Software Integration

• Software Consultancy

An Q Spiders (Chennai), we specialize in developing high-quality software projects and providing software solutions for our clients. We also provide software support to help you with any issues or concerns you may have with your software. Whether you need custom software development, software integration, or software maintenance, we have the expertise to provide you with the best possible solution. Contact us today to learn more about how we can help you achieve your software goals.

**1.2 OBJECTIVE**

The primary objective of this project is to develop a robust and efficient facemask detection system utilizing deep learning techniques, specifically designed for integration with surveillance cameras. The system aims to enhance public health and safety by ensuring compliance with mask-wearing guidelines in various environments, such as public transportation hubs, workplaces, and retail spaces.

In the wake of the COVID-19 pandemic, facemasks have become a crucial preventive measure. However, manual enforcement of mask-wearing can be resource-intensive and prone to human error. This project addresses this challenge by automating the detection process through a sophisticated neural network model that can accurately identify whether individuals in the camera's field of view are wearing masks. The project involves several key components, starting with data preparation. A diverse dataset of images, categorized into 'with\_mask' and 'without\_mask', is preprocessed and augmented to improve the model's generalization capabilities. This ensures that the model can handle various real-world scenarios, such as different lighting conditions and angles.

The core of the project is the training of a deep learning model using TensorFlow and the MobileNetV2 architecture, known for its balance between accuracy and computational efficiency. The model is fine-tuned to detect masks with high precision, minimizing false positives and negatives. This phase includes extensive experimentation with hyperparameters to optimize the model's performance. Once trained, the model is deployed in a real-time detection system. This involves integrating the model with OpenCV to process live video streams from surveillance cameras. The system continuously scans the footage, identifying faces and determining mask status. When an individual without a mask is detected, the system triggers an audio notification, alerting the relevant authorities or security personnel. This automated alert mechanism serves as a proactive measure to enforce mask-wearing, thereby reducing the risk of viral transmission.

**CHAPTER 2**

**SYSTEM ANALYSIS**

**2.1 EXISTING SYSTEM**

The existing system for face mask detection utilizing surveillance cameras lacks comprehensive automation and real-time notification features. Typically, these systems rely on manual monitoring or rudimentary algorithms that lack the sophistication required for accurate detection and notification. Often, there is a disconnect between detection and action, with no immediate response mechanism in place for situations where a person is detected without a mask. These systems may lack integration with audio notification capabilities, which are crucial for alerting individuals in real-time about non-compliance with mask-wearing protocols. Furthermore, the absence of a unified framework for training and deploying the detection model limits scalability and adaptability across different surveillance setups. Overall, the existing systems suffer from inefficiencies, limited functionality, and a lack of real-time response mechanisms, thereby hindering their effectiveness in ensuring compliance with mask-wearing guidelines.

**2.1.1 DISADVANTAGES:**

* Lack of real-time detection capability.
* Absence of audio notification for mask violations.
* Limited scalability for handling multiple surveillance points.
* Manual intervention required for monitoring.
* Inability to adapt to changing environmental conditions.

**2.2 PROPOSED SYSTEM**

The proposed system is a sophisticated face mask detection solution integrated with surveillance cameras, leveraging deep learning technology for accurate and real-time detection of individuals without masks. The system incorporates a unified framework for training and deploying the detection model, enabling seamless integration with existing surveillance setups. Utilizing state-of-the-art deep learning architectures such as MobileNetV2, the system achieves high accuracy in detecting individuals without masks while minimizing false positives. Additionally, the proposed system features real-time notification capabilities, employing audio alerts to immediately notify individuals when non-compliance with mask-wearing protocols is detected. This proactive approach not only enhances compliance but also contributes to public health safety by reducing the risk of virus transmission. Furthermore, the system is designed for scalability and adaptability, allowing for easy deployment across various surveillance environments, including public spaces, workplaces, and transportation hubs. By combining advanced deep learning algorithms with real-time notification features, the proposed system represents a significant advancement in face mask detection technology, offering an effective solution for enforcing mask-wearing protocols and promoting public health.

**2.2.2 ADVANTAGES:**

* Real-time mask detection with surveillance camera integration.
* Immediate audio notification for non-compliance.
* Scalable solution for monitoring multiple locations simultaneously.
* Automated monitoring without human intervention.
* Adaptability to various environmental conditions for reliable detection.

**2.3 FEASIBILITY STUDY**

The feasibility study for the proposed face mask detection system with notification audio involves assessing the technical, economic, and operational aspects to determine the viability and potential success of the project.

**2.3.1 Technical Feasibility**

From a technical standpoint, the project appears highly feasible. The advancement of deep learning technologies, coupled with the availability of pre-trained models such as MobileNetV2, provides a solid foundation for accurate face mask detection. These technologies have been extensively researched and implemented in various computer vision applications, indicating their reliability and effectiveness. Additionally, the integration of audio notification features is technically achievable, leveraging libraries and tools for audio playback within the application environment. Compatibility with existing surveillance cameras and systems can be ensured through standardized protocols and interfaces, further enhancing technical feasibility.

**2.3.2 Economic Feasibility**

The economic feasibility of the project depends on factors such as development costs, operational expenses, and potential returns on investment. While the initial development and implementation costs may be moderate, considering expenses related to hardware, software, and personnel, the long-term benefits outweigh these investments. The system's ability to enhance public health safety by promoting mask-wearing compliance can lead to tangible economic benefits, such as reduced healthcare costs associated with virus transmission and potential regulatory compliance advantages for businesses and public institutions. Furthermore, the scalability of the system allows for cost-effective deployment across various surveillance environments, maximizing its economic viability.

**2.3.3 Operational Feasibility**

Operationally, the proposed system offers significant advantages in terms of ease of use, maintenance, and integration with existing workflows. The user interface can be designed to be intuitive and user-friendly, requiring minimal training for operators and administrators. Regular maintenance tasks, such as software updates and system monitoring, can be automated to streamline operational efficiency. Moreover, the system's integration with existing surveillance infrastructure ensures minimal disruption to ongoing operations, facilitating smooth deployment and adoption. Overall, the operational feasibility of the project is high, with the potential to seamlessly integrate into diverse operational environments.

**CHAPTER 3**

**SYSTEM SPECIFICATION**

**3.1 HARDWARE REQUIREMENTS**

* Processor : Multi-Core Processor
* RAM : At least 8GB
* Storage : Sufficient SSD or HDD
* Audio Output Device : System Speaker
* Video : Surveillance or Remote Camera

**3.2 SOFTWARE REQUIREMENTS**

* Operating System : Windows or Linux
* Programming Language : Python
* IDE : Visual Studio
* Deep Learning Frameworks : Tensorflow or Pytorch
* Image Processing Libraries : OpenCV
* Audio Playback Library : playsound

**CHAPTER 4**

**SOFTWARE DESCRIPTION**

**4.1 PYTHON**

Python is an interpreted, object-oriented, high-level programming language that can be used for a wide variety of applications. Python is a powerful general-purpose programming language. First developed in the late 1980s by Guido van Rossum who named it after the BBC Comedy TV series Monty Python’s Flying Circus. Python is open-source programming language so numerous independent programmers are continually building libraries and functionality for it.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of their features support functional programming and aspect- oriented programming including metaprogramming and metaobjects. Many other paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It uses dynamic name resolution (late binding), which binds method and variable names during program execution.

Python is used for server-side web development, software development, mathematics, and system scripting, and is popular for Rapid Application Development and as a scripting or glue language to tie existing components because of its high-level, built-in data structures, dynamic typing, and dynamic binding. Program maintenance costs are reduced with Python due to the easily learned syntax and emphasis on readability. Additionally, Python's support of modules and packages facilitates modular programs and reuse of code

**Common Uses of Python**

* Python is commonly used for developing websites and software, task automation, data analysis, and data visualization.
* Python is used in multiple AI solutions like advanced computing, image recognition, data processing, and more.
* Python has its capacity of contributing to the gaming industry in a massive way by using its frameworks and its prime uses are to software developers.

**Characteristics of PHP**

Important characteristics make Python practical nature possible:

* + Simplicity
  + Cross-platform Language
  + Free and Open Source
  + Large Standard Library
  + Extensible
  + GUI Programming Support
  + Embeddable
  + Dynamic Memory Allocation

### Python Variables

Python variables are simply containers for storing data values. Unlike other languages, such as Java, Python has no command for declaring a variable, so you create one the moment you first assign a value to it. Python variables are simply containers for storing data values.

* The value of a variable is the value of its most recent assignment.
* Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
* Variables can, but do not need, to be declared before assignment.
* Variables used before they are assigned have default values.
* Typically, we use a single letter or a more descriptive name to represent a variable.

**TensorFlow**

TensorFlow is an open-source deep learning framework developed by the Google Brain team. It is designed to facilitate the development and deployment of machine learning models by providing a comprehensive and flexible ecosystem of tools, libraries, and community resources. TensorFlow supports various machine learning and deep learning algorithms, allowing researchers and developers to create complex neural networks and other machine learning models with ease.

