in collaboration with

Facial Mask Detection System In Python Using Image Processing

23rd August 2022

Coventry University

Department of Ethical Hacking

Submitted To:

MR. MANOJ SHRESTHA

Project Supervisor

Submitted By:

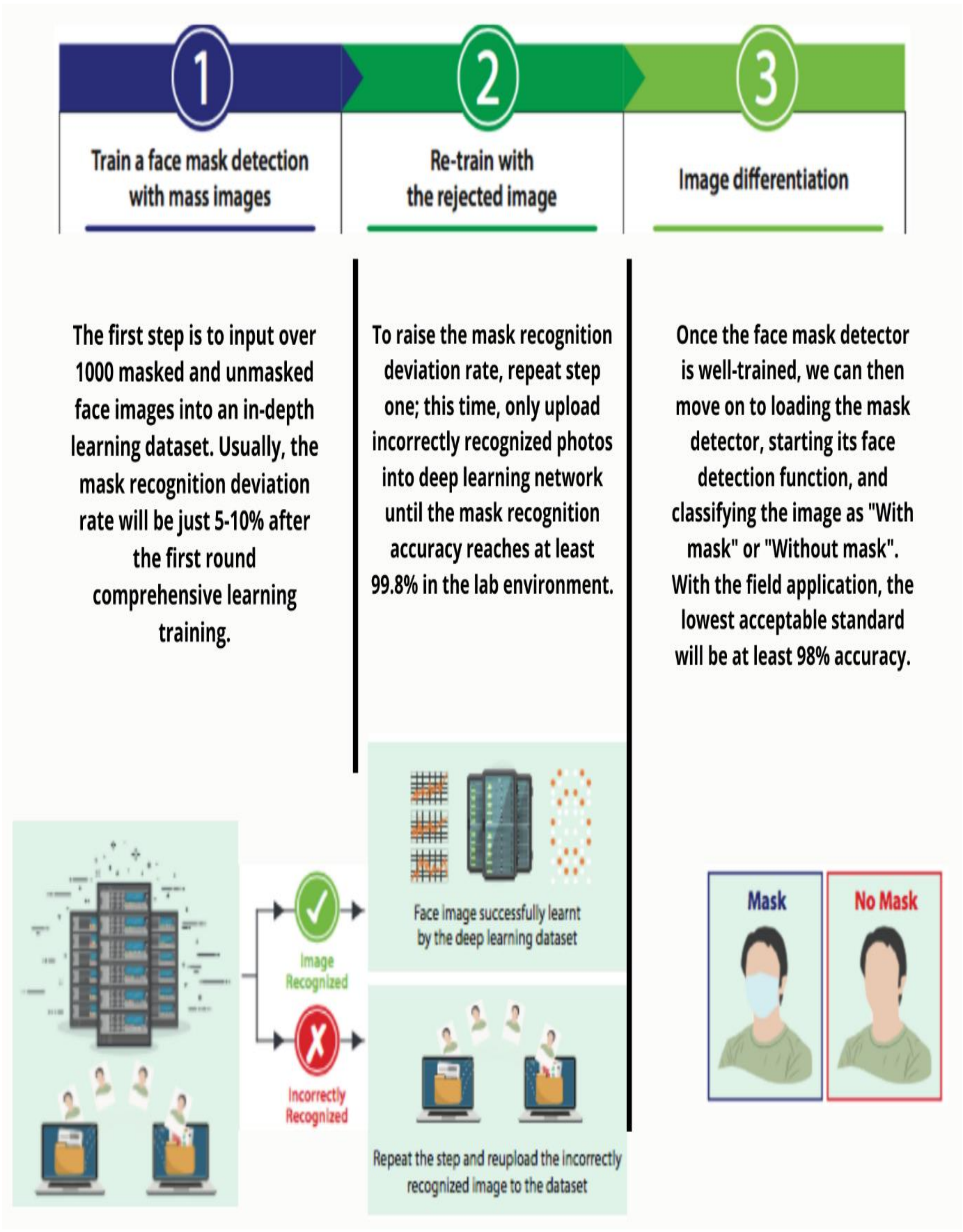
ALISH BHATTA

ID: 10278802

Dillibazar, Kathmandu

info@softwarica.edu.np

softwarica.edu.np



1 Train a face mask detection with mass images

2 Re-train with the rejected image

3 Image differentiation

The first step is to input over 1000 masked and unmasked face images into an in-depth learning dataset. Usually, the mask recognition deviation rate will be just 5-10% after the first round comprehensive learning training.

To raise the mask recognition deviation rate, repeat step one; this time, only upload incorrectly recognized photos into deep learning network until the mask recognition accuracy reaches at least 99.8% in the lab environment.

Once the face mask detector is well-trained, we can then move on to loading the mask detector, starting its face detection function, and classifying the image as "With mask" or "Without mask". With the field application, the lowest acceptable standard will be at least 98% accuracy.

Image Recognized
Incorrectly Recognized
Face image successfully learnt by the deep learning dataset
Repeat the step and reupload the incorrectly recognized image to the dataset
Mask
No Mask

ACKNOWLEDGMENT

I want to thank Coventry University and Softwarica College of IT and Ecommerce for what they have done for me. Also, I'd like to thank my supervisor, Mr. Manoj Shrestha, for his patience, direction, and help. Your wide range of knowledge and careful editing has helped me a lot. I'm very thankful that you accepted me as a student and kept believing in me.

ABSTRACT

In response to the COVID-19 outbreak, the World Health Organization (WHO) made it a biosafety requirement for everyone to wear a face mask. This has made it hard for current facial recognition systems to work, which is why this research was done. This document talks about how a way was made to recognize people from pictures, even if they were wearing a mask. A classification model is used that is based on the architecture of MobileNetV2 and the face detector of OpenCV. Using these steps, it is possible to figure out where the face is and whether or not it is wearing a mask. For instructing the facial recognition models, a set of 2000 observations is made, half of which have a face mask and the other half don't. The test results show that there is a 99.99% chance of being able to tell if someone is wearing a mask or not. Faces of five persons with masks are recognized with 99.25% accuracy, while faces of people without masks are recognized with 99.5% accuracy.

KEYWORDS

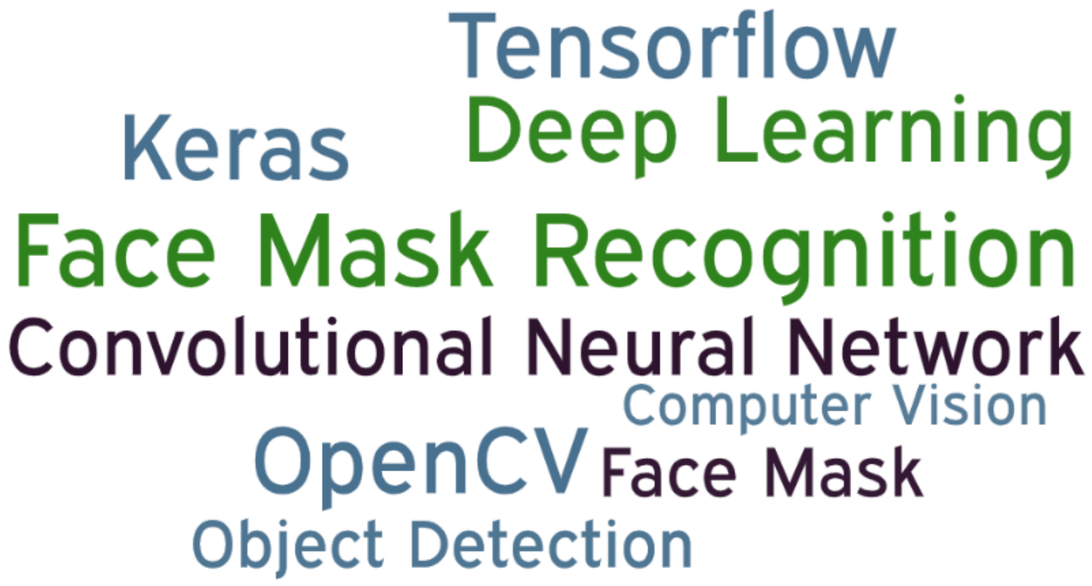


Figure 1: Keywords

Table of Contents

Table of Figures

INTRODUCTION

Object detection is a way for computers to find and locate things in pictures and videos. Object detection can find people, animals, cars, or buildings in digital photos or video frames. It can also find objects that belong to certain classes. When the eyes see a picture, the brain recognizes the objects right away. On the other hand, it takes a long time and a lot of training data for a machine to learn how to recognize images. But new technology in machine learning, deep learning, and computer vision has made these things a lot easier and more straightforward ("AI Detect Objects", 2022). Object detection technology has become very popular in many different fields. It let self-driving cars safely navigate traffic, spot violent behavior in crowded places, keep an eye on objects with video surveillance, recognize faces behind masks with an object detection module, and do a lot of other things ("Object Detection", 2022). And this is just the tip of the iceberg when it comes to what object detection technology can do. Face detection in video streams is a very important part of modern video surveillance systems.

Deep learning algorithms that can recognize faces have only recently been made. Deep learning is a subset of machine learning in AI. It uses algorithms that are based on how the human brain is built and how it works to make computers smarter. Using deep learning, a computer model learns to sort things into groups just by looking at pictures, reading text, or listening to sounds ("Deep Learning", 2022). Deep learning models can get very accurate, sometimes even more accurate than humans. A lot of labeled data and multi-layer neural network architectures are used to train models. Deep learning is a part of data science, which is a field that includes statistics and predictive modeling ("Python in Deep Learning", 2022). Deep learning makes it much faster and easier for data scientists to gather, analyze, and make sense of huge amounts of data. Deep learning is an important part of self-driving cars because it lets them tell the difference between a stop sign and a lamppost. This feature makes it possible for phones, tablets, TVs, and hands-free speakers to be controlled by voice.

Image and video-based detection algorithms made possible by artificial intelligence can accurately find an object and tell if a person is wearing a mask or not. Using deep learning and machine learning techniques, such as support vector machines and decision trees, it is possible to use different datasets to identify a face mask. The main goal of this thesis is to make a model for detecting a face mask. The MobileNetV2 architecture was used to train the model. Two different sets of images were used to train and test the model. Faces were picked out of video streams by using a MobileNetV2 model that had already been trained (Team, 2022). In addition to the OpenCV framework, a number of packages of machine learning, deep learning, and image processing techniques were used. To provide effective monitoring and enable proactive action, the following steps will be taken: data augmentation, loading the classifier, setting up the fully linked layer, pre-processing and loading the dataset, adapting the classifier, training phase, validation, and testing stage.

AIM

The goal of the project “Face Mask Detection” is to create a tool that identifies the image of a human that can calculate the probability that he/she wearing a mask or not.

OBJECTIVES

- To search for different papers and documents which will add support to the project.
- To upload custom-made pictures into the dataset.
- To train the detection system using Keras/TensorFlow.
- To load the mask detector and classify the face with or without a mask.

MOTIVATION

In the past few years, there has been a lot of interest in deep learning, particularly in fields like machine vision, text analytics, object tracking, and other ways of processing information. Convolutional neural network models have been used in most object detection research done in the past. Convolutional neural networks (CNNs) have become more popular in recent years for a variety of tasks, such as identifying pictures, synthesizing speech, tracking objects, and finding the threshold of an image. These tasks are done with deep learning architectures. CNN is very good at pulling out features from images in the above areas. Traditional classification methods are being replaced by CNNs in more and more research methods in order to better capture image information and improve classification performance. Many deep neural networks aren't good for mobile-based facial image classification because their evaluation phase takes a long time and costs a lot of money. To solve this problem, we describe a MobileNet-based facial image classification model that uses a technique called "Depth-wise separable convolution." DSC (Depth-wise separable convolution) was first shown in and is often used for classification tasks in image processing. The Depth-wise separable convolution is like the ordinary convolution, but it uses numbers instead of a single number. Most of the time, convolutions are split into two types: depth-wise and 1x1 point-wise. Instead of applying each filter to all input channels, as in traditional convolution, the Depth-wise convolution layer applies one filtering to one pulse and then uses 1x1 point-wise convolution to combine the Depth-wise convolution results. Depth-wise separable convolution cuts down on the number of parameters that can be learned and the cost of test-and-train computations.

JUSTIFICATION

Wearing masks is a fundamental need nowadays. In public places because of the large volume of people, it becomes tough for security officials to check every person who is not wearing a mask. This model detects whether a person wearing a mask or not. Implementing a similar principle to evaluate a group of people wearing masks is the real challenge.

RESEARCH QUESTION

- How can a Convolutional Neural Network focused on MobileNet be used to sort photos so that facial masks can be found?
- How well does the suggested method work compared to other methods that are similar?
- Is there a lack of transparency and informed consent when using the dataset?

SCOPE

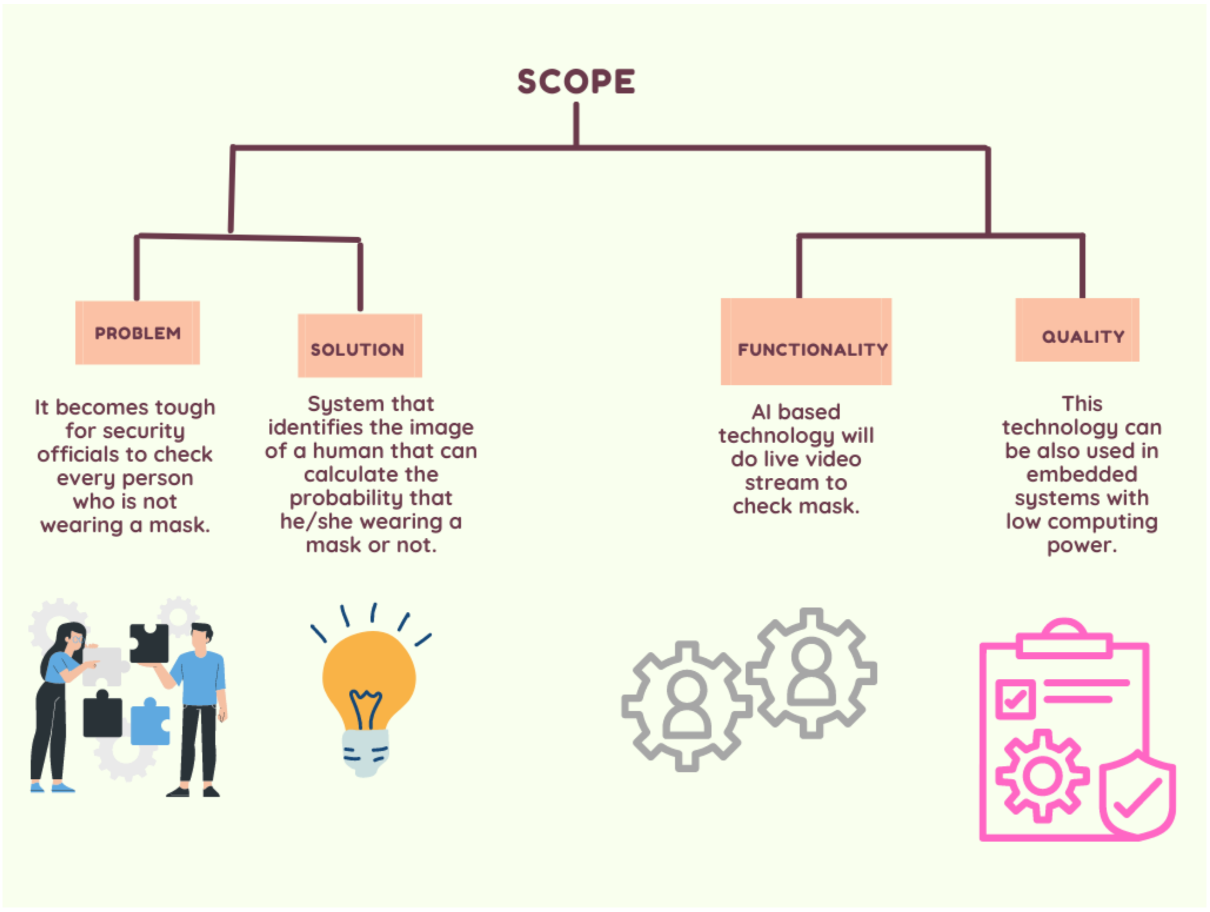

SCOPE
PROBLEM
It becomes tough for security officials to check every person who is not wearing a mask.
SOLUTION
System that identifies the image of a human that can calculate the probability that he/she wearing a mask or not.
FUNCTIONALITY
AI based technology will do live video stream to check mask.
QUALITY
This technology can be also used in embedded systems with low computing power.

Figure 2: Scope of the project

ETHICAL CONSIDERATION

"A science is said to be useful if its development tends to accentuate the existing inequalities in the distribution of wealth, or more directly promotes the destruction of human life." ("G.H. Hardy", 2022). Research in this field often tries to figure out how things work in real life, look into effective treatments, study behaviors, and find other ways to make people's lives better. There are important ethical questions about what we choose to research and how we do it. These things help protect the rights of researchers, improve the quality of research, and keep the integrity of scientific facts.

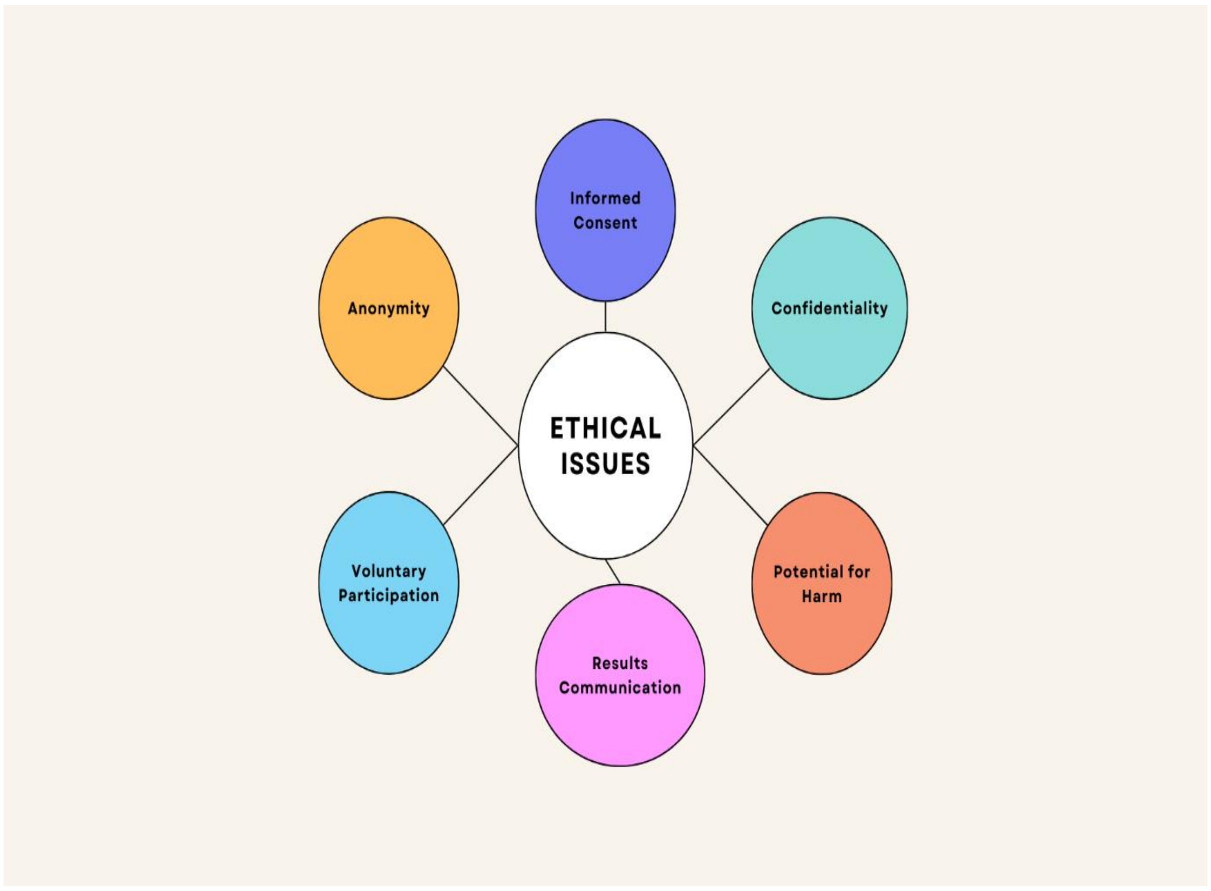


Figure 3: Ethical issues for consideration

This project has informed consent which implies the evaluator informs the participant. Participants must be informed of the project's objective, how the results will be utilized, any possible harmful effects of their involvement, and who will see the findings. Informed permission allows the subject to decide whether to engage in the assessment. If a person feels upset, further information should be supplied. Voluntary participation denotes consent involvement. Participants may resign at any moment without affecting future services, the present program, or ties with researchers or research agencies. It's tough to engage high-risk adolescents in a program, so it's disappointing when they drop out. Participants have the right to exit a program at any moment, therefore no pressure should be imposed on them. This project also causes no harm that may be physical or psychological, causing stress, discomfort, worry, low self-esteem, or privacy violation. Evaluations must not injure (unintentionally or otherwise) subjects. Confidentiality implies only the program coordinator may access identifiable information. Confidentiality also excludes identifiable information from reports and publications. Given the tiny numbers in peer-based programs, it's crucial to evaluate how reports are framed to guarantee no one can be recognized even without names. Anonymity is tougher than confidentiality since the study team doesn't know the participants' identity is harder than confidentiality since social research participants are frequently known to the program organizer. High-risk populations are occasionally utilized as guinea pigs or a captive audience for assessments that are of interest to the program/conductors initiatives but not to the project's participants.

PROJECT RESEARCH PLAN

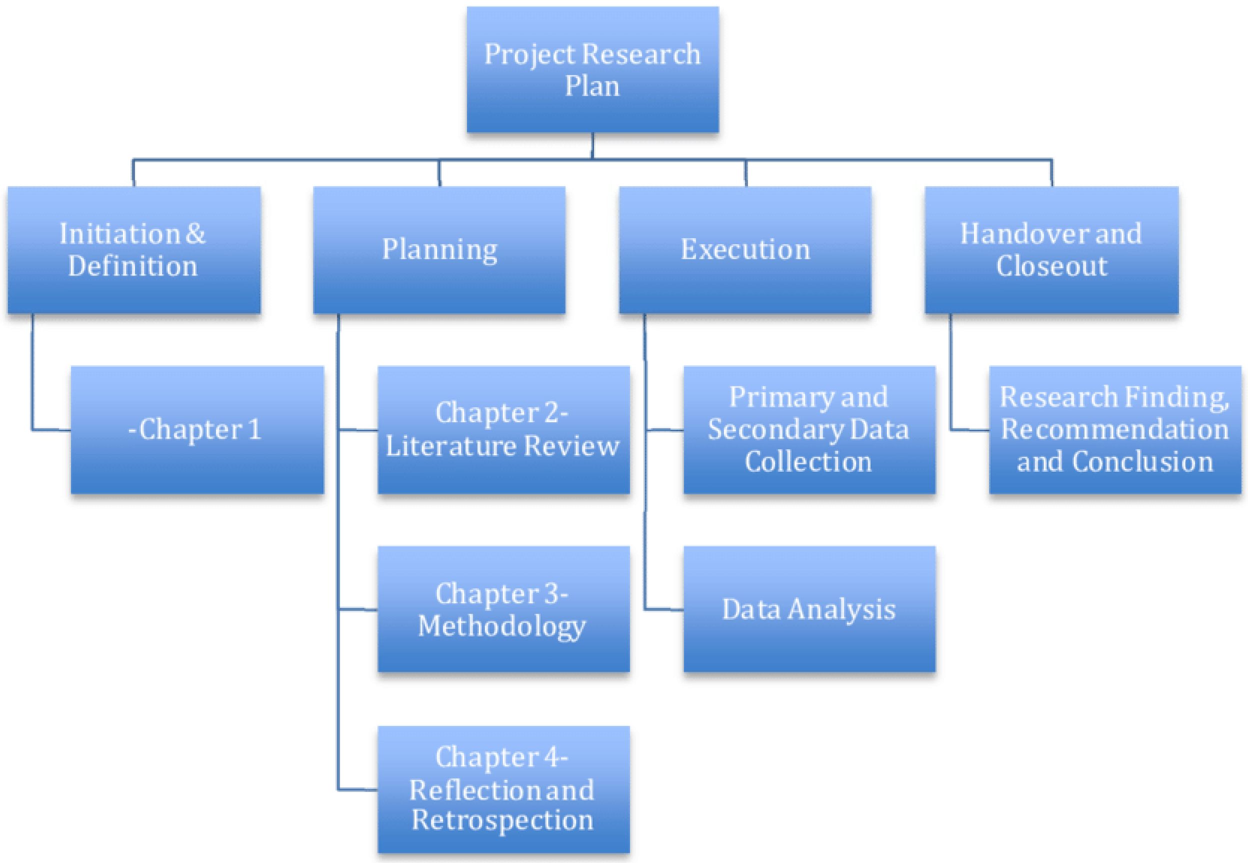

Project Research Plan
Initiation & Definition
Planning
Execution
Handover and Closeout
-Chapter 1
Chapter 2- Literature Review
Chapter 3- Methodology
Chapter 4- Reflection and Retrospection
Primary and Secondary Data Collection
Data Analysis
Research Finding, Recommendation and Conclusion

Figure 4: Plan for Research

Literature Review

KFC "Smile to Pay" in China

In China, the fast-food chain KFC now lets people pay for their fried chicken by scanning their faces. Ant Financial introduced the facial recognition payment method in the Hangzhou branch of KPro, the Chinese version of KFC. This made it the world's first physical shop where customers can make payments with their faces. This is just one way that China is moving toward a cashless economy ("KFC payment China", 2022).

Yum China, which owns KFC in the country, put the system in place for the first time a few years ago at KPRO, a high-end KFC restaurant in Hangzhou. It was a partnership with Alipay that used its Smile to pay system, which had been used to verify transactions on Alipay's mobile payments platform before. At KPRO, the system was put in self-service kiosks so that customers could place orders and pay using Smile to Pay's facial recognition system, which linked their faces to their Alipay accounts. That means they could make a purchase and pay without using a physical payment card or even a smartphone.

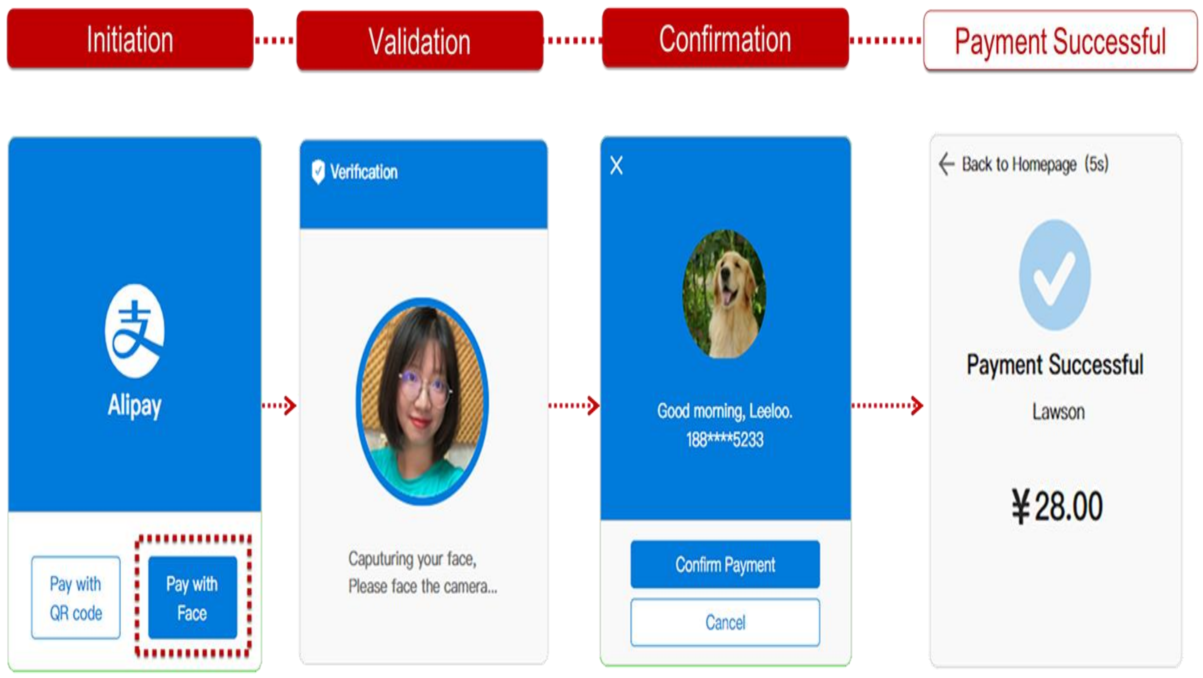
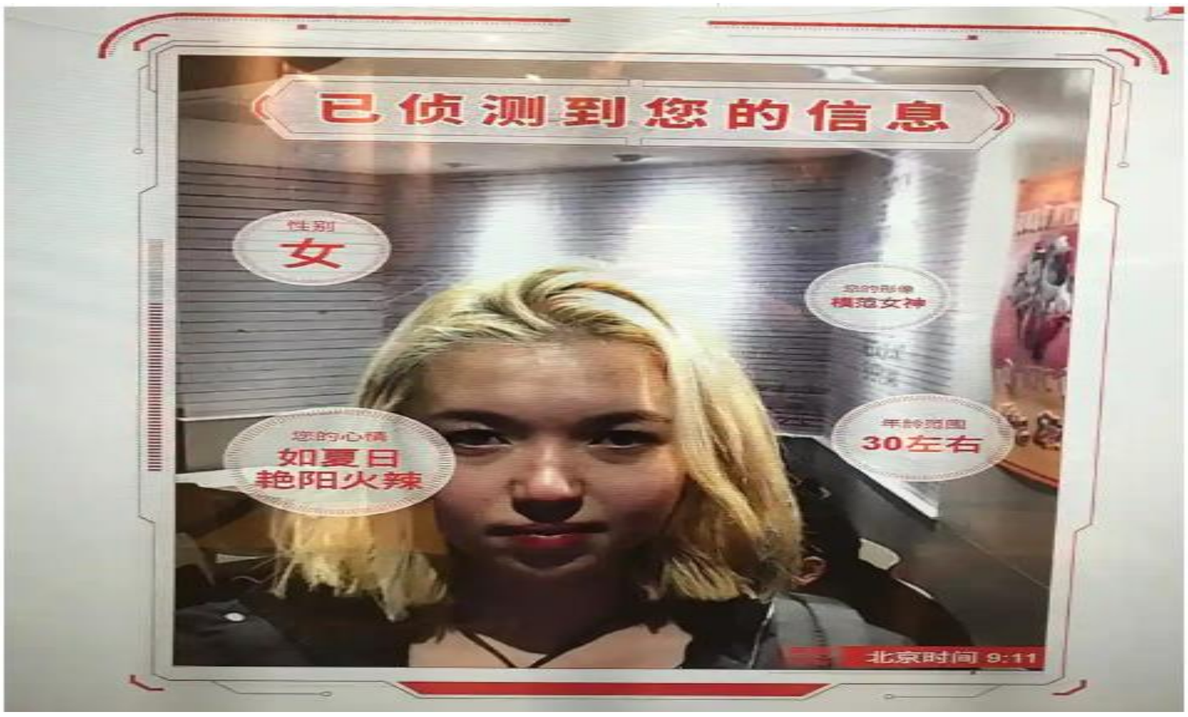


Figure 5: Three steps of FRP in China

The system works by scanning the customer's face against a photo ID that was already in the system. After that, the customer just puts in their phone number, and the payment is approved. Ant Financial lets its users sign in with their faces by using the digital payment platform Alipay. Alipay technology uses a real-time algorithm, a 3D camera, and an analysis of more than 600 facial features to find a match. It helps get people to the point where they don't need their cell phones or wallets anymore.



("KFC China", 2022)

Figure 6: Customer ordering and paying from a machine

Ant Financial has been able to get the software up and running thanks in part to images collected by the country's internet companies and China's database of citizen ID photos, which is something that western societies are still skeptical of because of how it affects people's privacy and security. KFC was one of the first companies in China to use facial recognition technology. Earlier, it made headlines when it teamed up with search engine enterprise Baidu to develop facial recognition technology that can be used to predict an order of the customer. The system uses artificial intelligence to make menu suggestions based on how old and how happy a customer is thought to be. More and more of these naked payment systems are being set up around the world, but they are still in their early stages ("Naked Payments Deployment", 2022). Yum China's system is one of the largest ones in the world right now. Given how often facial recognition technology is used for government surveillance and consumer applications, China may be a good place to sell this technology.

Technology for Event Access during 2019 BRIT Awards

Face recognition technology was used at multiple entrance points to the O2 in London to screen guests at the 2019 Brit Awards. This was done to make the event safer. Digital Barriers, a company that makes surveillance and security technology, worked with Super-Recognizer International (SRI), a corporation that trains people to recognize faces in crowds, to make sure that the Brits were safe ("Tech enhances The BRIT Awards", 2022).

The Digital Barriers facial recognition system was linked to mobile applications, which allowed SRI staff with special training to do additional face-to-face identity checks. When the National Television Awards were held at the O2 in January, the same technology was used. The O2 is the most famous music and entertainment venue in the world, and its security is always being looked at. Facial recognition is proving to be a great way to make the venue, its guests, and its staff safer and more secure. Digital Barriers and SRI have now agreed to work together at any event that needs both facial recognition and operators who know what they're doing. The SRI co-founder and chief operating officer, who is now part of the Digital Barriers team, said that this is the only thing of its kind anywhere in the world.

Figure 7: Entrance at BRIT awards

At The O2, Digital Barriers worked with Super-Recognizers International (SRI), whose qualified staff can find faces in crowds and quickly respond to any matches on the system. Digital Barriers' facial recognition system is linked to mobile apps, which allow for face-to-face secondary verification. The super recognizers were properly trained in this new tech. Digital Barriers also said that Kenny Long, SRI's co-founder and chief operating officer (COO), has joined the company because of this partnership. As a person who knows a lot about identifying people, they've seen every solution out there. They thought Digital Barriers' face recognition technology was the best they had seen, so they were excited to join ("National Television Awards | Event Industry News", 2022). Digital Barriers and SRI have agreed to work together whenever facial recognition and specialist operators are needed. The tests with Digital Barriers went well, and they are committed to making the venue safe and secure for fans, staff, and performers. About the Super Recognizers, International SRI is made up of people who can remember 80% of the faces they see, which is a skill that only 1-2% of the general population is thought to have. People who work at SRI come from many different backgrounds, such as law enforcement, the military, and more. They use their skills to look at CCTV footage to help with both private and police investigations. Most recently, they helped the police with the Salisbury Novichok case.