At its core, TensorFlow operates on a computational graph model, where operations are represented as nodes and data, or tensors, flow between them along the edges. This architecture enables efficient execution on diverse hardware platforms, including CPUs, GPUs, and TPUs, making it highly scalable from mobile devices to large-scale distributed systems.

One of the key features of TensorFlow is its flexibility. Users can construct models using high-level APIs like Keras, which simplifies the process of building and training neural networks. Alternatively, they can dive deeper with TensorFlow's low-level operations to gain more control and optimize performance. TensorFlow also supports automatic differentiation, which is crucial for training neural networks using gradient-based optimization algorithms.

TensorFlow provides extensive support for various types of deep learning models, such as convolutional neural networks (CNNs) for image recognition, recurrent neural networks (RNNs) for sequence modeling, and transformers for natural language processing. Additionally, TensorFlow's ecosystem includes TensorFlow Extended (TFX) for productionizing machine learning workflows, TensorFlow Lite for deploying models on mobile and embedded devices, and TensorFlow.js for running models in the browser.

**Uses of Tensorflow**

TensorFlow is widely used in various fields and applications due to its robust capabilities in machine learning and deep learning. In the healthcare sector, TensorFlow helps in developing models for diagnosing diseases from medical images, predicting patient outcomes, and personalizing treatment plans. In finance, it is used for algorithmic trading, fraud detection, risk management, and credit scoring. TensorFlow's ability to handle large datasets and complex models makes it ideal for these data-intensive tasks.

In the realm of natural language processing (NLP), TensorFlow powers applications like chatbots, sentiment analysis, machine translation, and text summarization. Its support for transformers and recurrent neural networks (RNNs) allows it to process and understand human language effectively. In image and video analysis, TensorFlow is used for facial recognition, object detection, and autonomous driving technologies, leveraging convolutional neural networks (CNNs) for accurate visual recognition and interpretation.

Moreover, TensorFlow is instrumental in the development of recommendation systems, such as those used by e-commerce platforms to suggest products to users based on their browsing history and preferences. In the field of robotics, TensorFlow aids in developing models for perception, motion planning, and control, enabling robots to navigate and interact with their environments intelligently.

**4.2 DEEP LEARNING**

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It’s achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that contain many layers.

While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful:

* + 1. Deep learning requires large amounts of labelled data. For example, driverless car development requires millions of images and thousands of hours of video.
    2. Deep learning requires substantial computing power. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.

### 4.2.1 Deep Neural Network

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks. Deep neural networks (DNN) is an class of machine learning algorithms similar to the artificial neural network and aims to mimic the information processing of the brain. DNN shave more than one hidden layer (l) situated between the input and output layers. In addition to object recognition, which identifies a specific object in an image or video, deep learning can also be used for object detection. Object detection algorithms like YOLO can recognize and locate the object in a scene, and can locate multiple objects within the image.

**CHAPTER 5**

**PROJECT DESCRIPTION**

**5.1 PROBLEM DEFINITION**

The widespread adoption of face masks has become a crucial measure to mitigate the transmission of the virus. However, ensuring compliance with mask-wearing protocols in public spaces, workplaces, and other communal environments poses significant challenges. Manual monitoring of mask-wearing behavior is impractical and resource-intensive, leading to inconsistencies and lapses in enforcement. Moreover, the absence of real-time feedback mechanisms exacerbates the risk of non-compliance and compromises public health safety. Consequently, there is a pressing need for an automated face mask detection system capable of accurately identifying individuals without masks in real-time and issuing prompt notifications to promote adherence to mask-wearing guidelines.

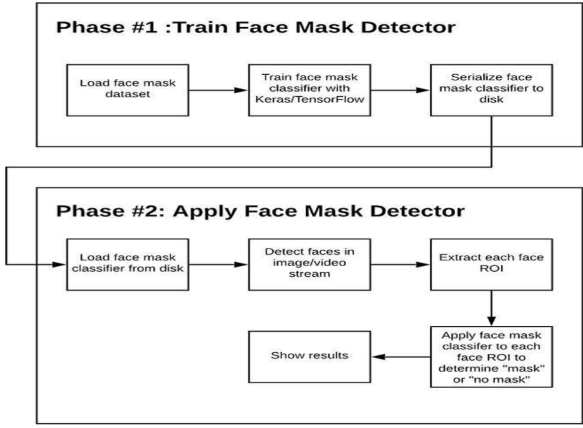
Existing solutions for face mask detection often lack the sophistication and real-time notification capabilities required to address this critical need effectively. Traditional methods relying on manual observation or rudimentary algorithms are prone to errors and do not provide timely feedback to individuals violating mask-wearing protocols. Furthermore, the integration of audio notification features is often overlooked, limiting the system's ability to engage and alert individuals in real-time. Thus, there is a clear gap in the existing solutions landscape, necessitating the development of a comprehensive face mask detection system with notification audio functionality. This system aims to enhance compliance with mask-wearing guidelines, thereby contributing to the overall public health efforts to combat the spread of infectious diseases.

**5.2 MODULE DESCRIPTION**

1. Train.py
2. Detect mask.py
3. Main.py

**5.2.1 Train.py**

The train.py module constitutes a pivotal component of the face mask detection system, focusing on the training of the deep learning model essential for accurate mask detection. This module encompasses a series of intricate processes facilitated by various packages from TensorFlow, scikit-learn, and other libraries. Initially, it undertakes the crucial task of loading image data sourced from the designated dataset directory. Subsequently, these images undergo preprocessing, including resizing and normalization, before being partitioned into distinct training and testing sets.



**Fig 5.1 Phases of Facemask Detector**

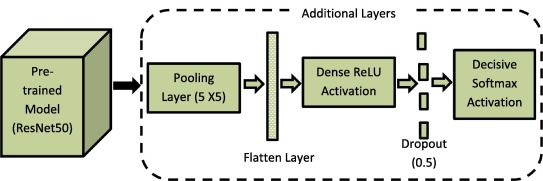
The trained facemask classifier obtained after transfer learning is applied to detect mask and no mask faces. The ultimate objective of enforcement of wearing of face mask in public area will only be achieved after retrieving the personal identification of faces, violating the mask norms.

The module discriminatively pre-trained the Convolutional Neural Network (CNN) on the biased dataset. CNN is a type of artificial neural network, which is widely used for image/object recognition and classification. The pre-training was performed using the open-source libraries in python. The other approach is to first remove the inherent bias present in the available dataset and then execute supervised learning over a domain specific balanced dataset. The bias is alleviated by applying random over sampling with data argumentation.

### Fine-tuning of pre-trained model

Facemask detection is achieved through deep neural networks because of their better performance than other classification algorithms. But training a deep neural network is expensive because it is a time- consuming task and requires high computational power.

To train the network faster and cost-effective, deep-learning-based transfer learning is applied here. Transfer learning allows to transferring of the trained knowledge of the neural network in terms of parametric weights to the new model. It boosts the performance of the new model even when it is trained on a small dataset.



**Fig 5.2 MobileNet Pre-trained Model**

Leveraging the MobileNetV2 architecture as the base model, custom layers are incorporated for classification purposes. To enhance the robustness and generalization of the model, data augmentation techniques such as rotation, zoom, and horizontal flip are applied via an ImageDataGenerator. Following model compilation with binary cross-entropy loss and Adam optimizer, the training commences, with the progression of epochs being meticulously tracked for performance evaluation. Upon completion, the trained model is rigorously assessed using classification metrics, and subsequently serialized to disk in the H5 format to ensure seamless access for subsequent usage. Additionally, the module meticulously plots training loss and accuracy curves to provide comprehensive visual insights into the model's training dynamics.

There are several pre-trained models like AlexNet, MobileNet, ResNet50 etc. that had been trained with 14 million images from the ImageNet dataset. **MobileNet\_V2** is chosen as a pre-trained model for Facemask classification. The last layer of MobileNet\_V2 is fine-tuned by adding five new layers. The newly added layers include an average pooling layer of pool size equal to 5 × 5, a flattering layer, a dense ReLU layer of 128 neurons, a dropout of 0.5 and a decisive layer with soft max activation function for binary classification.

Various face images are analyzed in terms of processing complexity. It is observed that dataset, we consider primarily, contains two major classes that is, mask and non-mask class but the mask class further, contains an inherent variety of occlusions other than surgical/cloth facemask, for example, occlusion of ROI by other objects like a person, hand, hair or some food item.These occlusions are found to impact the performance of face and mask detection. Thus, obtaining an optimal trade-off between accuracy and computational time for face detection is not a trivial task. So, an image complexity predictor is being proposed here. Its purpose is to split data into soft versus hard images at the initial level followed by a mask and non-mask classification at a later level through a facemask classifier.

The important question that we need to answer is how to determine whether an image is soft or hard. The answer to this question is given by “Semi-supervised object classification strategy”.

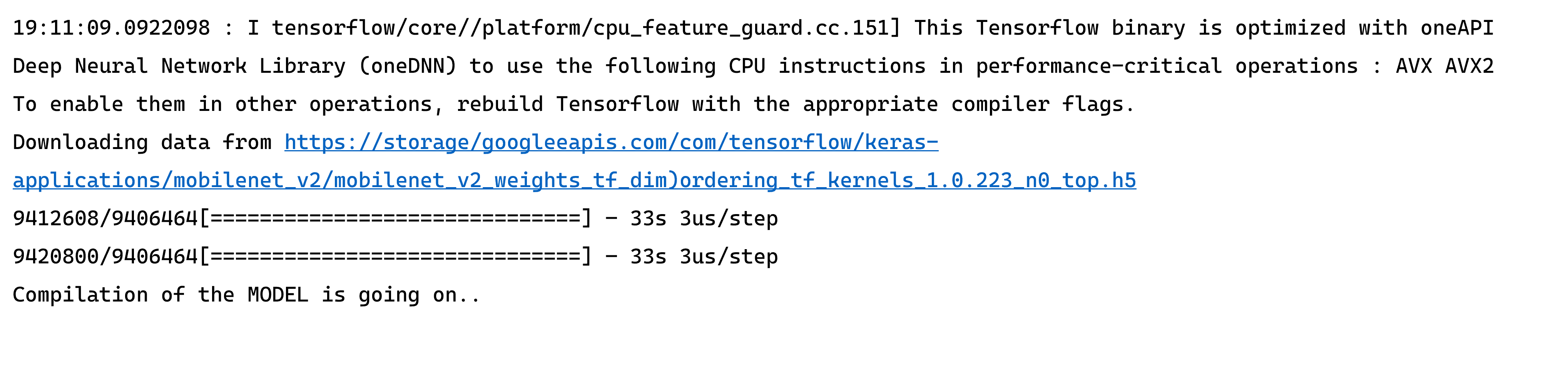


**Fig 5.3 Image Complexity Predictor 3**

**Facemask Dataset**

Images from various sources are used to build a dataset. The size of datasets can be expanded by the application of data enhancement techniques. The photographs are stored in two files, “training dataset” and “test dataset,” each of which comprises 80 and 20% of the images, respectively. Bounding boxes, sometimes known as “data annotations,” are created around an area of interest using a variety of methods. Labeling pictures as “mask” or “NO mask” will be done using the Label Image tool in the proposed system.

A facemask dataset with a total of more than 3000 images, categorized over two classes namely with mask and without mask. The number of with mask images in dataset are over 280 whereas non- masked images are only remainders. It is observed that dataset is put up with an extrinsic class imbalanced problem that may introduce a bias towards the majority class. So, analyze the performance of the image classifier once with the original dataset and proposed dataset.

****

**Fig 5.4 Epoch Train with TensorFlow**

**Deploy.protext**

Prototxt files are model definition files that are required when you train a CNN model. The snapshot. caffemodel and mean. binaryproto files are output files that are created after the model training is complete.

In deep learning, especially with frameworks like Caffe, the deploy.prototxt file is used for deploying trained neural network models during the inference or prediction phase. The deploy.prototxt file specifies the architecture of the network and the configuration settings needed for deploying the model for tasks such as image classification, object detection, or other types of predictions.

### Model Architecture

It outlines the architecture of the neural network, specifying the types of layers used (e.g., convolutional, pooling, fully connected) and their parameters. Unlike the prototxt file used during training (train\_val.prototxt), the deploy.prototxt file often excludes layers related to training, such as loss layers and data input layers.

### Input Layer

Defines the input layer, specifying the type of data expected by the model during inference. The dimensions and properties of the input data, such as the image size and number of channels.

### Preprocessing Steps

Describes any preprocessing steps applied to the input data before it is fed into the network for inference. This may include mean subtraction, scaling, or other normalization techniques.

### Output Layer

Specifies the output layer, which defines the format and size of the model's predictions. For classification tasks, this layer often uses the softmax activation function to produce probability scores for different classes.

### Blob Shape and Size

Describes the shape and size of the output blobs produced by each layer of the network.

### Parameters and Hyperparameters

Includes any additional parameters or hyperparameters needed for the network during deployment. This may include configuration settings like learning rates, weight decay, or other settings relevant to the inference phase.

### Inference Configuration

Can include settings related to how the model should handle inputs during deployment, such as batch size and whether to run the model on a GPU or CPU.

### Quantization (optional)

In some cases, the deploy.prototxt file might include information about quantization, which is a technique for reducing the precision of weights and activations to optimize model size and inference speed.

### Caffe Model file

In the context of the Caffe deep learning framework, a "CaffeModel" file typically refers to the file that contains the learned weights (parameters) of a trained neural network model. The model file usually has the extension .caffemodel. This file stores the trained parameters of the network that have been learned during the training phase.

### Trained Parameters

The .caffemodel file contains the learned weights and biases of the neural network's layers. These parameters represent the knowledge the model has gained during the training process.

### Architecture and Weights

While the deploy.prototxt file defines the architecture of the network, the .caffemodel file complements it by storing the learned weights associated with each layer.

### Usage during Inference

During the deployment or inference phase, the trained model architecture (defined in deploy.prototxt) is combined with the learned parameters (stored in the .caffemodel file) to make predictions on new data.

### Compatibility

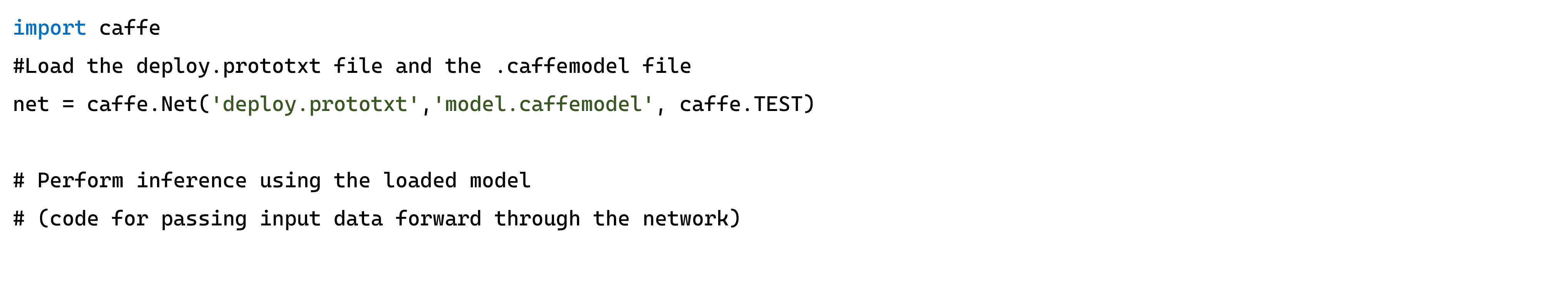
The .caffemodel file is specific to the version of the neural network architecture defined in the corresponding deploy.prototxt file. It's important to use the correct model architecture and weights together.

### Conversion and Deployment

The .caffemodel file is often used in conjunction with other files, such as the deploy.prototxt file and mean files, to deploy the trained model for tasks like image classification or object detection.

### Downloading Pre-trained Models

Many deep learning frameworks and libraries provide pre-trained models for popular tasks. Users can download the pre-trained .caffemodel files to use these models for specific applications without going through the training process.

****

**Fig 5.5 Caffe Model File Syntax**

**Mask Detector Model File**

When dealing with face mask detection using deep learning, the term "mask model file" typically refers to a file containing the trained weights and architecture of a deep neural network model specifically designed for face mask detection. This file is usually in a format that can be loaded into a deep learning framework for inference.

### Data Collection and Preprocessing

Collect a dataset of images containing people with and without face masks. Annotate the dataset to indicate regions of interest (ROI) corresponding to faces and whether a mask is present.

### Model Architecture

Choose or design a neural network architecture suitable for object detection or image classification Common architectures include Convolutional Neural Networks (CNNs) or more advanced architectures like Single Shot MultiBox Detector (SSD) or You Only Look Once (YOLO).

### Training

Train the model using the annotated dataset to learn the patterns associated with face masks. Fine-tune a pre-trained model or train from scratch depending on the size of your dataset.

### Model Saving

Save the trained model's architecture and learned weights into a file. This file is what is commonly referred to as the "mask model file."

### Inference:

During the inference phase (when you want to detect masks in new images or video streams), load the saved model file into your application. Use the model to make predictions on new data and identify whether faces in the input have masks or not.

### Plot PNG File.

To generate a plot and save it as a PNG file during the training of a machine learning model, you can use popular plotting libraries like Matplotlib in Python.

The random data for demonstration purposes. Replace epochs, train\_loss, and val\_loss with the actual variables you are tracking during your training loop.

* plt.plot(): Plots the training and validation loss.
* plt.title(), plt.xlabel(), plt.ylabel(): Adds a title and labels to the plot.
* plt.legend(): Adds a legend to the plot to distinguish between training and validation loss.
* plt.savefig(): Saves the plot as a PNG file with the specified filename ('training\_plot.png' in this case).

****

**Fig 5.6 Plot Png Image**

* + 1. **Detect Mask**

Detecting face is a sort of computer vision technology that can recognize people’s faces in digital photographs such descriptive of recognition and detection. The Model describes the loading the mask detector with the maskNet to detecting the face.