The CEO of Digital Barriers said that the effective O2 deployments and the fact that Long joined the team "show that our technology is world-class." They work with governments all over the world, and technology has been tested in the toughest environments. Face recognition technology is being used more and more in live entertainment. For example, Taylor Swift uses biometric technology to check concert crowds for known criminals and other people who shouldn't be there.

Amazon Rekognition

Amazon Rekognition makes it simple for our apps to do image and video analysis. The Amazon Rekognition API only needs a picture or video of an object, person, text, scene, or activity to figure out what it is. It can also find any content that isn't appropriate. Amazon Rekognition can also analyze, compare, and search for faces with a high level of accuracy. Faces can be found, analyzed, and compared for many different purposes, such as verification, listing, counting people, and community safety ("Rekognition", 2022).

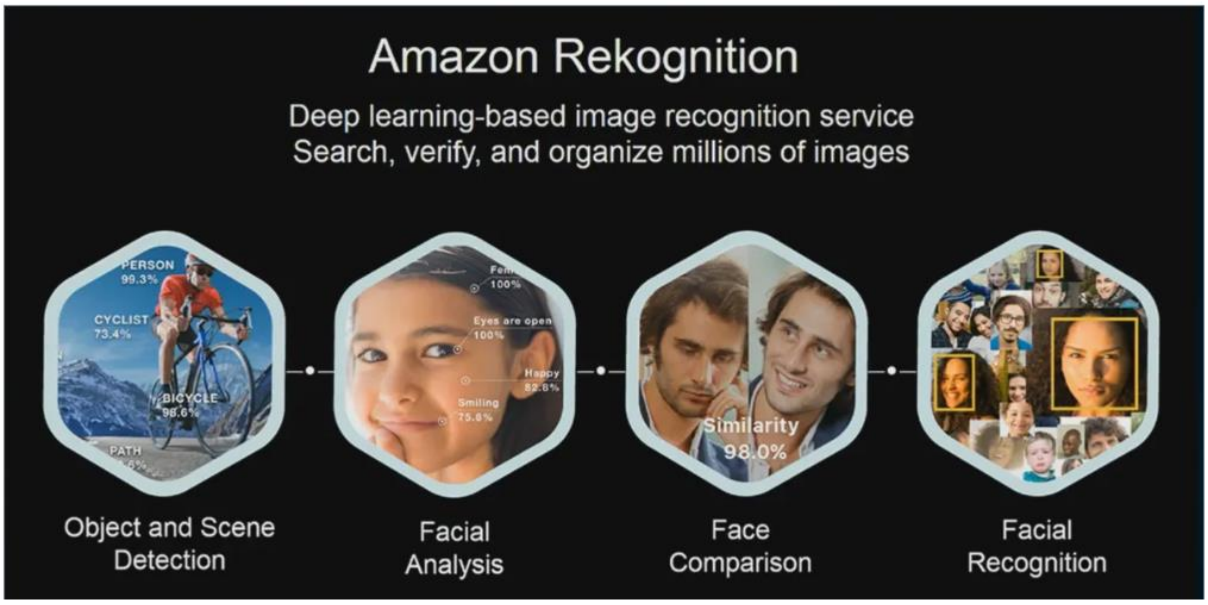


Figure 8: Amazon image recognition

(YouTube, 2022)

Amazon Rekognition is based on using the same deep learning technology that Amazon's machine vision scientists used to analyze billions of videos and pictures every day. This technology has been tested and is highly scalable. To use it, we don't need to know anything about machine learning. Amazon Rekognition has a simple, easy-to-use API that can quickly look at any image or video file in Amazon S3 and figure out what it is. Amazon Rekognition is always studying new data, and the service is always getting new labels and ways to compare faces.

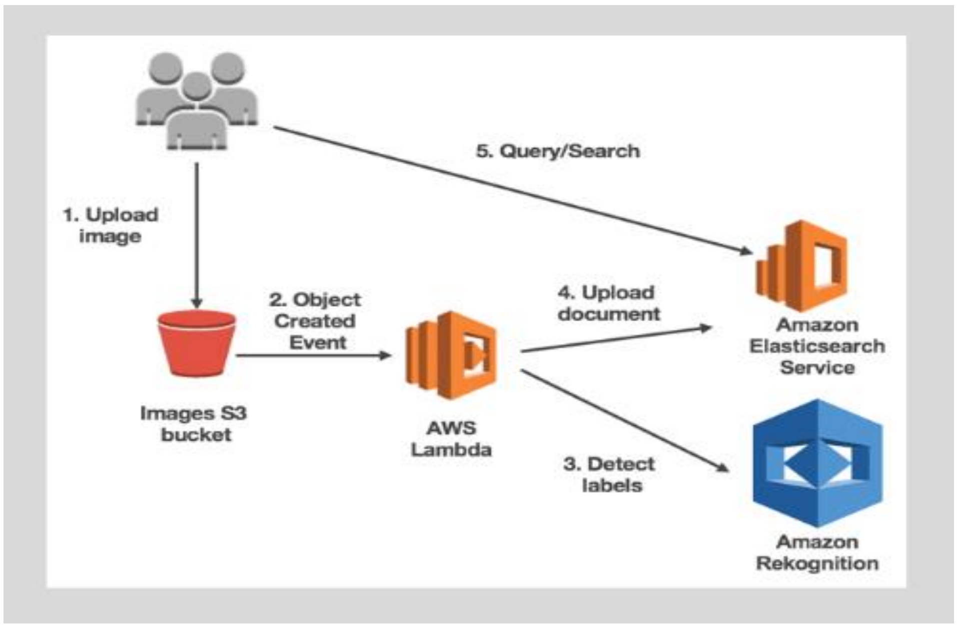


Figure 9: Amazon Rekognition build Photo Recognition System

Some common ways for using Amazon Rekognition are as follows:

- **Image and video libraries that can be searched:** Amazon Rekognition makes videos and images that have been saved searchable, so we can find things and scenes that appear in them.
- **Face-based user verification:** Amazon Rekognition lets our apps compare a user's live image to a reference image to make sure they are who they say they are ("Machine Learning Image and Video Analysis-AWS", 2022).
- **Personal Protective Equipment Detection:** Amazon Rekognition can tell when a person in an image is wearing Personal Protective Equipment (PPE), like a mask, a cap, or gloves. PPE detection can be used when safety is the most important thing. Industries like construction, manufacturing, health - care, food production, logistics, and retail are good examples. With PPE detection, you can tell if a person is wearing a certain type of PPE without having to look at them. We can use the results of the detection to send a message or find places where safety precautions or preparation could be better.
- **Sentiment analysis and demographics**: Amazon Rekognition can tell from facial images whether a person is happy, sad, or surprised. It can also tell things like their gender. Amazon Rekognition can look at pictures and send information about how they make people feel and who they are to Amazon Redshift so that it can report on trends like where stores are located and similar things. Keep in mind that a person's emotional state can only be guessed based on how their face looks. It doesn't show how a person is feeling on the inside, and Rekognition shouldn't be used to figure that out.
- **Face Search:** With Amazon Rekognition, we can look through images, videos we've saved, and live videos for faces that match those in a container called a "face collection." A face collection is a list of the faces you own and are in charge of. For

Amazon Rekognition to be able to find people based on their faces, they have to do two main things:

- **Unsafe content detection**: Amazon Rekognition can find images and videos that have adult or violent content. Developers can use the metadata that is returned to filter out content that doesn't fit their business needs. The API doesn't just mark an image as unsafe if it has harmful content; it also gives back a list of labels with confidence scores. These labels show specific types of unsafe content, which makes it possible to filter and manage large amounts of user-generated content in a more precise way (UGC). Some examples are social and dating sites, platforms for sharing photos, blogs and forums, apps for kids, e-commerce sites, entertainment sites, and services for advertising online.
- **Celebrity recognition:** Amazon Rekognition can find celebrities in photos and videos that people send it. Amazon Rekognition can recognize thousands of famous people from different fields, like politics, athletics, industry, entertainment, and the media.

NVidia FaceNet Model

This card talks about a model that can find one or more faces in an image or video. Compared to the FaceirNet model, this one works better with RGB images and faces that aren't as big. The model is based on an NVIDIA DetectNet v2 detector and a feature extractor called ResNet18. This architecture, which is also called Grid Box object detection, uses bounding-box regression on an image grid to find objects. The Grid box system divides an input image into a grid that predicts four normalized bounding-box parameters and a confidence value for each output class. ("NVIDIA NGC", 2022)

A clustering algorithm like DBSCAN or NMS must be used to process the raw normalized bounding-box and confidence detections to get the final bounding-box coordinates and category labels. The DetectNet v2 entry point in TAO was used to train this model. The training algorithm improves the network so that objects can be found and their confidence in their location doesn't drop as much. There are two parts to the training. In the first step, regularization is used to train the network so that pruning can be done more easily. After the first phase, they prune the network by removing channels whose kernel norms are below the pruning threshold. In the second step, the network that has been cut down is retrained. During the second phase, there will be no regularization. FaceNet v2.0 was trained on a set of more than 1.8 million faces that were kept secret. The training dataset is made up of pictures taken by cameras at different heights and angles, with different fields of view (FOV), and with and without occlusions. Human labelers put labels on the ground-truth bounding boxes and categories. This makes the training dataset. When labeling the training data for NVIDIA FaceNet, the following rules were used.

- Bounding boxes around faces should be as tight as possible.
- Put an occlusion level from 0 to 9 on each face's bounding box. 0 means you can see the whole face, and 9 means you can't see 90% or more of the face. Only faces with an occlusion level of 0–5 are used for training.

- If faces are at the edge of the frame and can't be seen more than 60% of the time because the image is cut off, this image is taken out of the dataset.

FaceNet v1.0's ability to conclude was tested with 8018 proprietary images taken in a variety of environments, with and without occlusion, at different camera heights, and from different angles. The inference is run at INT8 precision on the model that has been cut down that was given. Max-N configuration is used on the Jet-son devices to get the most out of the GPU frequency ("TAO Toolkit", 2022). The only performance shown here is based on inference. End-to-end efficiency with video streaming data could be a little different depending on where other hardware and software backlogs are presented.

Limitations

- Very little heads

The NVIDIA FaceNet model doesn't do a good job of finding small faces. In general, a face is small if it takes up less than 10% of the image area.

- Faces Hidden

FaceNet may not be able to find faces that are hidden or cut off so that less than 20% of the face is visible.

- Images that are bent and blurry

The FaceNet models were not taught how to use cameras with fish-eye lenses or cameras that move. So, the models might not work well for images that are bent or blurred by motion or something else.

NVIDIA FaceNet model detects faces. But you can't guess anything else about the faces, like their race, gender, or type of skin. With NVIDIA's platforms and application frameworks, developers can make many different AI apps. Think about the possibility of algorithmic bias when choosing or making the models that will be used. Work with the model's creator to make sure it meets the needs of the relevant industry and use case, that it comes with the instructions and documentation needed to understand error rates, confidence intervals, and results, and that it is being used as intended.

TOOLS AND TECHNOLOGIES

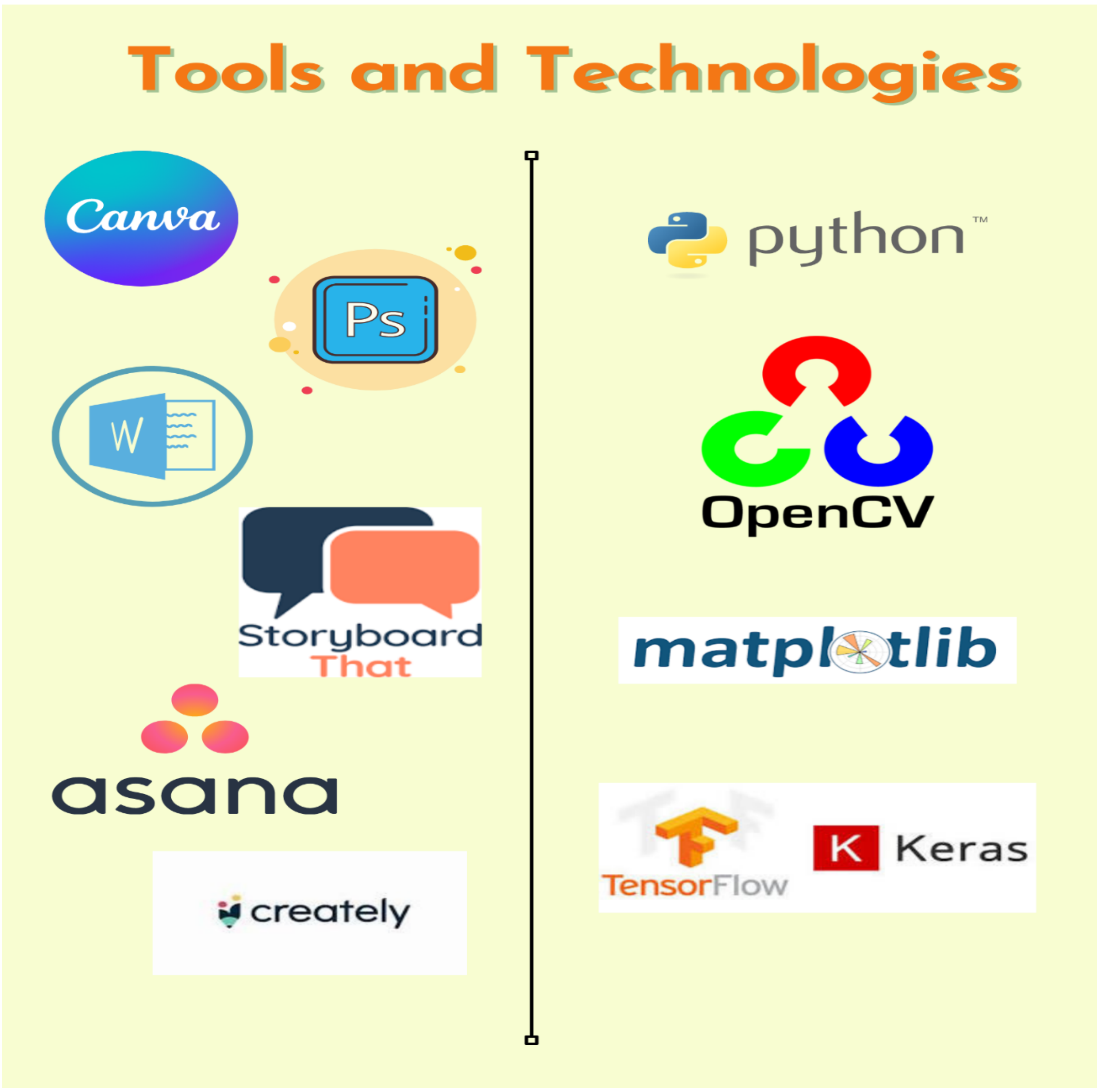


Figure 10: Tools and technologies used

Python

Python is an advanced programming language that can be used for many different tasks. Python is a programming language that can be used to make desktop graphical user interface (GUI) apps, websites, web apps, math programs, system scripts, and so much more. It has a dynamic type of system and a memory management system that works on its own. It has a large, well-rounded library and works with different ways of programming, like object-oriented, imperative, functional, and procedural. Python is mostly a programming language with dynamic types that came out in 1991. It was made by Guido van Rossum, and then the Python Software Foundation worked on it. It has a simple syntax that lets programmers express ideas in fewer lines of code. It was made mainly with how easy it is to read the code in mind ("Python Basics - GeeksforGeeks", 2022).

As a programming language, Python has a lot of good points. Since Python is easy to learn, anyone can learn how to write code in a short amount of time. Compared to Java, C, C++, and C#, Python is one of the easiest object-oriented programming languages to learn. It is an open-source programming language, which means that anyone can work on it and add to it. There is an online forum for Python where tens of thousands of programmers meet every day to improve the language. Python is also free to download and use on any operating system, including Windows, Mac OS X, and Linux. Python is one of the most popular programming languages because it has a lot of graphical user interfaces that are easy to add to the interpreter ("GeeksforGeeks", 2022). It knows about class and objects encapsulation, which makes it possible for applications to get better over time. Python comes with a lot of built-in libraries that can be used right away in a program by importing them.

TensorFlow

TensorFlow is open-source software for machine learning that focuses on deep neural networks from the very beginning to the very end. TensorFlow is a group of libraries, tools, and community resources that cover a wide range of topics. It lets programmers build and use cutting-edge applications based on machine learning ("TensorFlow Intro", 2022). The TensorFlow python deep-learning library was first made by the Google Brain team for use within Google. Since then, the number of R&D and manufacturing systems that use the open-source platform has grown. TensorFlow is based on a few key ideas. Tensors are the building blocks of TensorFlow. In the TensorFlow deep-learning framework for Python, a "tensor" is an array that can hold many different kinds of data. A tensor can have n dimensions, while a vector, an array, or a matrix can only have one or two. Dimensions are shown by the shape. A tensor with one dimension is a vector. A tensor with two dimensions is a matrix. A tensor with no dimensions is a scalar.

Keras

Google made Keras, which is an API for building neural networks that uses deep learning at a high level. It is written in the language Python and helps with the development of neural networks. It is quick and easy to put together and use. It was made by Francois Chollet, a Google developer. Keras doesn't take care of low-level computation. It instead uses a library called the "Backend." With Keras, you can switch between different back ends. Keras works with a number of frameworks, such as TensorFlow, Theano, PlaidML, MXNet, and CNTK (Microsoft Cognitive Toolkit). The only one of these five frameworks to accept Keras as its official high-level API is TensorFlow. Keras is a deep learning framework that is built on top of TensorFlow and has modules built in for all neural network computations (Team, 2022). TensorFlow's core API can be used at the same time to build custom computations with tensors, computation graphs, sessions, and so on. It gives users full control and flexibility over their apps and makes it easy to put ideas into action quickly.

OpenCV

OpenCV is a large open-source library for computer vision, image processing, and machine learning. OpenCV works with a number of programming languages, including Python, C++, Java, and others. It can tell what things are, what people look like, and even what people write by looking at an efficient library for numerical operations. One of the goals of OpenCV is to give people an easy-to-use computer vision infrastructure that lets them make complex vision applications quickly ("OpenCV-Py", 2022). The OpenCV library has more than 500 functions that cover a wide range of vision topics, such as factory product inspection, medical imaging, security, user interface, camera calibration, stereo vision, and robotics. Since computer vision and machine learning are often used together, OpenCV also has a complete library for machine learning ("Libraries", 2022).

NumPy

NumPy is the most important scientific computing package for Python. It is a library that has a multidimensional array object, derived objects (like masked arrays and matrices), and a number of fast array operations, such as mathematical, logical, shape manipulation, sorting, selecting, basic linear algebra, basic statistical operations, random simulation, and more (NumPy? 2022). NumPy works with a lot of other well-known Python packages, like pandas and matplotlib.

On the other hand, SciPy is an important free and open-source Python library for scientific and technical computing. It is a set of mathematical algorithms and utility functions based on the numpy extension for Python ("SciPy in Python", 2022). It gives the user a lot of power by giving them high-level commands and classes for manipulating and displaying data in an interactive Python session. SciPy is built on NumPy, so developers don't have to import NumPy if they've already imported SciPy.

Imutils and Matplotlib

Imutils is a set of functions that make basic image processing tasks like translating, rotating and scaling, skeletonizing, and showing matplotlib pictures easier to do in OpenCV, Python 2.7, and Python 3 ("imutils", 2022). Matplotlib is a very important library for 2D array plots in Python. Matplotlib is a package for plotting data that works on multiple platforms. It is based on NumPy arrays and is meant to work with the SciPy stack. It was first used by John Hunter in 2022. One of the best things about visualization is that it makes it easy to see large amounts of data in simple pictures. There are many types of plots in Matplotlib, such as line, bar, scatter, histogram, and so on ("plot lib Tutorial", 2022). It is a package that works on multiple platforms and gives you a variety of tools for making things in Python from data stored in lists or arrays. It is one of the most powerful libraries in Python.

METHODOLOGY

Waterfall Software Development Life Cycle is the mechanism that will be used to build this project (WF-SDLC). Since this is a system that brings together surveillance systems and facial expression systems to make a real-time surveillance system, we saw the need to add collaborative methods to the WF-SDLC because the WFSDLC can't work by itself in a collaborative system. So, the Collaborative Waterfall Software Development Life Cycle will be used to make the system (C-WFSDLC). This system was made so that it can tell if someone is wearing a mask on their face. This will help the guards know if someone is wearing a mask or not. The C-WF-SDLC methodology has six main steps: planning, analyzing, designing, putting the plan into action, testing, and maintaining.

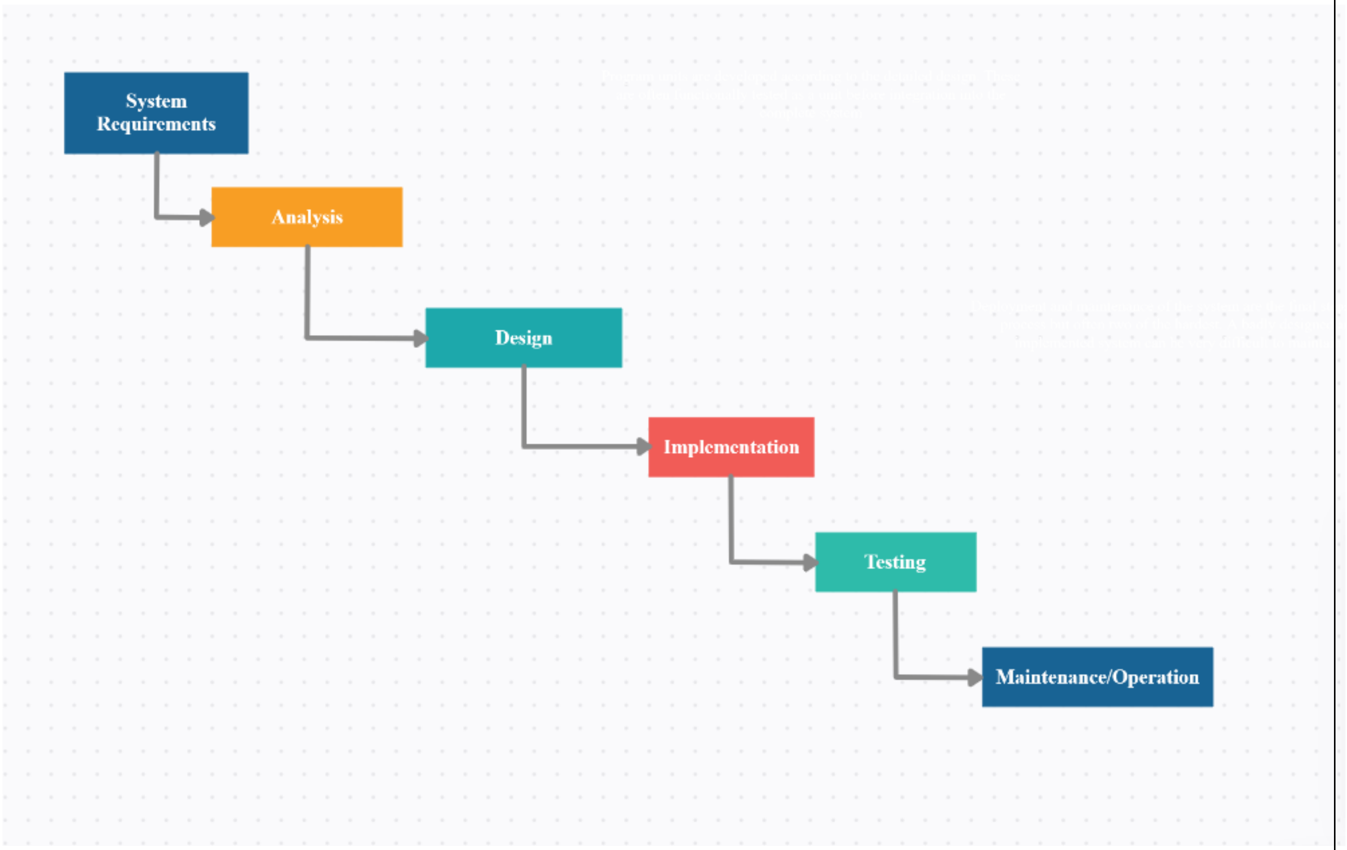


Figure 11: Waterfall Methodology

This project is about making a Face Mask Detection model. A model has been made to figure out if someone is wearing a mask. In this system, the computer's main camera was used, and the video was sent to the model that was being used. This system is built with the help of OpenCV libraries and images that are given to the system as it learns. The MobileNet algorithm was used to find the system.

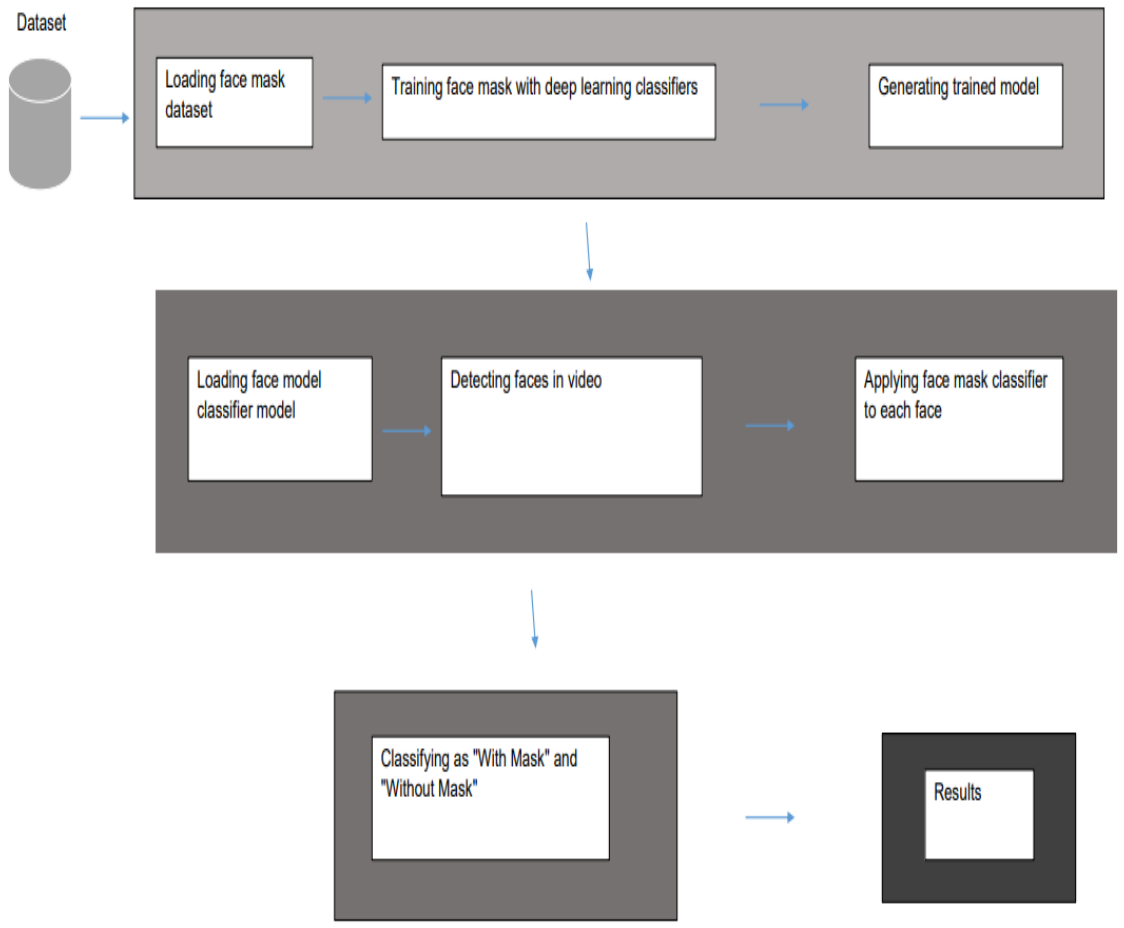

Dataset
Loading face mask dataset
Training face mask with deep learning classifiers
Generating trained model
Loading face model classifier model
Detecting faces in video
Applying face mask classifier to each face
Classifying as "With Mask" and "Without Mask"
Results

Figure 12: System design

Dataset preparation

Two sets of images were taken from the Kaggle website and used to train and test the model ("Kaggle", 2022). The image dataset was then divided into two groups: with mask and without mask. With these datasets, it is possible to make a model that can tell the difference between people who are wearing masks and those who are not. The "with mask" and "without mask" folders each have about 1,000 images in these datasets. Images with and without masks can be seen below.

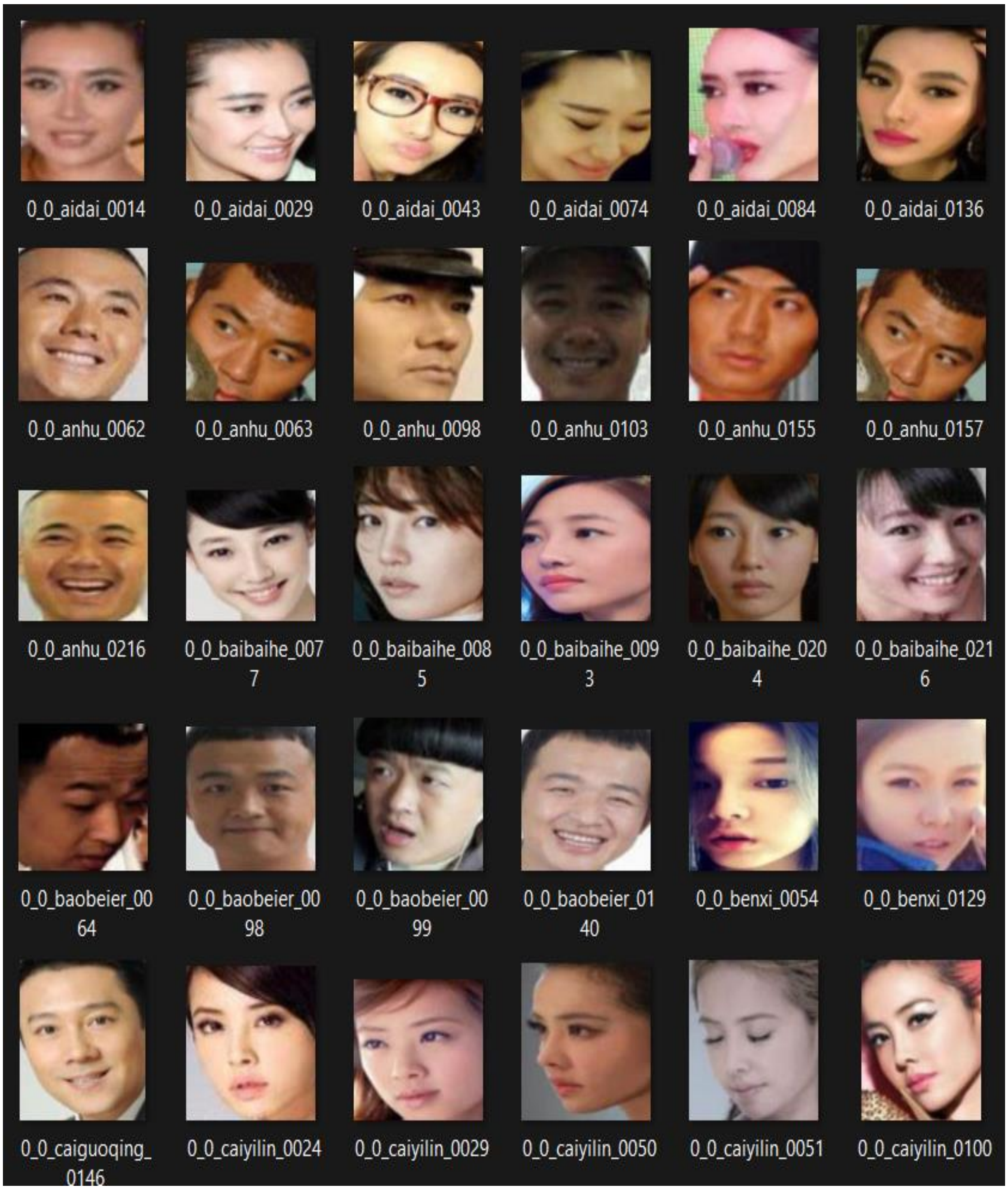


Figure 13: Without mask images

Faces of different kinds of people are shown above the image figure. These pictures were taken from the website of Kaggle. The goal of model training was met by using these pictures.