Facial recognition entails recognizing the face in a picture as belonging to person X rather than person Y. It is frequently used for biometric applications, like unlocking a smartphone. Facial analysis attempts to learn something about people based on their facial features, such as their age, gender, or the emotion they are displaying. Facial tracking technique is commonly used in video analysis and attempts to follow a face and its features (eyes, nose, and lips) from frame to frame.

**Data At Source:** OpenCV was used to increase the size of the images. At the time, the images were titled “cover” and “no veil.” The images available were of various sizes and goals and were most likely extracted from various sources or from machines (cameras) of various goals.

**Data Processing:** Ventures, as indicated below, were applied to all the raw data images to convert them into clean forms that could be handled by a neural organization AI model.

* + - * Resizing the information picture (256 x 256).
      * Applying the shading sifting (RGB) over the channels (Our model MobileNetV2 underpins 2D 3 channel picture).
      * Scaling/Normalizing pictures utilizing the standard mean of PyTorch work in loads.
      * Center trimming the picture with the pixel estimation of 224x224x3.
      * Finally Converting them into tensors (Similar to Numpy exhibit).
      * Training
      * Deployment.

The Module describes the common libraries to process the detection of faces and masks through device camera to detect the face and result the alert message.

### Tensor Flow

TensorFlow is an open-source software library. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google’s Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

TensorFlow accepts data in the form of multi-dimensional arrays of higher dimensions called tensors. Multi-dimensional arrays are very handy in handling large amounts of data. TensorFlow works on the basis of data flow graphs that have nodes and edges. As the execution mechanism is in the form of graphs, it is much easier to execute TensorFlow code in a distributed manner across a cluster of computers while using GPUs.

**Keras**

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.

**Sklearn**

Scikit-Learn, also known as sklearn is a python library to implement machine learning models and statistical modelling. Through scikit-learn, we can implement various machine learning models for regression, classification, clustering, and statistical tools for analyzing these models. The scikit-learn library in Python is built upon the SciPy stack for efficient numerical computation. It is a fully featured library for general purpose machine learning and provides many useful utilities in developing deep learning models.

**Imutils**

Imutils library, a Python toolset designed to streamline common image processing operations, particularly when working with OpenCV. This library, developed by Adrian Rosebrock, offers several convenient functions that enhance the readability and efficiency of our code. Imutils is the resize function, which allows us to resize video while preserving their aspect ratio by calling Video Stream library. This functionality proves invaluable when adapting video for display or processing without distorting their original proportions.

The *translate* function simplifies the process of shifting an image or video in the x and y directions. Correcting the orientation of images becomes more accessible with the *rotate* function, enabling us to rotate images by a specified angle. Imutils streamlines the display of images or video with the *imshow* function, handling the underlying OpenCV commands needed to present images in a window.

**Matplotlib**

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals.

Matplotlib consists of several plots like line, bar, scatter, histogram etc.

**Open CV**

OpenCV is a Python open-source library for computer vision in artificial intelligence, machine learning, facial recognition, etc. The term "computer vision" (abbreviated as "CV") in OpenCV refers to a branch of research that enables computers to comprehend the content of digital images like pictures and movies. To comprehend the content of the images is the goal of computer vision. It takes the description of the images-which may be of an object, a text description, a three-dimensional model, etc.-and extracts it from the images. Computer vision, for instance, can help cars by enabling them to recognize various roadside items, such as pedestrians, traffic signs, and traffic lights, and then respond appropriately.

* **Object Classification:** In this process, new objects are classified as belonging to one or more of your training categories by a trained model on a dataset of specific objects.
* **Object identification:** In the object identification phase, our model will pinpoint a specific instance of an object. For instance, it may parse two faces in an image and identify Rohit Sharma and Virat Kohli, respectively.

There are two typical techniques to recognize the pictures:

### GRAYSCALE

Images that just include the two colors black and white are known as grayscale images. Black is considered to have the lowest intensity, whereas white has the highest, according to the contrast assessment of intensity. The computer gives each pixel in the grayscale image a value dependent on how dark it is when we use it.

### RGB

An RGB is a mixture of red, green, and blue that results in a new color. The computer extracts Each pixel's value, which then organizes the information into an array for interpretation.

* + 1. **Main.py**

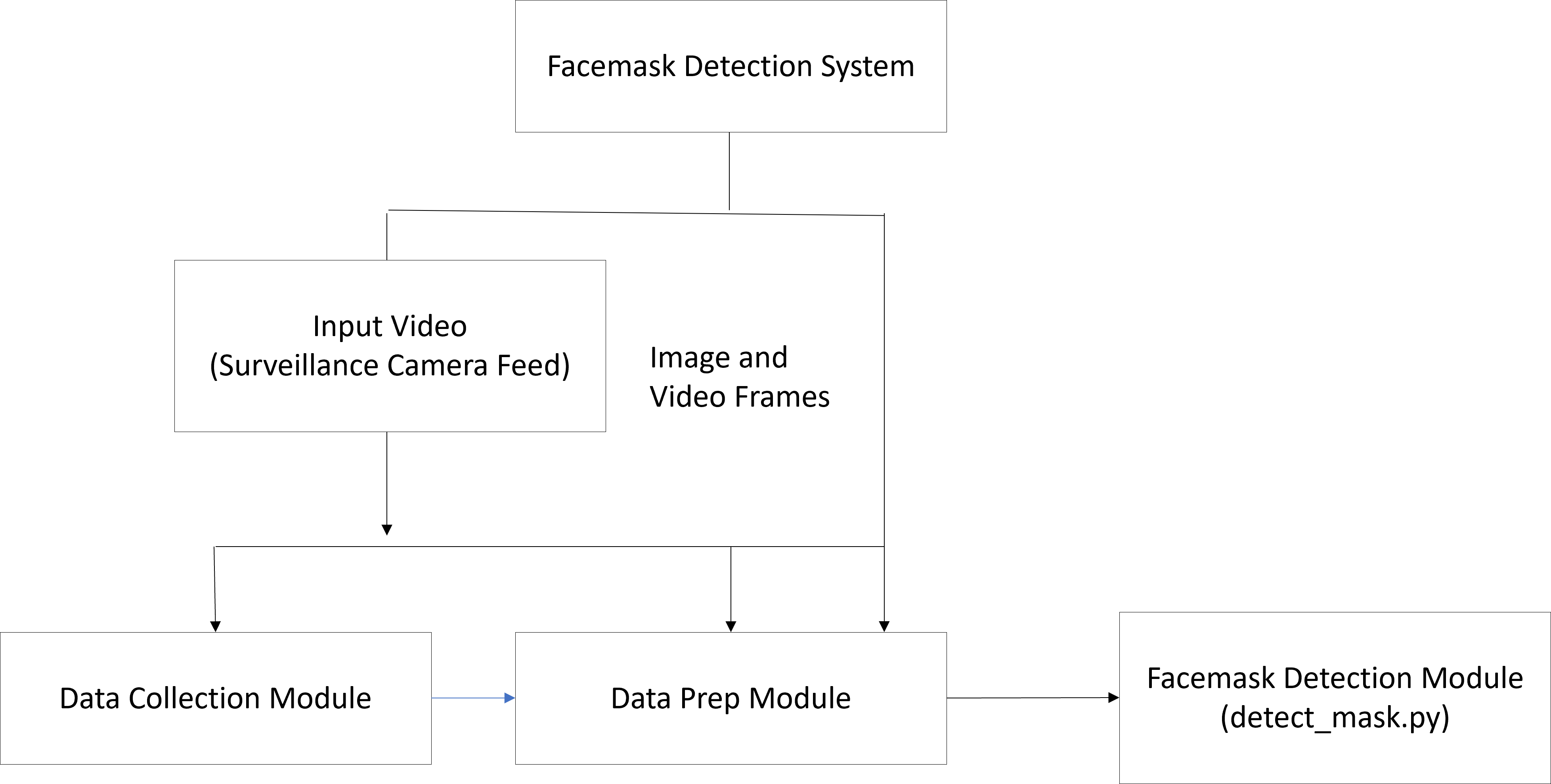
Serving as the nexus of the face mask detection system, the main.py module orchestrates a harmonious integration of the data preparation, model training, and mask detection processes. Through seamless collaboration with the data\_prep, train\_model, and detecting\_mask modules, this pivotal entity drives the holistic workflow of the system.

The module's operational paradigm is encapsulated within the main() function, which meticulously sequences the execution of discrete tasks, each marked by informative progress indicators. Initially, the data preparation phase is initiated, laying the groundwork for subsequent model training endeavors. Following this preparatory phase, the module seamlessly transitions into the model training stage, wherein the intricacies of deep learning are harnessed to imbue the model with the capability to discern between masked and unmasked faces.

Subsequently, with the trained model at its disposal, the module pivots towards real-time mask detection, leveraging the capabilities of the detect\_mask module to analyze surveillance camera feeds and promptly intervene upon the detection of non-compliance. In essence, the main.py module embodies the orchestration of the face mask detection system, epitomizing its operational prowess and efficacy in ensuring adherence to mask-wearing protocols.