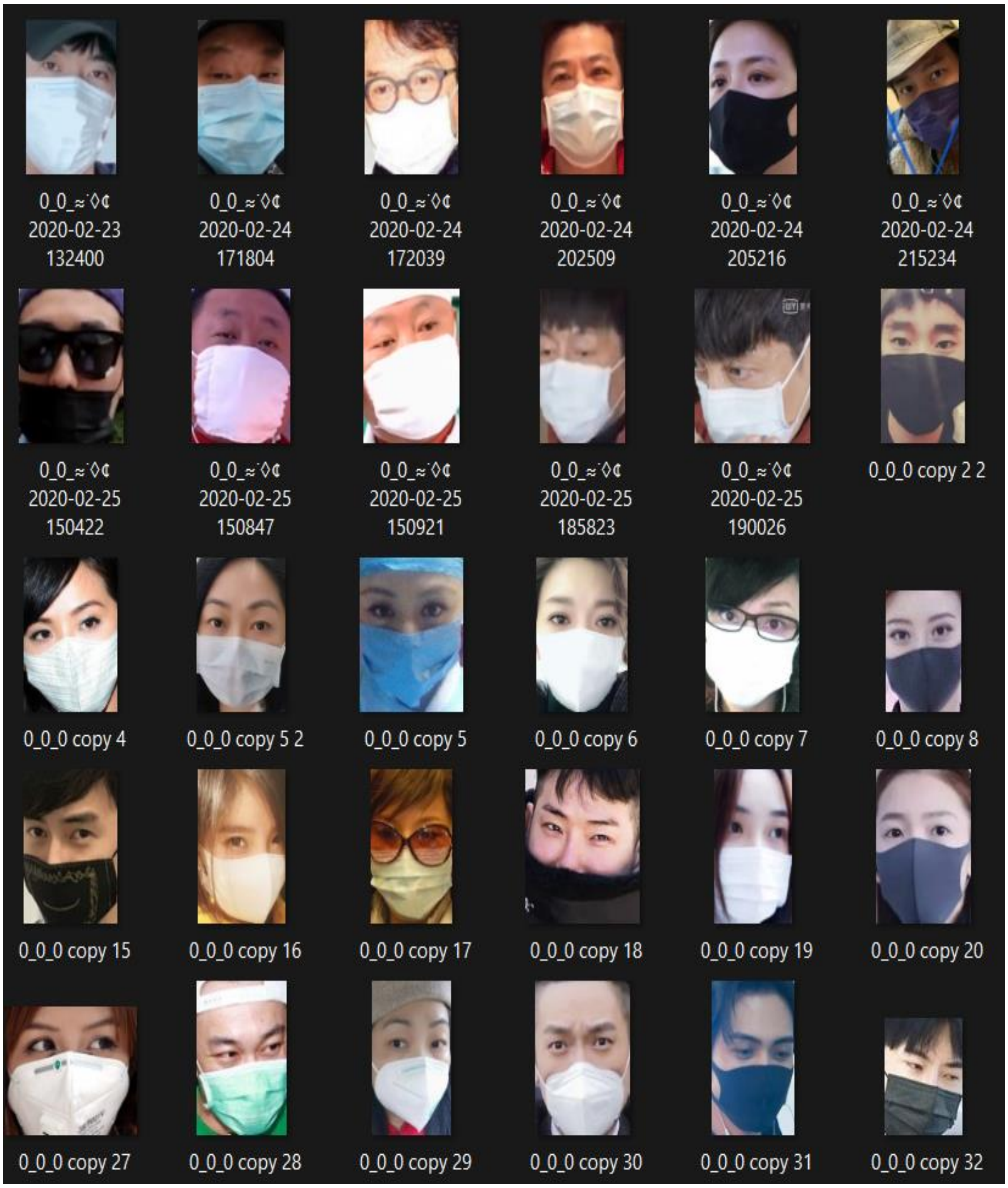
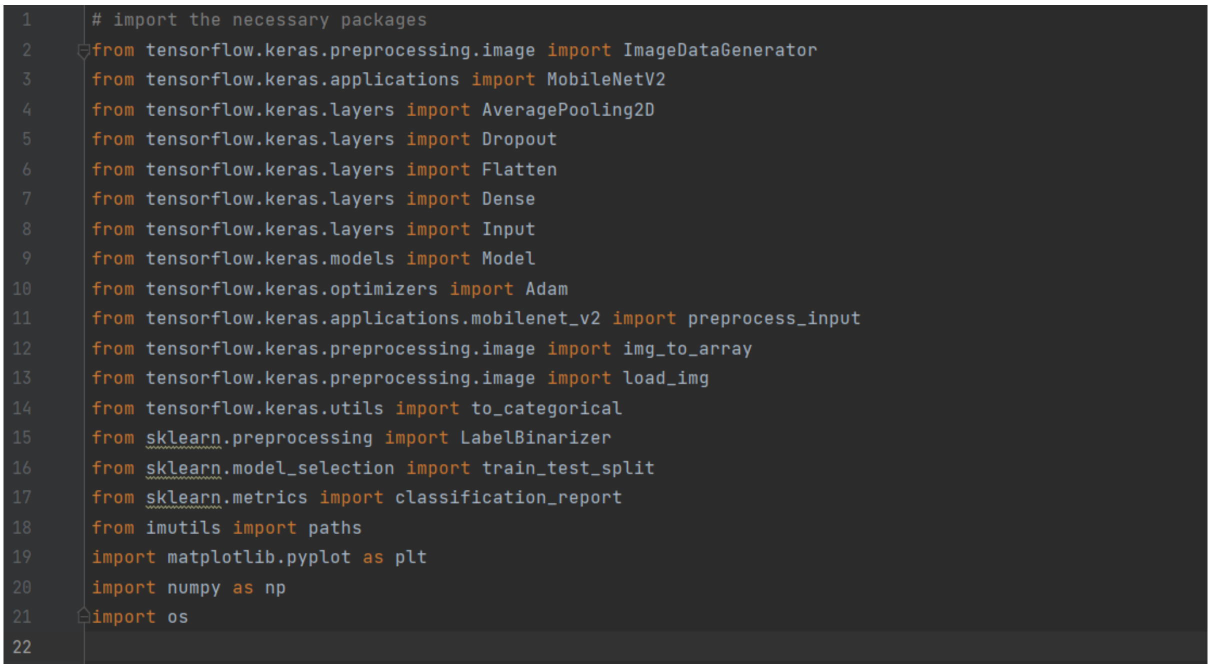


Figure 14: With mask images

There are also images with masks in the datasets. These were also taken from the website Kaggle. For data training, different people and places took pictures from different points of view.

Importing required libraries

Machine learning and deep learning, as well as the programming language Python, were used to figure out what the face masks were. Several needed libraries were put in place in the project path. The command prompt was used to install the libraries. After the installation, packages such as tensorflow, numpy, imutils, keras, opencv, scipy, and matplotlib were brought in. The functions of these packages are taken care of by themselves as needed by the application.

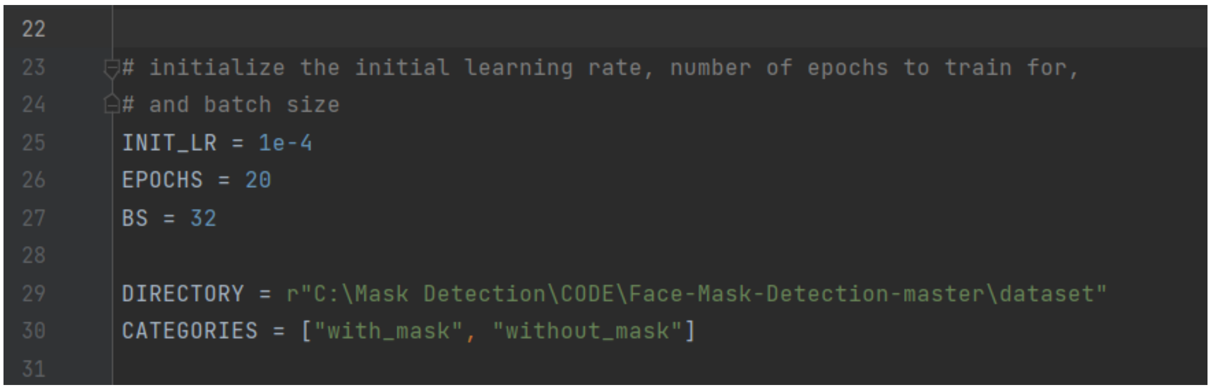


```python
# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
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
```

Figure 15: Importing libraries

Training CNN model

To start, tensorflow, numpy, Imutils, Keras, OpenCV, SciPy, and matplotlib, which were needed for the implementation process, were imported. The import method was also used to bring in data sets. The directory for the model has been made, and the path to the folder with the datasets has been put in the directory. There is a type of thing in this area. There are two values in the category. One has a mask, and the other does not. A loop function has been used with these two categories. It also shows that the program started with the initial learning rate and then switched to a slower learning rate. The loss has been correctly estimated because the learning rate has been used less than the loss. So, there was a high level of accuracy.

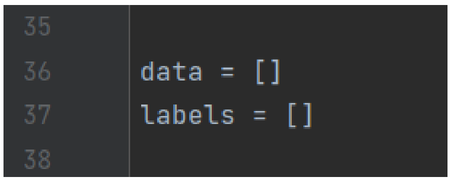


```
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32

DIRECTORY = r"C:\Mask Detection\CODE\Face-Mask-Detection-master\dataset"
CATEGORIES = ["with_mask", "without_mask"]
```

Figure 16: Path, Epochs and Batch size

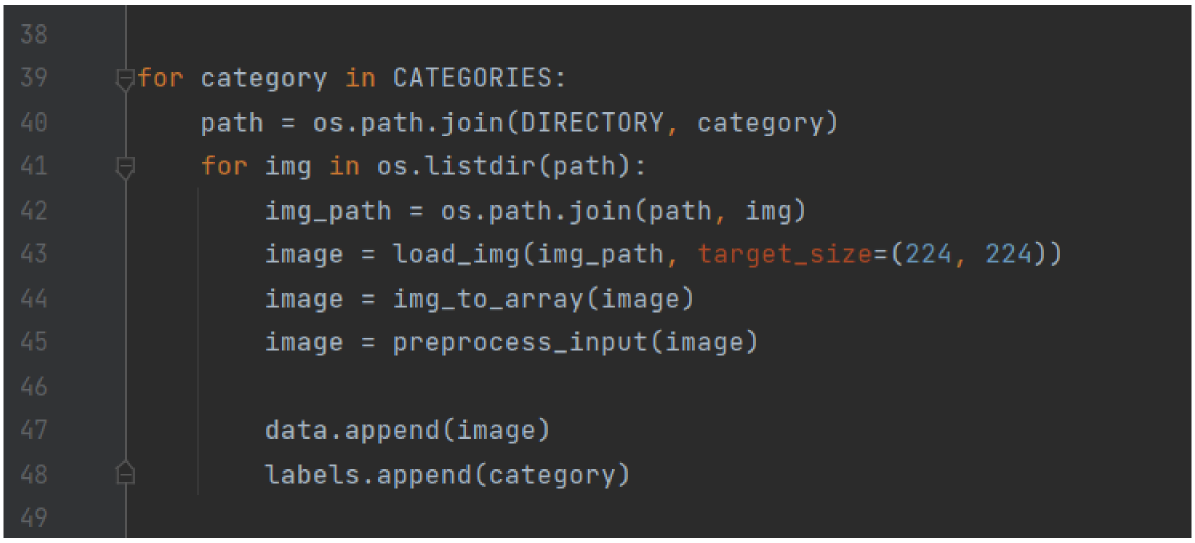
There have been two lists of data and labels made. Neither list had anything on it. All of the image arrays were added to the data at the same time. All of the matching images, whether or not they have a mask, were added to the labels list. They acted as labels for the pictures, showing whether or not they had masks.



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

Figure 17: List

The categories were then cycled through using the for-loop function. A link has been made to the route with the directory and category. "Os.lisdir" makes a list of the paths to all the images in the loop. The next step is to use the Load image function. The image load is taken care of by the "keras.preprocessing.image" library that was imported. With load image, the image path has been loaded, and the target image size is (224, 224) pixels. The width and height of an image describe its size. When the images were loaded, they were put into a list called an array. The images were turned into an array with the "img to array" method. The image was put into the array module with the help of the "keras.preprocessing.image" package. After putting the images into an array, the "preprocess input" function of a library was used. MobileNet was used to make this model. Then, all of the pictures were added to the data list and given names in the labels list. Because of the MobileNet library, the "preprocess input" method was used. When using the MobileNet library, you need to use the "preprocess input" function.

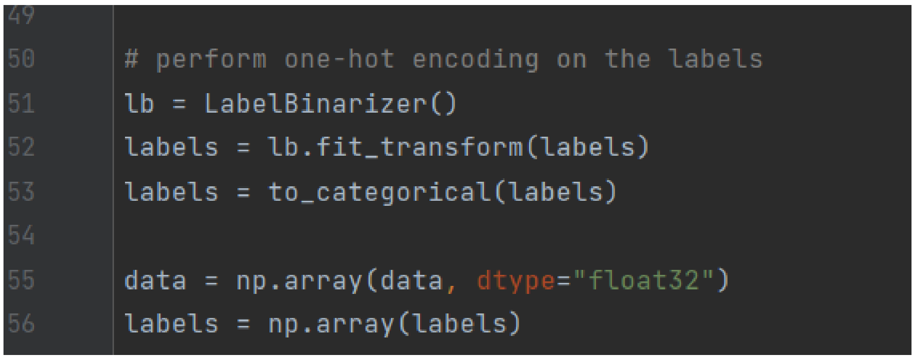


```
for category in CATEGORIES:
    path = os.path.join(DIRECTORY, category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = preprocess_input(image)

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

Figure 18: For loop applied

All of the data were given in the form of numbers. On the other hand, the values of the labels stayed the same whether they had a mask or not. Because of this, these text characters were changed into Binary. This was done with the "LabelBinarizer" function from the "sklearn.preprocessing" package. Then, categorical variables were made using "with mask" and "without mask." After the data was split into categorical variables, it was processed and labeled in numpy arrays, since the deep learning module only works with arrays. The "np.array" methods were used to turn the list into an array. NP is short for the word "numpy." A numpy array has also been made out of the labels list.

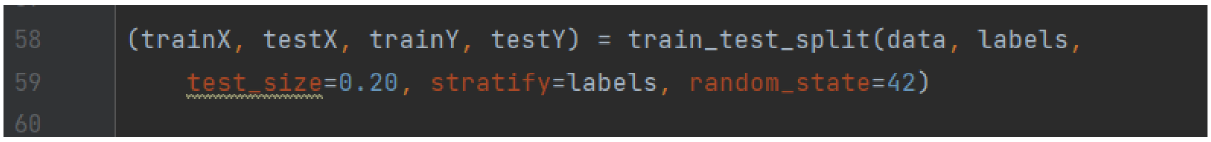


```
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

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

Figure 19: Labels for data

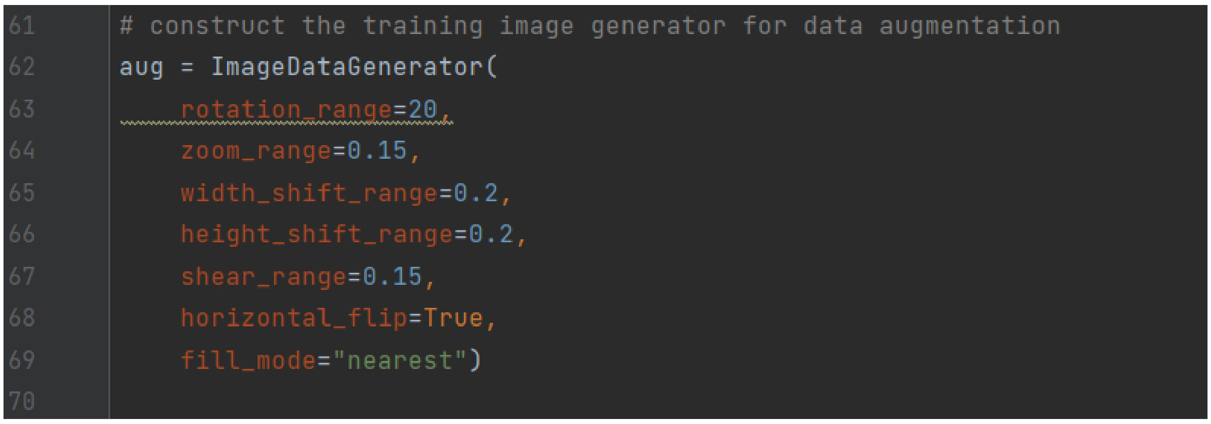
Training and testing data were split into two groups: trainX and testY and trainY and testY. There were two kinds of datasets used in the process of classifying: training datasets and testing datasets. The training set was used to teach a model, while the test set was used to test the model that had been taught. As only 30% of the testing sets were used, the other 80% were used to train more. For the labels, the value stratify was used, and for the random state value, the value 42 was used. Data preparation was done because the datasets were split into training and testing sets. The data was split into train and test sets using the "sklearn.model selection import train test split" package. In addition to the convolution neural network, MobileNet was also used to process images. When the images were done being worked on, they were sent to MobilNet. Then, max-pooling was used to make a fully connected layer.



```
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)
```

Figure 20: Data split

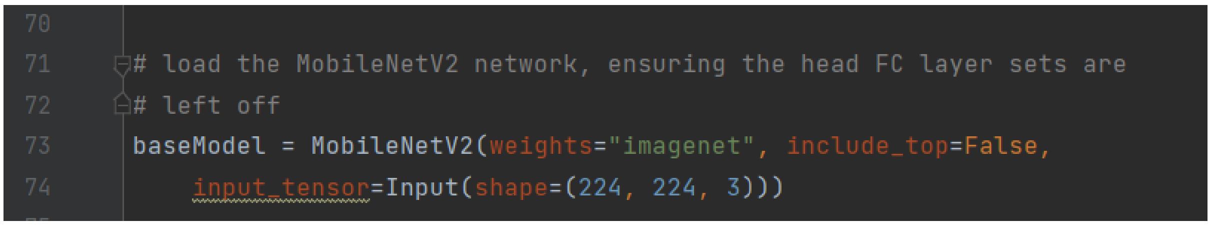
It has been made into an "ImageDataGenerator." The "ImageDataGenerator" function was used to make data documentation. By flipping, shifting, and rotating it, among other things, it was able to make many different images. So, it made it possible to make more datasets. Each one has a value written next to it. Image generator was mostly used to make multiple images out of a single image by changing its properties.



```
# 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")
```

Figure 21: Image Data Generation

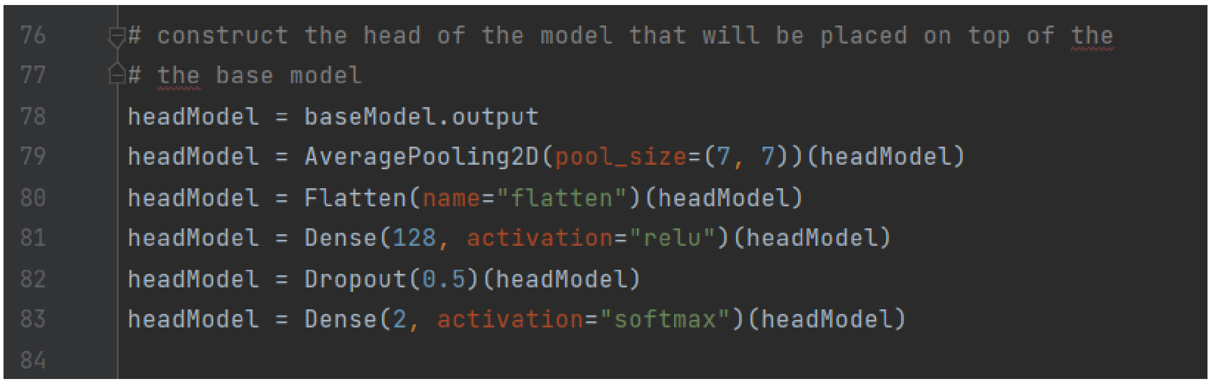
A baseModel has been made using MobileNetV2. A parameter called "weights = imagenet" was used to pre-train an image model. The Boolean value "include top = False" was then talked about. At the top of the network was a layer that was fully linked. The shape of the image after it went through "input tensor." The size of the image (224, 224) was given, which was set in the section on preprocessing the data. Three values, like 224, 224, and 3, were used to represent three channels in the image. In RGB color images, these three channels were used to show red, blue, and green.



```
# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
```

Figure 22: Parameters setting

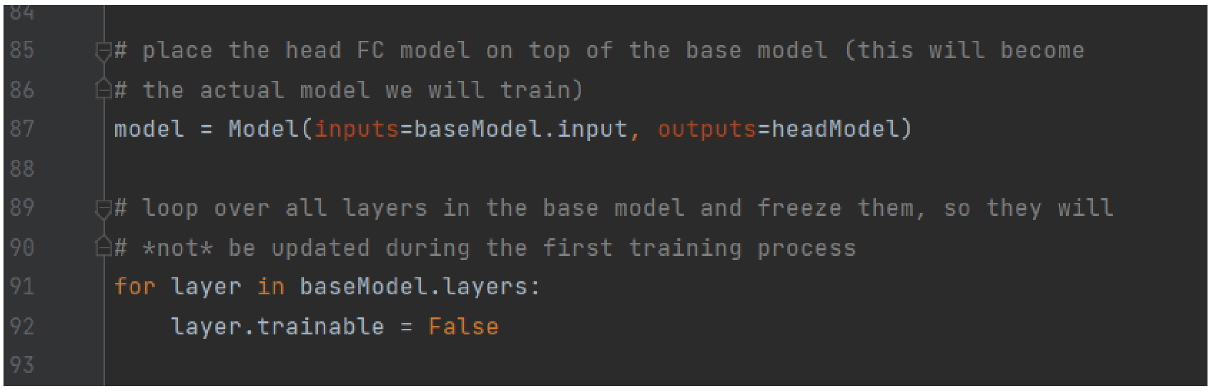
After the baseModel was done, Pooling was used to build a fully connected layer. HeadModel object was made, and the output was passed as the first parameter to the baseModel. Someone built a pool, and the size of the pool is (7, 7). The layer was made flat, and a layer of 128 neurons was put on top of it. Relu was the activation layer, so in non-linear use cases, relu is the activation function. Dropout is only used to make sure the model doesn't fit too well. The final model had two layers, one of which wore a mask and the other of which did not. The HeadModel also uses the softmax activation function.



```
# construct the head of the model that will be placed on top of the
# 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)
```

Figure 23: Head of the model creation

Someone has made a model function. The model function has two parameters that are both inputs and outputs. The baseModel held the inputs, and the HeadModel held the outputs. At first, the base model's layers were frozen so that they couldn't be changed during the training process. They weren't really convolutional neural networks; they were just a stand-in. They were kept so that they could be trained.

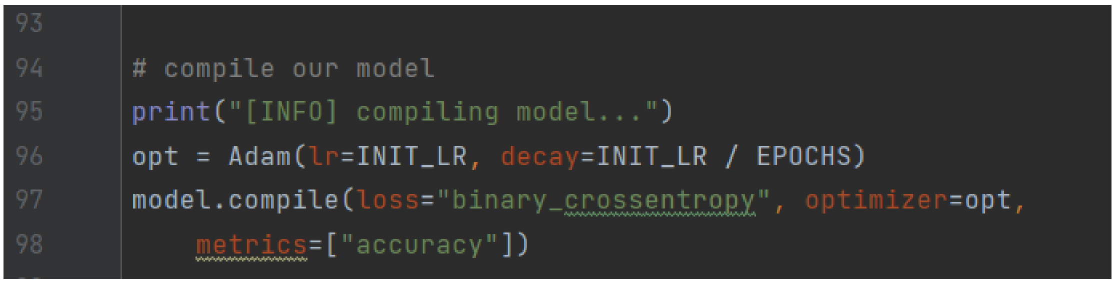


```
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them, so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

Figure 24: Function in model

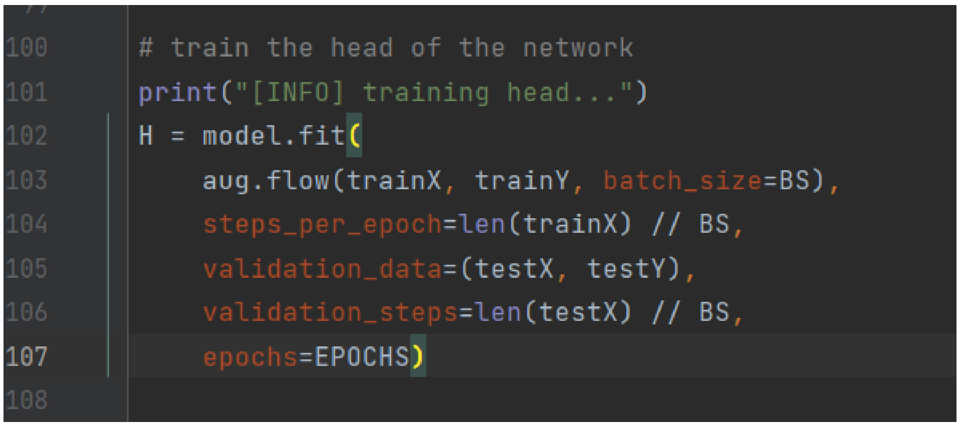
At the beginning of the code, it says that the initial learning rate was 1e-4. Since then, it has changed to decay. "Binary crossentropy" was chosen as the loss function. Adam Optimizer was used as the optimizer in the compilation. The adam optimizer was related to Relu, which was the best optimizer for any image prediction method. Metrics were used to measure how well the function worked, and that was the only metric to keep track of.



```
# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
```

Figure 25: Rate, accuracy and loss

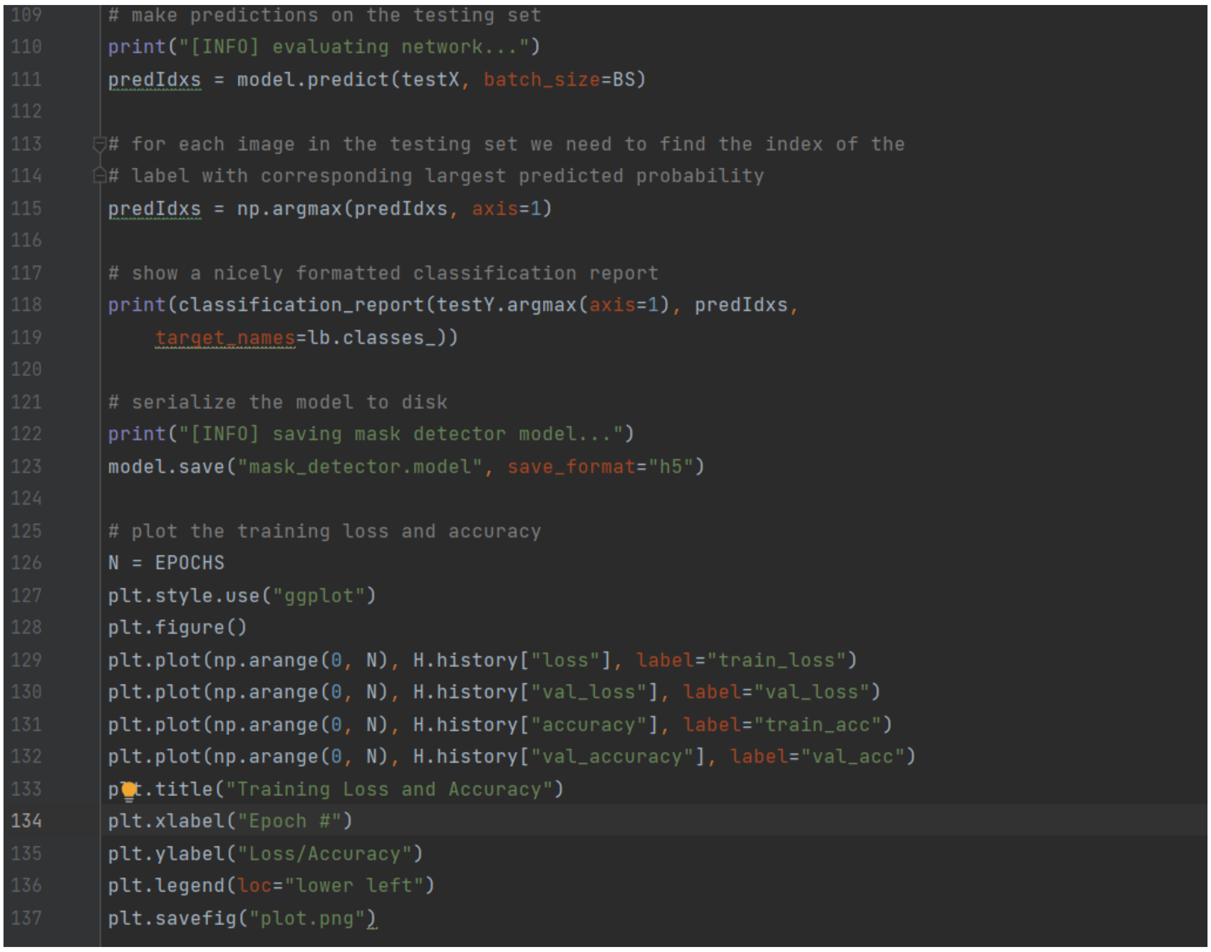
The model was currently being fitted. Tracking the data was done to see if there was a way to get more image data for training. A large number of datasets were made with the help of an image data generator. It is always best to use an image data generator for small datasets. The 'testX' and 'testY' have been used to check the data. The number of epochs given to the model, such as 20 epochs, which we've already talked about.



```
# train the head of the network
print("[INFO] training head...")
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)
```

Figure 26: Head network training

To test the model network, the "model.predict" method was used. For each image in the testing set, the highest predicted probability was used to figure out the label's index. As shown in FIGURE 27, this was done with the "np.argmax" command. The format of the classification report was good, and the model was saved at the end. It shows how Matplotlib was used to plot the accuracy and metrics. Matplotlib was also used to save the image. Two files showed up on the desktop as a result. One was a model file, and the other was a file called "plot.png" that was used by matplotlib to draw graphs.



```python
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
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("[INFO] saving mask detector 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")
```

Figure 27: Network evaluating

Testing the model

To run the model, the computer's command prompt had to be opened and the training file's location had to be found. The word "python" and the file name "train mask.py" were typed, and then the "enter" key was pressed to start it. After that, this model moved on to the training phase. It took a long time to teach all of the images what to do. After the mask detector model was trained, the "plot.png" image of the results was saved in the project folder. They made a graph of the accuracy and training loss. Both the accuracy and the loss have been shown in a clear way. The model looked like she was in good shape. Up to 20 epochs were given to the epochs down. The model worked well and did what it was supposed to do. On the local desk, the "plot.png" and "mask detector" files have been saved. Two files that were downloaded and saved in the face-detector folder were used to find faces. OpenCV was used for the camera to work.

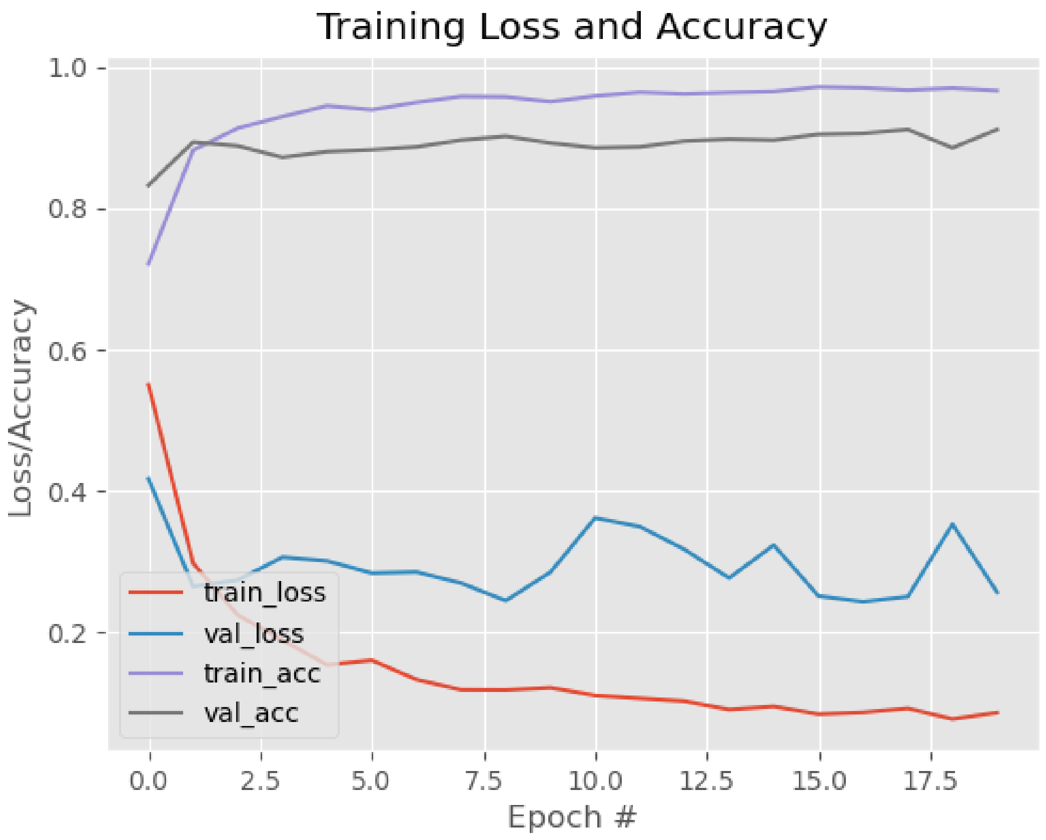
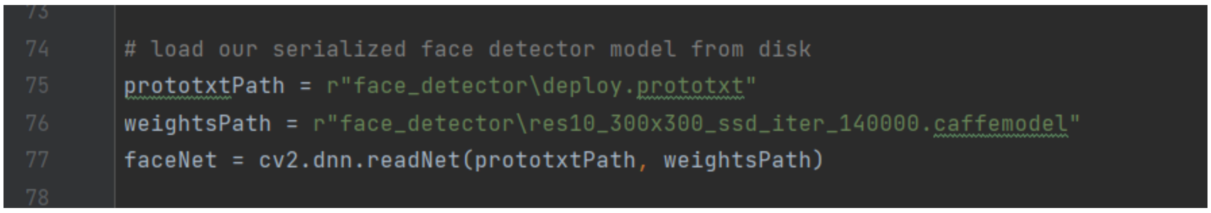


Figure 28: Matplotlib visualization

Mask identification process

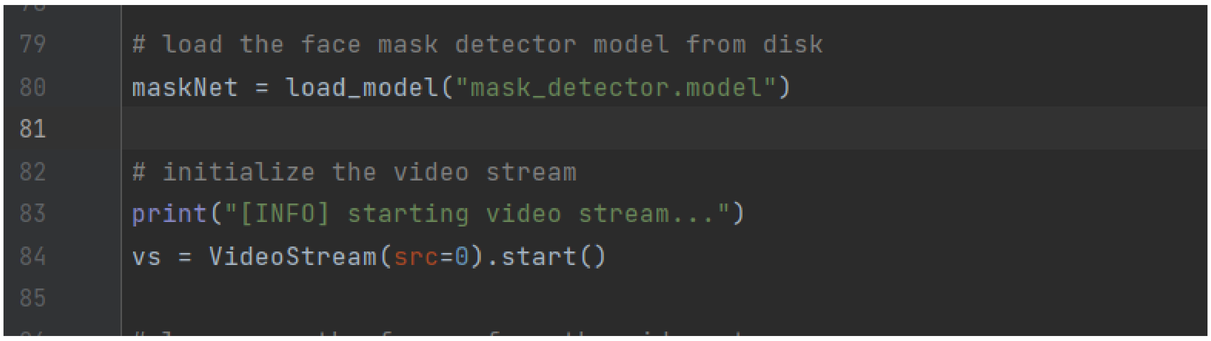
The first program to load was called "FaceNet." The purpose of using "FaceNet" is to find where the face detection pairs file is. The paths to the files were linked and put into a variable called "weightsPath." To use them, the "readNet" method from CV2 and the DNN model has been used. A deep neural network is what the letter DNN stands for.