* 1. **USE CASE DIAGRAM**

A use case is a list of steps, typically defining interactions between a role (known in Unified Modeling Language (UML) as an "actor") and a system, to achieve a goal. The actor can be a human, an external system, or time. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. Use Case Diagram has actors like sender and receiver. Use cases show the activities handled by both sender and receiver.



**Fig 5.7 Use Case Diagram**

* I**nput Video (Surveillance Camera Feed):** The system ingests input from a surveillance camera feed, capturing video frames in real-time.
* **Data Collection Module:** Responsible for collecting video frames from the input video stream and passing them to subsequent modules for further processing.
* **Data Preparation Module:** This module (implemented in data\_prep.py) handles the preprocessing of image data required for training the face mask detection model. It loads, resizes, normalizes, and partitions the image data into training and testing sets.
* **Face Mask Detection Module:** The core component of the system, comprising the detect\_mask.py module. It performs real-time detection of face masks in video frames captured from the surveillance camera feed. The module utilizes pre-trained models for face detection and face mask detection to identify and label faces as either "Mask" or "No Mask." Detected faces are overlaid with bounding boxes and labels for visual feedback.
  1. **DATAFLOW DIAGRAM**

A two-dimensional diagram explains how data is processed and transferred in a system.

The graphical depiction identifies each source of data and how it interacts with other data

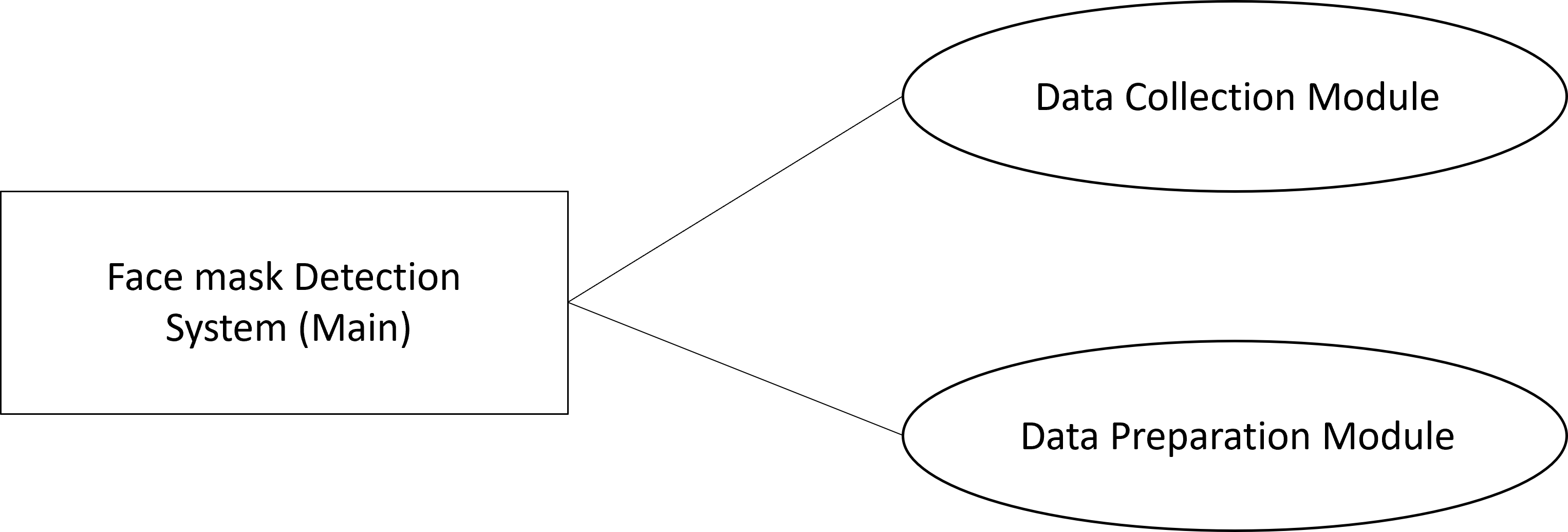
sources to reach a common output. Individuals seeking to draft a data flow diagram must

identify external inputs and outputs, determine how the inputs and outputs relate to each other, and explain with graphics how these connections relate and what they result in. This type of diagram helps business development and design teams visualize how data is processed and identify or improve certain aspects.

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.

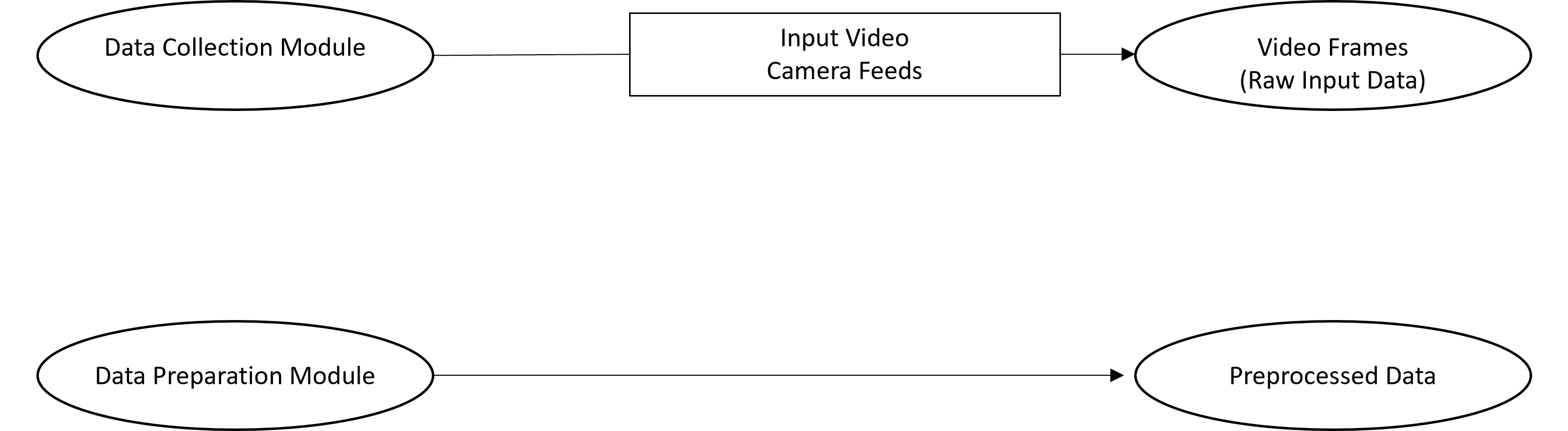
The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.

**LEVEL 0 DFD**

****

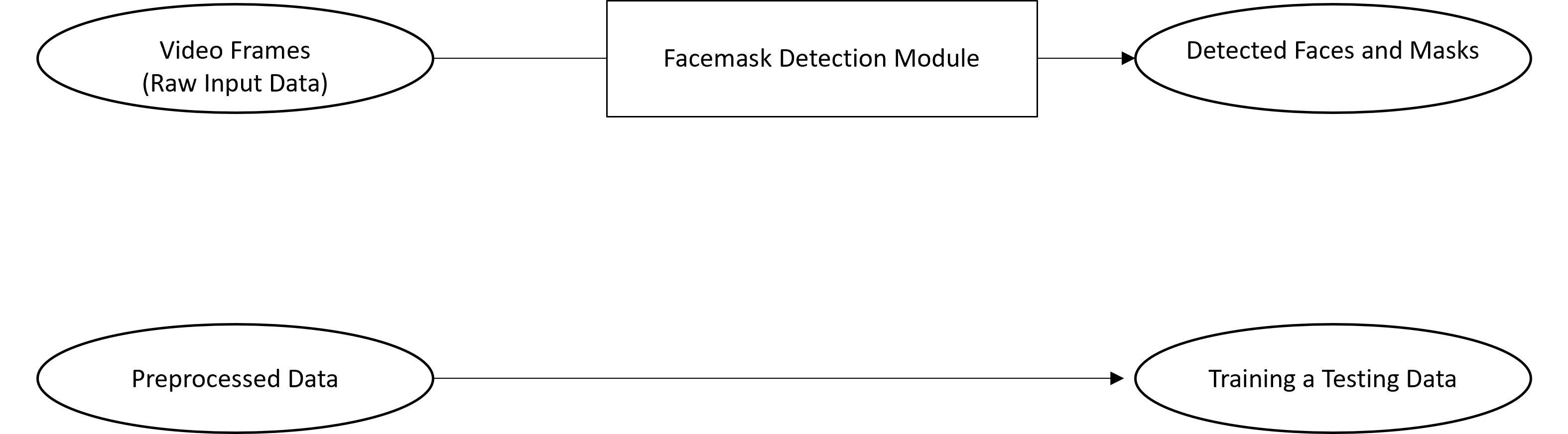
**Fig 5.8 LEVEL 0 DFD**

**LEVEL 1 DFD**

****

**Fig 5.9 LEVEL 1 DFD**

**LEVEL 2 DFD**

****

**Fig 5.10 LEVEL 2 DFD**

**5.5 INPUT DESIGN:**

The input design of the face mask detection system encompasses the mechanisms through which users interact with the system and provide input data for processing. In this project, the primary source of input is the surveillance camera feed capturing real-time video frames. The design considerations for input management include:

The system interfaces with a surveillance camera to capture live video streams. Users can configure the system to access the camera feed either locally or remotely, depending on the deployment environment. Users may need to specify configuration parameters such as camera resolution, frame rate, and connection settings to establish a stable connection and optimize video quality. The system may offer preprocessing options for input data, such as image resizing, noise reduction, and color normalization, to enhance the accuracy of mask detection.

During the training phase, users may have the option to specify data augmentation parameters such as rotation range, zoom range, and shear range to augment the training dataset and improve model generalization. In the training phase, users may provide input parameters such as the number of epochs, batch size, and learning rate for training the face mask detection model.

**5.6 OUTPUT DESIGN:**

The output design focuses on presenting the results of the face mask detection system to users in a comprehensible and actionable format. The design considerations for output management includes, The system displays the live video feed from the surveillance camera, overlaying detected faces with bounding boxes and labels indicating whether a mask is detected or not. In case a person is detected without a mask, the system triggers an alert mechanism, such as a visual alert on the user interface and an audible alarm, to notify users of non-compliance with mask-wearing protocols. The system may log detection results, including timestamps, detected faces, and mask status, to a database for further analysis and reporting. Users can access these logs for audit purposes or generating compliance reports.

During model training, the system provides real-time feedback on training progress, displaying metrics such as training loss, validation loss, training accuracy, and validation accuracy on a graphical user interface (GUI) for monitoring model performance. After model training, the system presents evaluation results, including precision, recall, F1-score, and classification accuracy, to assess the model's performance on the test dataset. This information helps users gauge the efficacy of the trained model and identify areas for improvement.

**CHAPTER 6**

**SYSTEM TESTING**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre- driven process links and integration points . It ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**6.1 UNIT TESTING**

Unit testing involves testing individual components or units of the system in isolation to verify their correctness and functionality. For the face mask detection project, unit tests are conducted for each module, including the data collection module, data preparation module, face mask detection module, and main system module. These tests evaluate the functionality of each module, ensuring that they perform their intended tasks accurately. Unit tests for the face mask detection module, for example, assess its ability to correctly identify faces and determine whether masks are being worn.

**6.2 INTEGRATION TESTING**

Integration testing focuses on testing the interactions and interfaces between different modules to ensure that they work together seamlessly as a unified system. In the face mask detection project, integration tests verify the integration of modules such as data collection, data preparation, and face mask detection, ensuring smooth data flow and communication between them. For example, integration tests validate that the output of the data collection module is correctly processed by the data preparation module and subsequently utilized by the face mask detection module for analysis.

**6.3 ACCEPTANCE TESTING**

Acceptance testing, also known as user acceptance testing (UAT), evaluates the system's compliance with user requirements and determines whether it meets user expectations. In the context of the face mask detection project, acceptance tests are conducted to validate the system's overall performance and usability from the end user's perspective. This includes verifying the accuracy of mask detection, assessing the responsiveness of the user interface, and evaluating the effectiveness of alert mechanisms. Acceptance testing may involve real-world scenarios, such as deploying the system in a live environment and observing its behavior under different conditions.

**6.4 PERFORMANCE TESTING**

Performance testing assesses the system's responsiveness, scalability, and stability under various load conditions. For the face mask detection project, performance tests measure the system's ability to process video streams in real-time, handle multiple concurrent users, and maintain consistent performance over extended periods. Performance metrics such as frame processing rate, resource utilization, and response time are evaluated to identify any bottlenecks or performance issues that may affect the system's reliability and efficiency.

**6.5 REGRESSION TESTING**

Regression testing ensures that recent changes or updates to the system do not inadvertently introduce new defects or regressions in existing functionality. For the face mask detection project, regression tests are performed after implementing new features, bug fixes, or system enhancements to validate that they have not adversely affected the system's behavior or performance. This involves re-running existing test cases and verifying that the system continues to meet its functional and non-functional requirements.

**6.6 BLACK BOX TESTING**

Black box testing involves evaluating the functionality of the system without examining its internal structure or implementation details. For the face mask detection project, black box testing focuses on verifying the system's behavior based on its specified requirements and user interactions.

During black box testing, testers interact with the system as end users would, without knowledge of the underlying algorithms or code. They input various scenarios and observe the system's responses to ensure that it behaves as expected. This includes testing different aspects of the system, such as:

* Input Validation: Verifying that the system handles valid and invalid inputs correctly, such as different video resolutions, frame rates, and lighting conditions.
* Mask Detection Accuracy: Testing the system's ability to accurately detect faces and determine whether masks are being worn, under different scenarios and angles.
* Alert Mechanism: Validating the system's alert mechanism, including visual alerts on the user interface and audible alarms, to ensure timely notification of non-compliance with mask-wearing protocols.
* Real-time Performance: Assessing the system's responsiveness and performance in processing video streams in real-time, without significant delays or lags.
* User Interface Usability: Evaluating the user interface for ease of use, clarity of instructions, and intuitive interaction flow to ensure a positive user experience.

**6.7 WHITE BOX TESTING**

White box testing, also known as structural or glass box testing, involves examining the internal structure and code of the system to evaluate its logic and ensure thorough test coverage. For the face mask detection project, white box testing focuses on verifying the correctness and efficiency of the underlying algorithms and data processing mechanisms.

During white box testing, testers analyze the source code and execute test cases designed to exercise specific paths, conditions, and branches within the code. This includes:

* Code Coverage Analysis: Ensuring that all lines of code, branches, and conditional statements are executed and tested to achieve maximum code coverage.
* Boundary Value Analysis: Testing boundary conditions and edge cases to validate the behavior of the system at the extremes of input ranges, such as minimum and maximum values for image dimensions and pixel intensities.
* Error Handling: Verifying that the system handles exceptions and errors gracefully, with appropriate error messages and fallback mechanisms to prevent system crashes or unexpected behavior.
* Algorithm Validation: Validating the correctness and effectiveness of face detection and mask detection algorithms, including their accuracy, robustness, and computational efficiency.
* Resource Utilization: Assessing the system's resource utilization, such as CPU and memory usage, to ensure optimal performance and scalability under varying load conditions.

**6.8 TEST CASE REPORT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **Test Case Description** | **Expected Result** | **Pass/Fail** |
| TC\_001 | Verify system initialization | System initialized successfully | Pass |
| TC\_002 | Test input validation for video feed | System accepts valid video feed | Pass |
| TC\_003 | Test input validation for camera resolution | System accepts valid resolution | Pass |
| TC\_004 | Test mask detection accuracy | System accurately detects masks | Fail |
| TC\_005 | Test alert mechanism, for no mask detection | System triggers alert | Pass |
| TC\_006 | Test alert mechanism for mask compliance | System does not triggers alert | Fail |
| TC\_007 | Test real-time performance | System processes video in real-time without lag | Fail |
| TC\_008 | Test user interface usability | User interface is intuitive and easy to use | Pass |

**CHAPTER 7**

**SYSTEM IMPLEMENTATION**

System implementation of the face mask detection project involves translating the design specifications and requirements into a functional software system capable of accurately detecting masks in real-time surveillance camera feeds. The implementation process consists of several key steps, including software development, integration of machine learning algorithms, deployment of the system architecture, and testing to ensure reliability and effectiveness.

The first phase of system implementation involves software development, where the various modules of the system, including data collection, preprocessing, mask detection, and user interface, are developed using programming languages such as Python. Machine learning algorithms, such as convolutional neural networks (CNNs), are implemented to train the model on a dataset of masked and unmasked faces, leveraging frameworks like TensorFlow and Keras. Additionally, the system's user interface is designed and implemented to provide a seamless interaction experience for users, incorporating features such as video streaming, face detection overlays, and alert mechanisms.

Following software development, the next phase entails the integration of machine learning algorithms into the system architecture. This involves incorporating trained models into the face mask detection module, optimizing their performance for real-time inference on video streams. Integration efforts focus on ensuring smooth data flow between modules, efficient processing of video frames, and accurate classification of masked and unmasked faces.

Once the system architecture is fully integrated, the final phase of implementation involves deployment and testing to validate the system's functionality, performance, and reliability. The deployed system is tested under various real-world scenarios, including different lighting conditions, camera angles, and crowd densities, to assess its effectiveness in detecting masks accurately and issuing timely alerts. Rigorous testing, including black box, white box, and user acceptance testing, is conducted to identify and address any issues or deficiencies in the system's behavior. Continuous monitoring and refinement ensure that the implemented system meets the desired objectives and delivers a robust solution for face mask detection in surveillance settings.

**CHAPTER 8**

**CONCLUSION AND FUTURE ENHANCEMENT**

**8.1 CONCLUSION**

In conclusion, the face mask detection project has successfully addressed the critical need for automated monitoring of mask compliance in public spaces using surveillance cameras and deep learning technology. The implementation of the system has demonstrated its capability to accurately detect the presence or absence of masks in real-time video streams, contributing to efforts aimed at preventing the spread of infectious diseases such as COVID-19. By leveraging machine learning algorithms and robust system architecture, the project has provided a reliable solution for organizations, businesses, and public authorities to enforce mask-wearing protocols and enhance public safety.