```
# load our serialized face detector model from disk
prototxtPath = r"face_detector\deploy.prototxt"
weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
```

Figure 29: FaceNet

The "load model" method was used to load the model, and the "VideoStream" method was used to load the camera. Masks were found with the help of the load model. Masks and faces could be found with two different models. There was a function called src in the "VideoStream" method that represented the camera. And src=0 showed the main camera start function, which was used to get the camera to start loading.



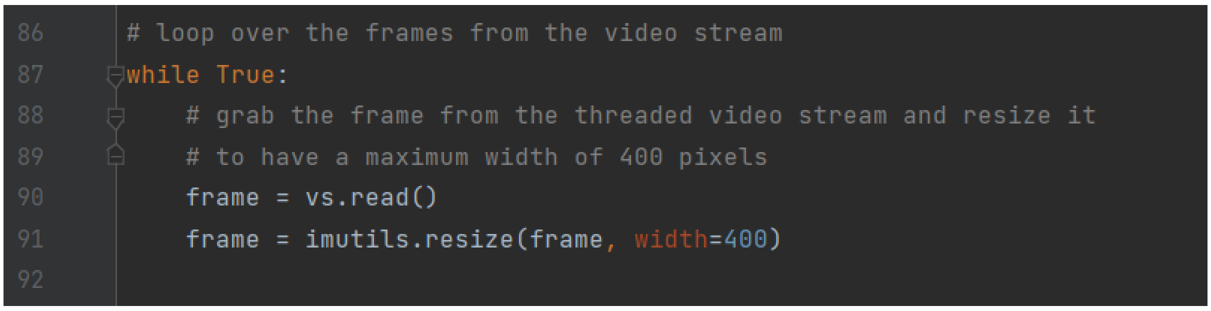
```
# load the face mask detector model from disk
maskNet = load_model("mask_detector.model")

# initialize the video stream
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()

```

Figure 30: Load face mask detector and video stream

The value was set to true, and a while loop was used. The read function went through each frame one by one. Each frame was just an image. So, to our eyes, any image that follows the right order and has that many frames per second looks like a video. After the frame was read, a new frame was made with a width of 400 pixels. It was possible to access a FaceNet, a maskNet, and a frame. FaceNet was used to guess faces, maskNet was used to recognize masks, and frame was used to show the video from the camera as it was being loaded.

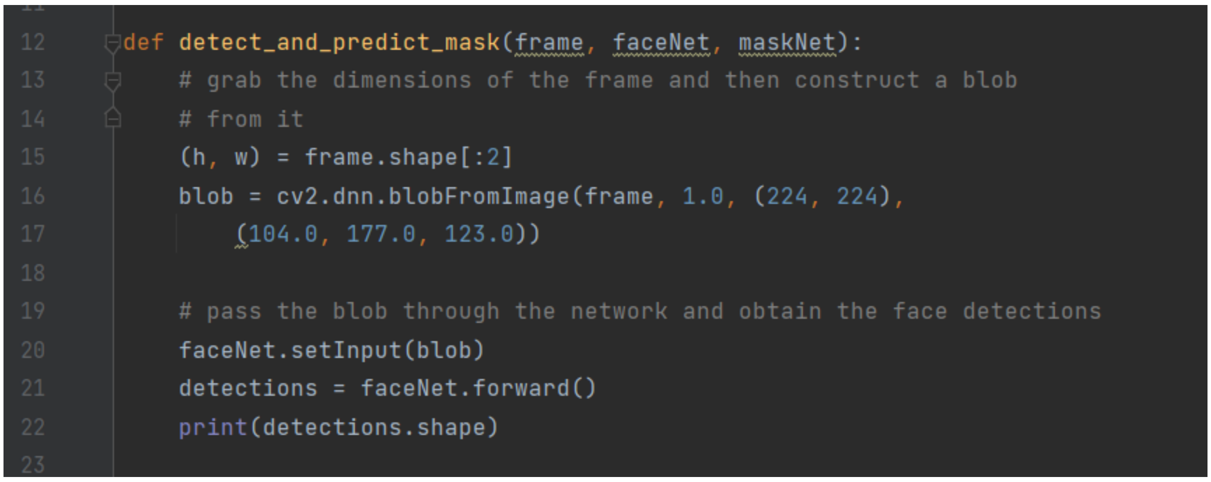


```
# loop over the frames from the video stream
while True:
    # grab the frame from the threaded video stream and resize it
    # to have a maximum width of 400 pixels
    frame = vs.read()
    frame = imutils.resize(frame, width=400)

```

Figure 31: While loop over the video frame

A detect and predict mask function was made using the frame, FaceNet, and maskNet arguments. After that, a normal manipulation took place. At the end of the method, the user's location and prediction were sent back to them. To figure out where it was, the x and y coordinates of the exact rectangle around the face were used. The prediction was based only on how well the person in the mask guessed. So, it was thought that the mask would be worn 90% of the time and not worn 10% of the time.

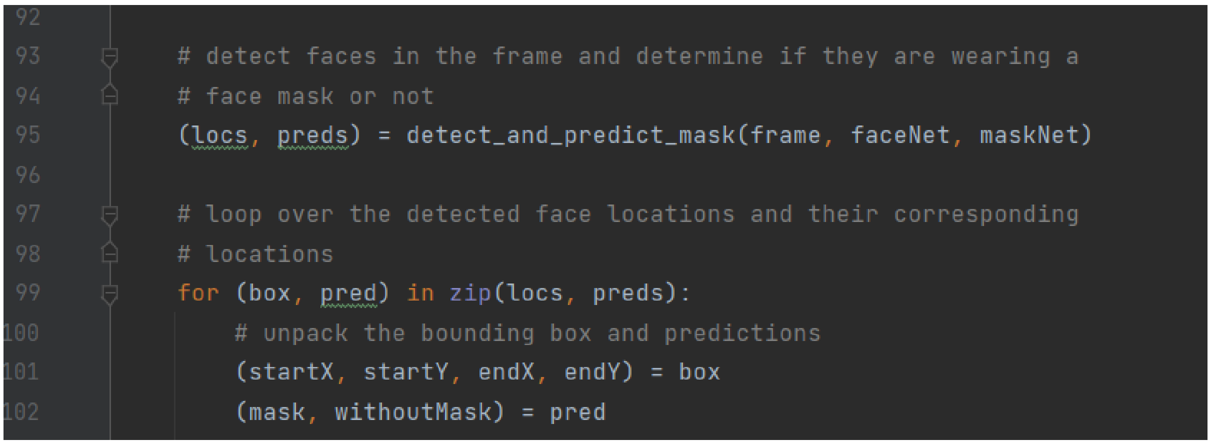


```python
def detect_and_predict_mask(frame, faceNet, maskNet):
    # grab the dimensions of the frame and then construct a blob
    # from it
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
        (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the face detections
    faceNet.setInput(blob)
    detections = faceNet.forward()
    print(detections.shape)
```

Figure 32: Obtaining the face detection

A call was made to the method, and a tuple showing the location and prediction was shown. The tuple's value was taken out and divided by the X and Y coordinates. For the mask prediction, a square was drawn in that spot. X1 and Y1 marked the beginning and end of X and Y, respectively. So, the first prediction was based on the mask, while the second was based on the fact that the mask wasn't there.

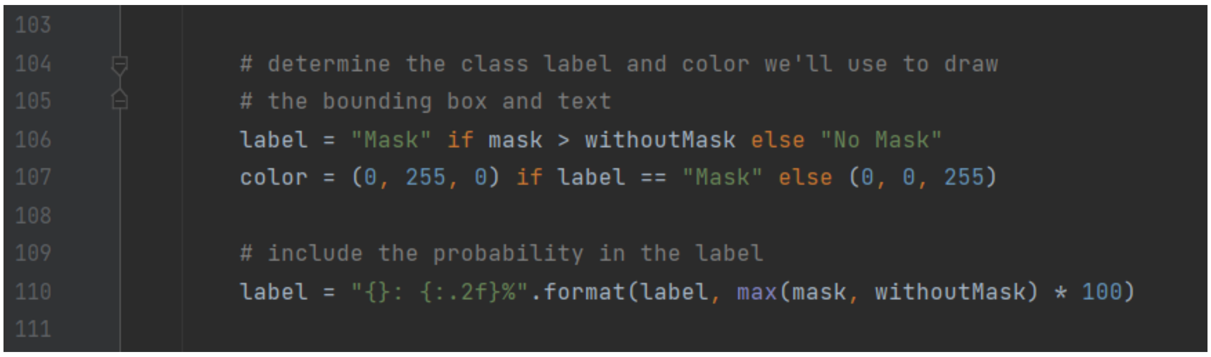


```python
    # detect faces in the frame and determine if they are wearing a
    # face mask or not
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

    # loop over the detected face locations and their corresponding
    # locations
    for (box, pred) in zip(locs, preds):
        # unpack the bounding box and predictions
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred
```

Figure 33: Detection and prediction function called

A label has been put in for image labeling. The label has been put in the right place in the image, with or without a mask. After the labeling step was done, it was put on the rectangle that was made. RGB, which was the standard for color coding on OpenCV2, has been used for color. As a result, the lowest value was 0 and the highest value was 255. In RGB, green was shown by b=0, g=255, and r=0. The color green showed the image with the mask, while the color red showed the image without the mask. For the second pair, red was worth the most. The label was then set up with a format string. It showed the percentage of the prediction. As was already said, the mask was expected to work 90% of the time, while no mask was expected to work 10% of the time. So, the most likely mask was the one with the highest percentage.

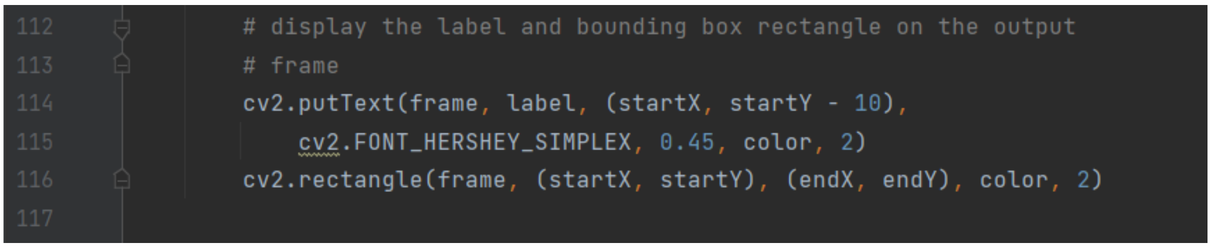


```
    # determine the class label and color we'll use to draw
    # the bounding box and text
    label = "Mask" if mask > withoutMask else "No Mask"
    color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

    # include the probability in the label
    label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
```

Figure 34: Image labeling

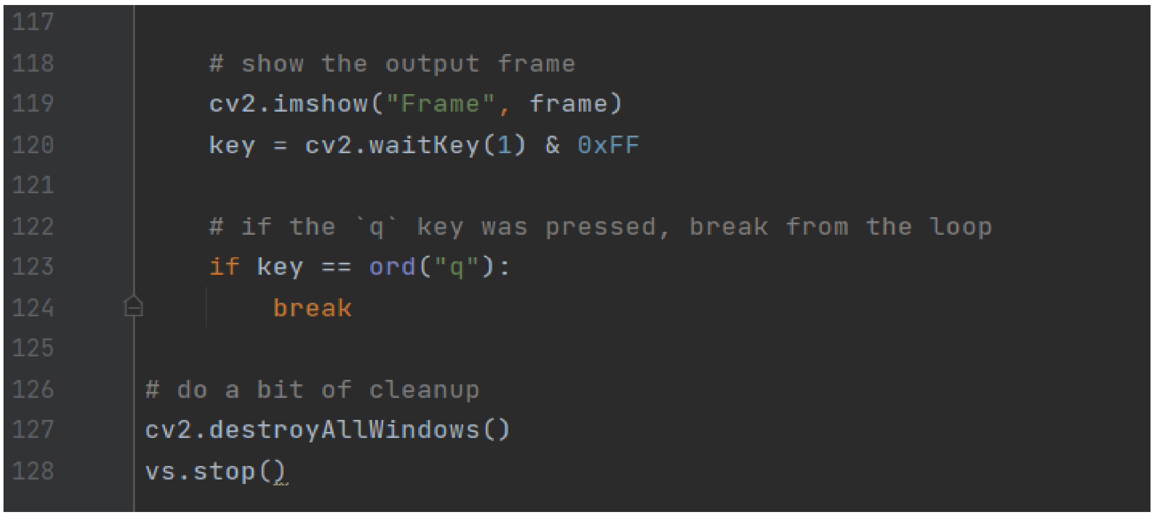
The picture frame was part of the text. The size of the font was set to 0.45. The X value stayed the same, but 10 pixels were taken away from the Y value. There was no overlap between the text and the picture box. A square was painted on the frame. At the beginning of each frame was a rectangle with the X and Y coordinates and the color code. The rectangle was 2 inches thick.



```
    # 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)
```

Figure 35: Image frames algorithms

Here, the frame presentation and the order in which the images were shown were both made up. This was where the while loop was finally broken. The letter "q" was saved as a configuration for getting out of the loop, and the q button was used to get out of the loop. In the end, all of the windows were closed, and the video was turned off.



```

    # show 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

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```

Figure 36: Output frame, loop break and window turn off

RESULT

To see the result, the computer's command line was opened and the detect mask video file's exact location was found. By entering "python detect mask video.py" and pressing the enter button, the program was started. The machine then used an algorithm to find faces. A face has been found in a window called "frame." The detector put the face into two groups: mask and no mask. The accuracy score showed that the mask was found. Face mask detection was done in real-time with my own face. The MobileNet algorithm was used to figure out the faces in the video. Also, every time a new face image comes into the camera, the system finds if he/she is wearing the face mask or not.

FINDINGS

This thesis built a face mask detector. This was accomplished utilizing OpenCV, MobileNet, machine learning, and deep learning. Masks are fashionable now. Masks are one of the only strategies to guard against air disease and keep people healthy. This idea may be utilized at airports, railway stations, workplaces, schools, and public venues to keep people secure. This system now uses a webcam. This device might be used with CCTV cameras to detect persons without masks. The face mask detector didn't employ altered photos. The concept is valid, and it's simple to execute on embedded computers since it utilizes MobileNetV2 architecture. Due to the propagation of viruses, this method may be used to identify face masks in real-time. This initiative ensures safety in airports, railway stations, offices, schools, and other public locations.

The hardware and software that will be utilized to develop the suggested system will be examined. The training and performance parameters of the depth-wise separable convolutional network structure are thus exhaustively investigated. For testing, we used an Intel Core processor and Windows 10. Python 3.5 was the programming language used. It utilizes Python 3.5's PyTorch package for image embedding processing and MATLAB 2019b for image analysis. STIF frames are sufficient to use the pre-trained model. This dataset is used to evaluate the strategy proposed in the section addressing research issues. There are now training and testing datasets available. The scaling factor was initially set at 0.001, and it decreases by a factor of 0.90 every 10 epochs. The momentum value used by the Adam optimizer is 0.999. The training technique is repeated for a total of 100 epochs. The design of the proposed MobileNet parameter settings for successful face image classification (Mask/no mask) is adopted. For the implementation of a face image classification system, the allocation of processor time for a certain deductive logic across network tiers. Three interpretations per second were sufficient for feature extraction and picture source processing during network analysis.

FUTURE WORK

For now, this system works with a live video webcam. With more work, this model could work with CCTV cameras to find and identify people who aren't wearing masks. The face mask detector did not use a dataset of morphed masked images. The model is correct, and since it uses the MobileNetV2 architecture, it's also easy to run on embedded systems because it uses less computing power (Google Coral, Raspberry Pi, etc.). Due to the spread of different viruses, this system can be used in real-time applications that need to detect face masks for safety reasons. This project can be used in airports, train stations, offices, schools, and other public places to make sure that safety rules are followed.

CONCLUSION

Object detection is a type of method that is part of computer vision. Using this method, we can find and follow objects in pictures and videos. Object identification, which is also called "object detection," can be utilized for face recognition, vehicle recognition, self-driving cars, surveillance systems, and many other things. Using deep learning and machine learning to find objects has become a major area of study in recent years.

The aim of this thesis was to make a face mask detector. This goal was reached by using technologies such as OpenCV, MobileNet, machine learning, and deep learning to put the idea into practice. Masks are getting more and more popular these days. Masks are among the few ways to protect against air illness, and they play an important role in keeping people healthy and free of respiratory illnesses. To keep people safe, this project can be combined with embedded technology and used in airports, train stations, offices, schools, and public spaces, among other places.

REFERENCES

AI Detect Objects. Medium. (2022). Retrieved 5 August 2022, from https://towardsdatascience.com/how-does-ai-detect-objects-technical-d8d63fc12881.

Object Detection. Medium. (2022). Retrieved 5 August 2022, from https://medium.com/ml-research-lab/what-is-object-detection-51f9d872ece7.

Deep Learning. Medium. (2022). Retrieved 5 August 2022, from https://medium.com/analytics-vidhya/deep-learning-artificial-neural-network-ann-13b54c3f370f.

Python in Deep Learning. Built In. (2022). Retrieved 5 August 2022, from https://builtin.com/data-science/deep-learning-python.

Team, K. (2022). *MobileNetV2 and MobileNetV3*. Keras.io. Retrieved 6 August 2022, from https://keras.io/api/applications/mobilenet/.

G.H. Hardy. Quotefancy.com. (2022). Retrieved 16 July 2022, from https://quotefancy.com/g-h-hardy-quotes.

Python Basics - GeeksforGeeks. GeeksforGeeks. (2022). Retrieved 19 July 2022, from https://www.geeksforgeeks.org/python-basics/.

GeeksforGeeks. GeeksforGeeks. (2022). Retrieved 20 July 2022, from https://www.geeksforgeeks.org/libraries-in-python/.

TensorFlow Intro. W3schools.com. (2022). Retrieved 20 July 2022, from https://www.w3schools.com/ai/ai_tensorflow_intro.asp.