**8.2 FUTURE ENHANCEMENTS**

* Enhance the system to support multiple surveillance cameras simultaneously, allowing for comprehensive coverage of larger areas and improved monitoring capabilities.
* Implement advanced alert mechanisms such as SMS notifications or integration with existing security systems to enable proactive responses to mask non-compliance incidents.
* Integrate real-time analytics capabilities to analyze mask-wearing trends, crowd density, and compliance levels, providing valuable insights for decision-making and resource allocation.
* Implement adaptive learning algorithms to continuously improve the system's performance over time, enabling it to adapt to evolving conditions and enhance its accuracy and reliability.
* Integrate the mask detection system with access control systems to automate entry restrictions based on mask compliance status, ensuring compliance with safety protocols in controlled environments.
* Develop a mobile application for remote monitoring and management of the face mask detection system, allowing administrators to monitor compliance status and receive alerts on their smartphones from anywhere.

**CHAPTER 9**

**APPENDICES**

**9.1 SOURCE CODE**

Train.py

# import the necessary packages

from tensorflow import keras

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

# initialize the initial learning rate

INIT\_LR = 1e-4

EPOCHS = 20

BS = 32

DIRECTORY = r"C:/Users/sathi/OneDrive/Desktop/Face\_Mask\_Project-master/dataset"

CATEGORIES = ["with\_mask", "without\_mask"]

#grab the list of images in our dataset directory, then initialize

print("[INFO] loading images...")

data = []

labels = []

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)

# convert text to Binary

lb = LabelBinarizer()

labels = lb.fit\_transform(labels)

labels = to\_categorical(labels)

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

labels = np.array(labels)

(trainX, testX, trainY, testY) = train\_test\_split(data, labels,

test\_size=0.20, stratify=labels, random\_state=42)

# construct the training image generator for data augmentation

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")

# load the MobileNetV2 network

baseModel = MobileNetV2(weights="imagenet", include\_top=False,

input\_tensor=Input(shape=(224, 224, 3)))

# Create head and the base model

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)

# Call head and the base model

model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them

for layer in baseModel.layers:

layer.trainable = False

# compile our model 17

print("Compilation of the MODEL is going on...")

opt = Adam(lr=INIT\_LR, decay=INIT\_LR / EPOCHS)

model.compile(loss="binary\_crossentropy", optimizer=opt,

metrics=["accuracy"])

# train the head of the network

print("Training Head Started...")

H = model.fit(

aug.flow(trainX, trainY, batch\_size=BS),

steps\_per\_epoch=len(trainX) // BS,

validation\_data=(testX, testY),

validation\_steps=len(testX) // BS,

epochs=EPOCHS)

# make predictions on the testing set

print("Network evaluation...")

predIdxs = model.predict(testX, batch\_size=BS)

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

# show a nicely formatted classification report

print(classification\_report(testY.argmax(axis=1), predIdxs,

target\_names=lb.classes\_))

# serialize the model to disk

print("saving mask model...")

model.save("mask\_detector.model", save\_format="h5")

# plot the training loss and accuracy

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")

plt.xlabel("Epoch #")

plt.ylabel("Loss/Accuracy")

plt.legend(loc="lower left")

plt.savefig("plot.png")

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 cv2

import os

import time

from playsound import playsound

def detect\_and\_predict\_mask(frame, faceNet, maskNet):

# Preprocess the frame

(h, w) = frame.shape[:2]

blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224), (104.0, 177.0, 123.0))

# Pass the blob through the network and obtain the face detections

faceNet.setInput(blob)

detections = faceNet.forward()

faces = []

locs = []

preds = []

# Loop over the detections

for i in range(0, detections.shape[2]):

confidence = detections[0, 0, i, 2]

if confidence > 0.5:

# Extract the coordinates of the bounding box

box = detections[0, 0, i, 3:7] \* np.array([w, h, w, h])

(startX, startY, endX, endY) = box.astype("int")

# Ensure the bounding box falls within the dimensions of the frame

(startX, startY) = (max(0, startX), max(0, startY))

(endX, endY) = (min(w - 1, endX), min(h - 1, endY))

# Extract the face ROI, convert it to RGB, resize it to 224x224, and preprocess it

face = frame[startY:endY, startX:endX]

face = cv2.cvtColor(face, cv2.COLOR\_BGR2RGB)

face = cv2.resize(face, (224, 224))

face = img\_to\_array(face)

face = preprocess\_input(face)

# Add the face and bounding boxes to their respective lists

faces.append(face)

locs.append((startX, startY, endX, endY))

# Only make a predictions if at least one face was detected

if len(faces) > 0:

faces = np.array(faces, dtype="float32")

preds = maskNet.predict(faces, batch\_size=32)

return (locs, preds)

# Function to play the alert sound

def play\_alert\_sound():

playsound("Wearthemask.mp3")

time.sleep(2)

# Load the face detector model

prototxtPath = r"deploy.protext"

weightsPath = r"res10\_300x300\_ssd\_iter\_140000.caffemodel"

faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# Load the face mask detector model

maskNet = load\_model("mask\_detector.model")

# Initialize the video stream from mobile camera using IP Webcam

print("[INFO] Starting the video stream from mobile camera...")

# Replace "<your\_ip\_address>" and "<port\_number>" with the IP address and port from IP Webcam

vs = VideoStream(src="http://192.168.43.1:8080/video").start()

# Loop over the frames from the video stream

while True:

frame = vs.read()

frame = imutils.resize(frame, width=400)

(locs, preds) = detect\_and\_predict\_mask(frame, faceNet, maskNet)

for (box, pred) in zip(locs, preds):

# Unpack the bounding box and predictions

(startX, startY, endX, endY) = box

(mask, withoutMask) = pred

# Determine the class label and color for the bounding box

label = "Mask" if mask > withoutMask else "No Mask" and play\_alert\_sound()

color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

# Include the probability in the label

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

# Display the label and bounding box rectangle on the output frame

cv2.putText(frame, label, (startX, startY - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.45, color, 2)

cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

# Play alert sound if no mask is detected

if label == "No Mask":

play\_alert\_sound()

# Display the output frame

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

# If the 'q' key was pressed, break from the loop

if key == ord("q"):

break

# Clean up

cv2.destroyAllWindows()

vs.stop()

main.py

import data\_prep

import train\_model

import detecting\_mask

def main():

print("[INFO] Data preparation...")

data\_prep.load\_and\_preprocess\_data()

print("[INFO] Training model...")

train\_model.train\_and\_save\_model()

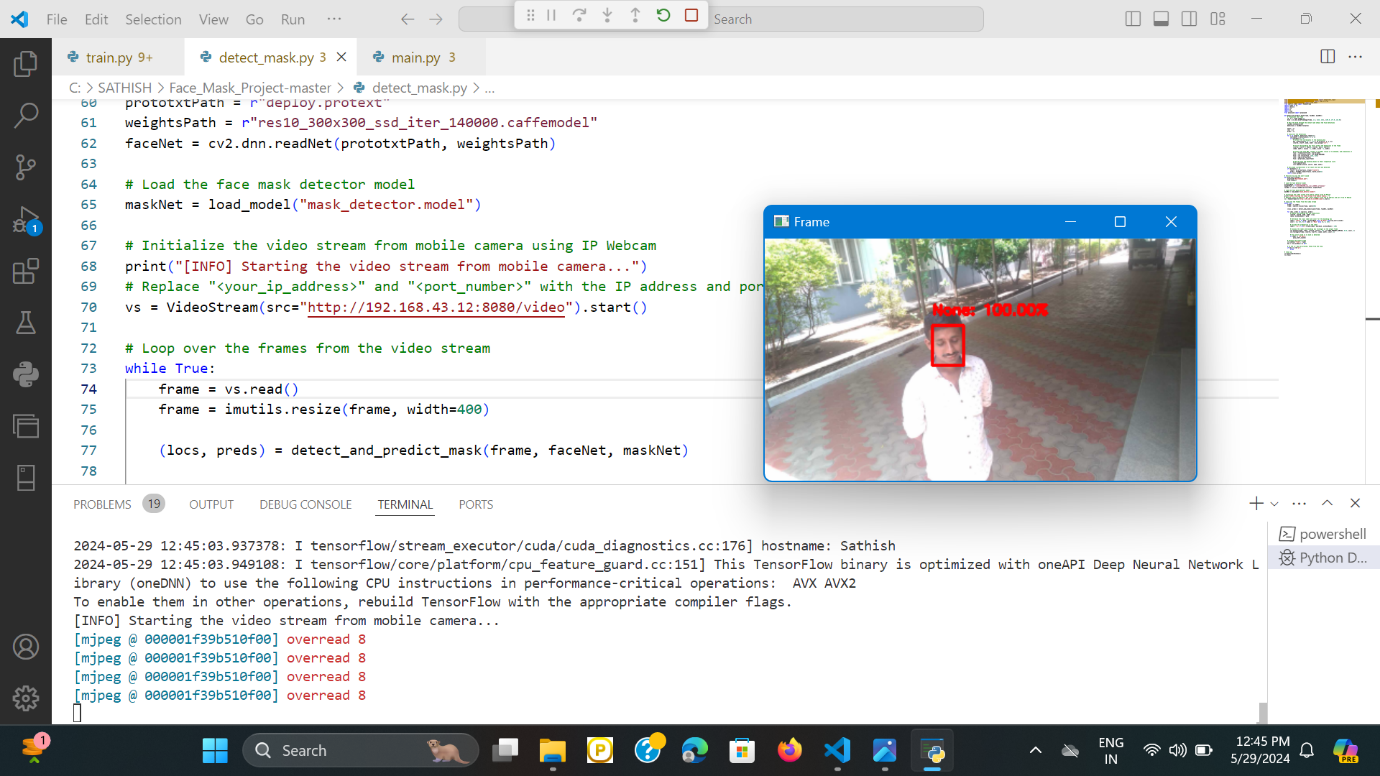
print("[INFO] Detecting masks...")

detecting\_mask.detect\_and\_predict\_mask()

if \_\_name\_\_ == "\_\_main\_\_":

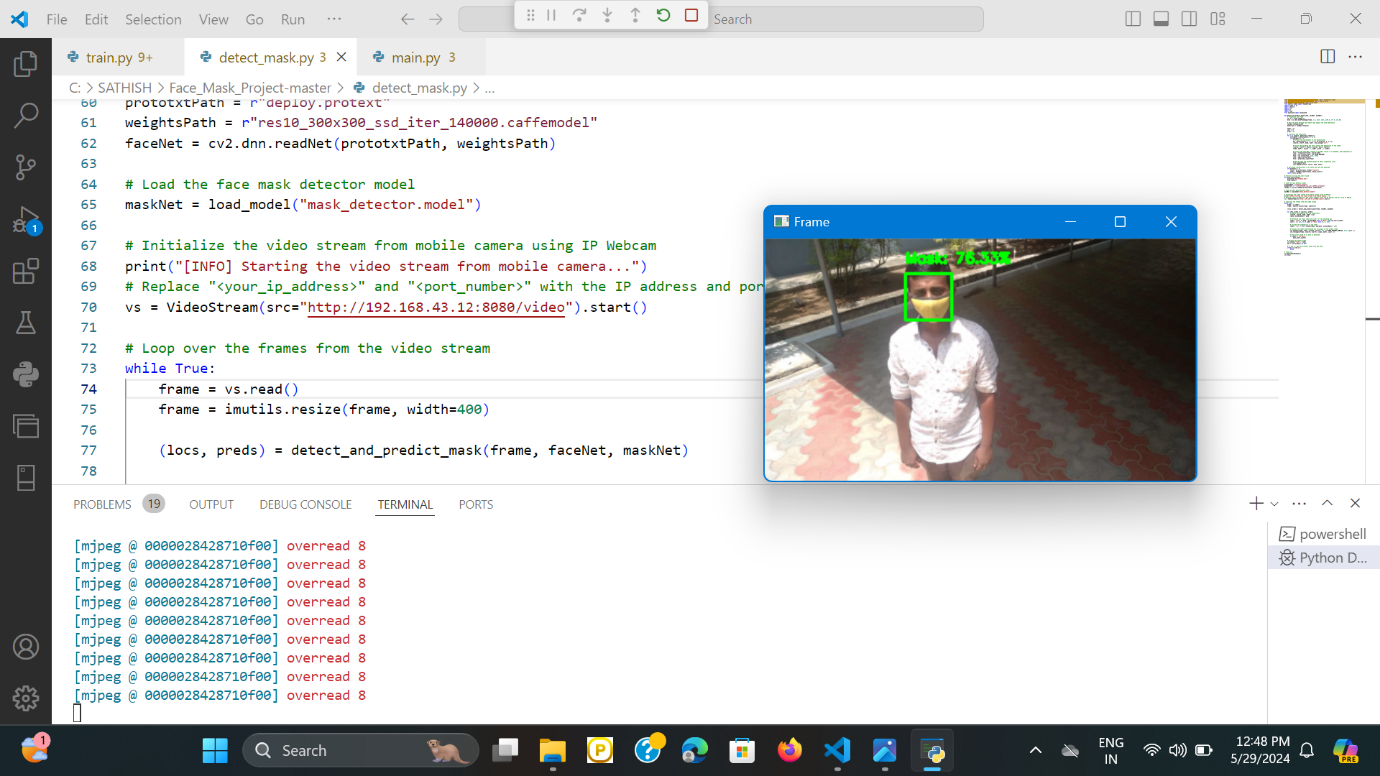
main()

**9.2 SAMPLE OUTPUT**

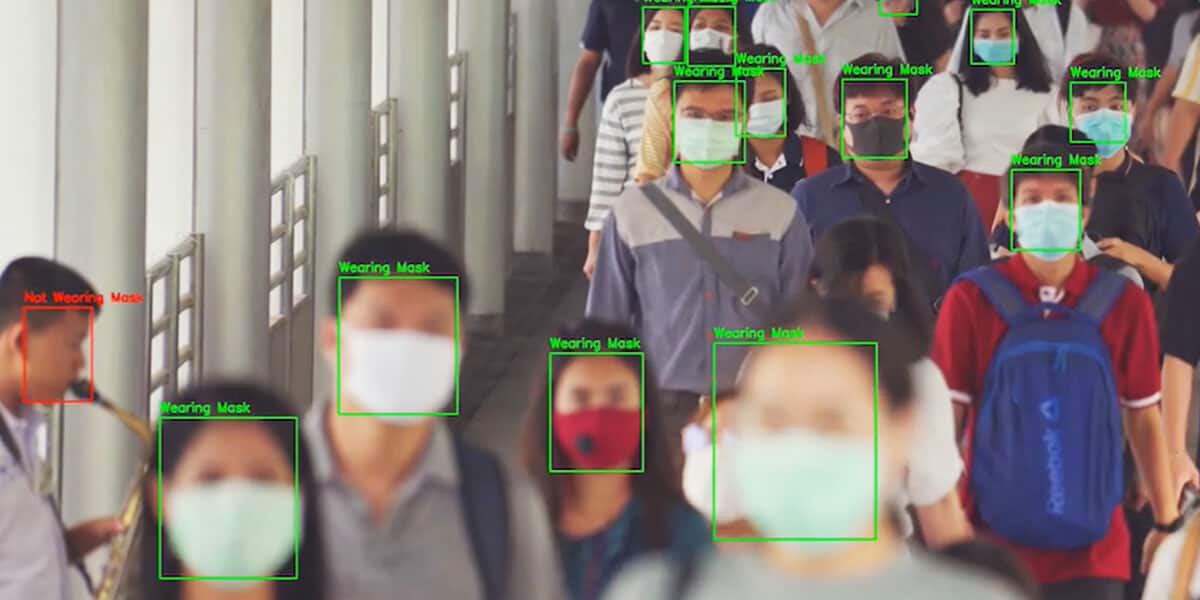
****

Notification Audio is Played

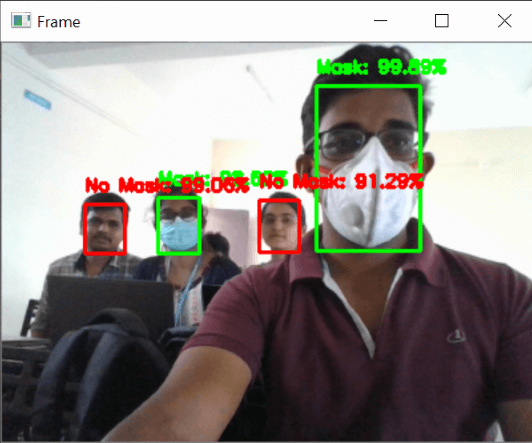
**Fig 9.1 Without Mask**

****

**Fig 9.2 With Mask**

****

**Fig 9.3 Multiple Faces Detection**

****

**Fig 9.4 Detecting the Face in Camera**

**CHAPTER 10**

**REFERENCES**

**BOOK REFERENCES**

* R. Chellappa and A. K. Jain, Handbook of Face Recognition, Springer, 2005.
* L. S. Davis and T. Kanade, Eds., Machine Learning: From Theory to Applications, Springer, 2020.
* M. Turk and A. Pentland, "Eigenfaces for Recognition," Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991.
* R. O. Duda, P. E. Hart, and D. G. Stork, Pattern Classification, John Wiley & Sons, 2001.
* Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, 2016.
* K. Jain, P. J. Flynn, and A. A. Ross, Handbook of Biometrics, Springer, 2008.

**WEB REFERENCES**

* ResearchGate. (https://www.researchgate.net)
* IEEE Xplore. (https://ieeexplore.ieee.org)
* ArXiv. (https://arxiv.org)
* GitHub. (https://github.com)