Team, K. (2022). *Keras: deep learning API*. Keras.io. Retrieved 8 August 2022, from https://keras.io/.

OpenCV-Python Tutorials. Docs.opencv.org. (2022). Retrieved 8 August 2022, from https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html.

Libraries. Medium. (2022). Retrieved 8 August 2022, from https://medium.com/javarevisited/important-libraries-of-opencv-56b14705bf0e.

NumPy?, W. (2022). *NumPy Need*. EDUCBA. Retrieved 9 August 2022, from https://www.educba.com/what-is-numpy/.

SciPy in Python. Guru99. (2022). Retrieved 9 August 2022, from https://www.guru99.com/scipy-tutorial.html.

imutils. Kaggle.com. (2022). Retrieved 10 August 2022, from https://www.kaggle.com/datasets/meckdahl/imutils.

plotlib Tutorial. W3schools.com. (2022). Retrieved 9 August 2022, from https://www.w3schools.com/python/matplotlib_intro.asp.

Kaggle. Kaggle.com. (2022). Retrieved 11 August 2022, from https://www.kaggle.com/datasets.

NVIDIA NGC. NVIDIA NGC Catalog. (2022). Retrieved 14 July 2022, from https://catalog.ngc.nvidia.com/orgs/nvidia/teams/tao/models/facenet.

TAO Toolkit. Docs.nvidia.com. (2022). Retrieved 14 July 2022, from https://docs.nvidia.com/tao/tao-toolkit/text/purpose_built_models/facedetectnet.html.

KFC China. the Guardian. (2022). Retrieved 14 July 2022, from https://www.theguardian.com/technology/2017/jan/11/china-beijing-first-smart-restaurant-kfc-facial-recognition.

KFC payment China. Thales Group. (2022). Retrieved 14 July 2022, from https://www.thalesgroup.com/en/markets/digital-identity-and-security/banking-payment/magazine/kfc-use-facial-recognition-payment-china.

Naked Payments Deployment. FindBiometrics. (2022). Retrieved 14 July 2022, from https://findbiometrics.com/china-kfc-worlds-biggest-naked-payments-deployment-511142/#:~:text=mobile%20payments%20platform.-,At%20KPRO%2C%20the%20system%20was%20installed%20in%20self%2Dservice%20kiosks,or%20even%20a%20mobile%20device.

Tech enhances The BRIT Awards. Digital Barriers. (2022). Retrieved 15 July 2022, from https://www.digitalbarriers.com/facial-recognition-enhances-the-o2-security-operations-at-the-brit-awards-and-national-television-awards/.

National Television Awards | Event Industry News. Eventindustrynews.com. (2022). Retrieved 15 July 2022, from https://www.eventindustrynews.com/news/facial-recognition-enhances-the-o2-security-operations-at-the-brit-awards-and-national-television-awards.

Rekognition. (2022). Retrieved 15 July 2022, from https://docs.aws.amazon.com/rekognition/latest/dg/what-is.html.

Machine Learning Image and Video Analysis-AWS. Amazon Web Services, Inc. (2022). Retrieved 15 July 2022, from https://aws.amazon.com/rekognition/.

Youtube.com. 2022. *youtube*. [online] Available at: https://www.youtube.com/watch?v=fk-TxySUAzw

Shreya, D. (2021). DIGITAL IMAGE PROCESSING AND RECOGNITION USING PYTHON. *International Journal Of Engineering Applied Sciences And Technology*, *5*(10). https://doi.org/10.33564/ijeast.2021.v05i10.046

van der Walt, S., Schönberger, J., Nunez-Iglesias, J., Boulogne, F., Warner, J., & Yager, N. et al. (2014). scikit-image: image processing in Python. *Peerj*, *2*, e453. https://doi.org/10.7717/peerj.453

Dey, S. (2020). *Python Image Processing Cookbook*. Packt Publishing.

Hearty, J. *Advanced machine learning with Python*.

Image Recognition in Python based on Machine Learning – Example & Explanation for Image Classification Model | ASPER BROTHERS. ASPER BROTHERS. (2022). Retrieved 10 August 2022, from https://asperbrothers.com/blog/image-recognition-in-python/.

Image Recognition and Classification in Python with TensorFlow and Keras. Stack Abuse. (2022). Retrieved 8 August 2022, from https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/.

Python for Image Recognition - OpenCV. Topcoder.com. (2022). Retrieved 10 August 2022, from https://www.topcoder.com/thrive/articles/python-for-image-recognition-opencv.

TensorFlow Image Recognition Python API Tutorial. Medium. (2022). Retrieved 12 August 2022, from https://towardsdatascience.com/tensorflow-image-recognition-python-api-e35f7d412a70.

Malik, M. (2021). DETEKSI SUHU TUBUH DAN MASKER WAJAH DENGAN MLX90614, OPENCV, KERAS/TENSORFLOW, DAN DEEP LEARNING. *Jurnal Engine: Energi, Manufaktur, Dan Material*, *6*(1), 19. https://doi.org/10.30588/jeemm.v6i1.910

Patil, M., Tiwari, P., & Sonkamble, R. (2022). Covid-19 Face Mask Recognition using Live Camera and Face Mask Detection Using TensorFlow and Keras. *International Journal For Research In Applied Science And Engineering Technology*, *10*(4), 2778-2782. https://doi.org/10.22214/ijraset.2022.41909

Mishra, A., & Amutha, M. (2022). Real Time Face Mask Detector using Machine Learning, Python, OpenCV and Keras. *International Journal Of Computer Applications*, *184*(8), 50-52. https://doi.org/10.5120/ijca2022922055

M.S, A. (2020). Estimation of Body Fat Percentage Using Neural Networks, Tensorflow and Opencv. *Journal Of Research On The Lepidoptera*, *51*(2), 696-709. https://doi.org/10.36872/lepi/v51i2/301128

LI, A., LI, Y., & LI, X. (2017). TensorFlow and Keras-based Convolutional Neural Network in CAT Image Recognition. *Destech Transactions On Computer Science And Engineering*, (cmsam). https://doi.org/10.12783/dtcse/cmsam2017/16428

Nagoriya, H. (2020). Attendance System using Face Recognition utilizing OpenCV Image Processing Library. *International Journal For Research In Applied Science And Engineering Technology*, *8*(6), 1811-1814. https://doi.org/10.22214/ijraset.2020.6297

Babcock, J., & Bali, R. *Generative AI with Python and TensorFlow 2*.

Real-Time Face Recognition: An End-To-End Project. Medium. (2022). Retrieved 11 August 2022, from https://towardsdatascience.com/real-time-face-recognition-an-end-to-end-project-b738bb0f7348.

Top 7 Use Cases for Facial Recognition in 2022 | FaceMe® by CyberLink. Cyberlink.com. (2022). Retrieved 10 August 2022, from https://www.cyberlink.com/faceme/insights/articles/228/How_is_Facial_Recognition_Used_in_2021.

(2022). Retrieved 10 August 2022, from https://www.mygreatlearning.com/blog/real-time-face-detection/.

Where is facial recognition used?. Thales Group. (2022). Retrieved 11 August 2022, from https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/inspired/where-facial-recognition-used.

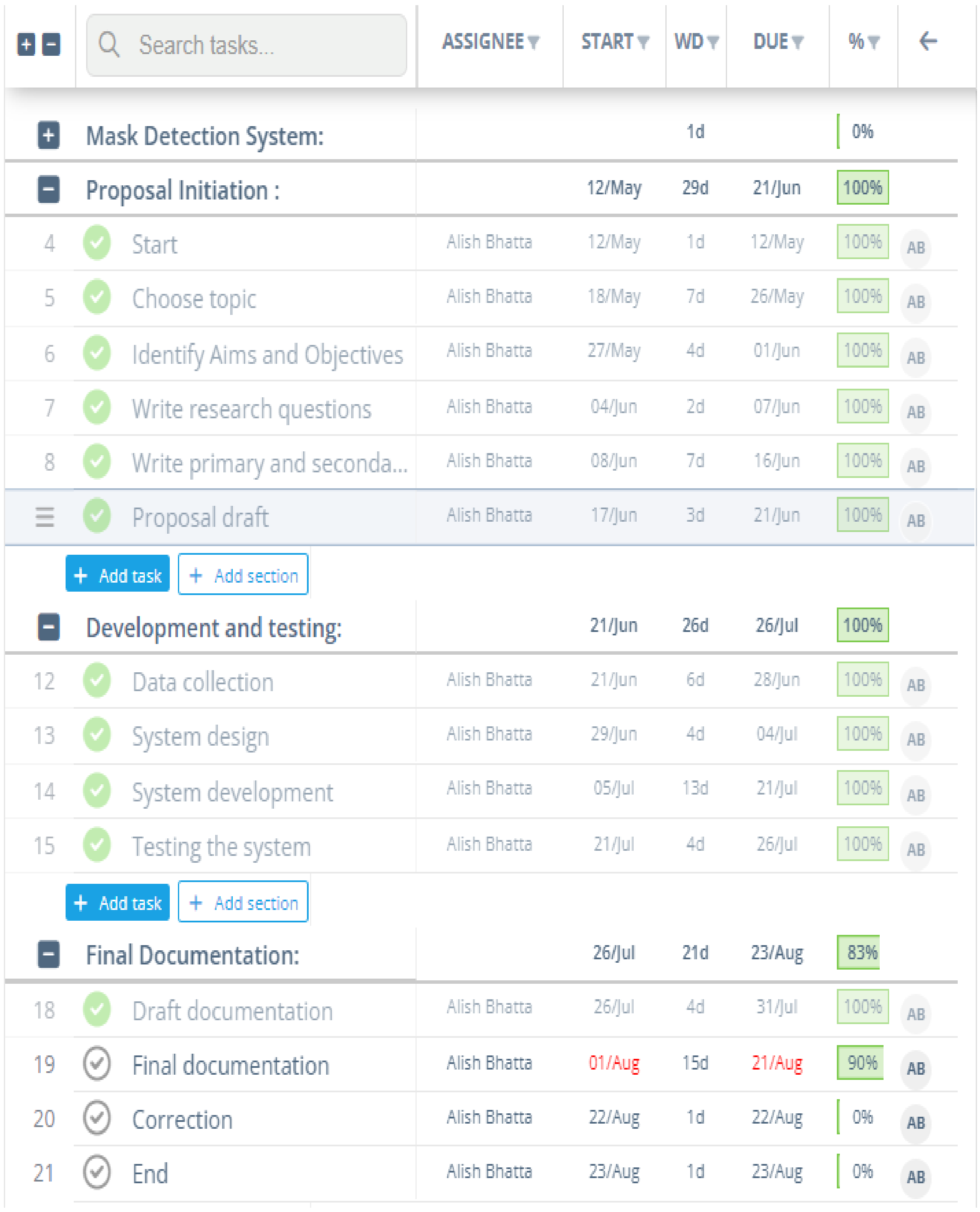
How does a Real-Time Face Recognition work with OpenCV?. Medium. (2022). Retrieved 10 August 2022, from https://medium.com/eazy-ciphers/how-does-a-real-time-face-recognition-work-with-opencv-ddd4c1cd4b43.

Malik, M. (2021). DETEKSI SUHU TUBUH DAN MASKER WAJAH DENGAN MLX90614, OPENCV, KERAS/TENSORFLOW, DAN DEEP LEARNING. *Jurnal Engine: Energi, Manufaktur, Dan Material*, *6*(1), 19. https://doi.org/10.30588/jeemm.v6i1.910

Shreya, D. (2021). DIGITAL IMAGE PROCESSING AND RECOGNITION USING PYTHON. *International Journal Of Engineering Applied Sciences And Technology*, *5*(10). https://doi.org/10.33564/ijeast.2021.v05i10.046

M.S, A. (2020). Estimation of Body Fat Percentage Using Neural Networks, Tensorflow and Opencv. *Journal Of Research On The Lepidoptera*, *51*(2), 696-709. https://doi.org/10.36872/lepi/v51i2/301128

APPENDIX



		ASSIGNEE	START	WD	DUE	%	
	Mask Detection System:			1d		0%	
	Proposal Initiation :		12/May	29d	21/Jun	100%	
4	✓ Start	Alish Bhatta	12/May	1d	12/May	100%	AB
5	✓ Choose topic	Alish Bhatta	18/May	7d	26/May	100%	AB
6	✓ Identify Aims and Objectives	Alish Bhatta	27/May	4d	01/Jun	100%	AB
7	✓ Write research questions	Alish Bhatta	04/Jun	2d	07/Jun	100%	AB
8	✓ Write primary and seconda...	Alish Bhatta	08/Jun	7d	16/Jun	100%	AB
	✓ Proposal draft	Alish Bhatta	17/Jun	3d	21/Jun	100%	AB
	Development and testing:		21/Jun	26d	26/Jul	100%	
12	✓ Data collection	Alish Bhatta	21/Jun	6d	28/Jun	100%	AB
13	✓ System design	Alish Bhatta	29/Jun	4d	04/Jul	100%	AB
14	✓ System development	Alish Bhatta	05/Jul	13d	21/Jul	100%	AB
15	✓ Testing the system	Alish Bhatta	21/Jul	4d	26/Jul	100%	AB
	Final Documentation:		26/Jul	21d	23/Aug	83%	
18	✓ Draft documentation	Alish Bhatta	26/Jul	4d	31/Jul	100%	AB
19	✓ Final documentation	Alish Bhatta	01/Aug	15d	21/Aug	90%	AB
20	✓ Correction	Alish Bhatta	22/Aug	1d	22/Aug	0%	AB
21	✓ End	Alish Bhatta	23/Aug	1d	23/Aug	0%	AB

Figure 37: Project planning with timeline

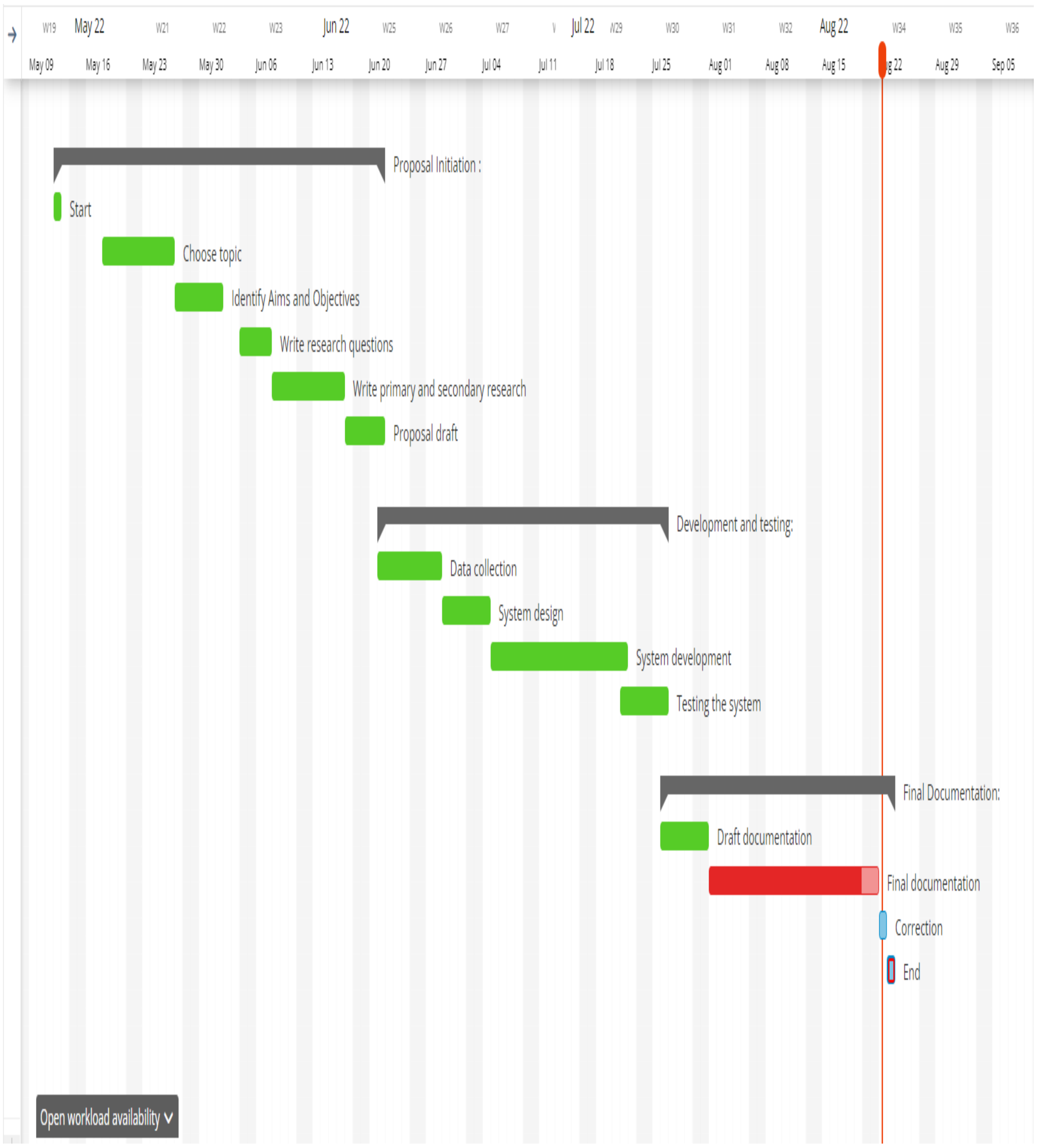
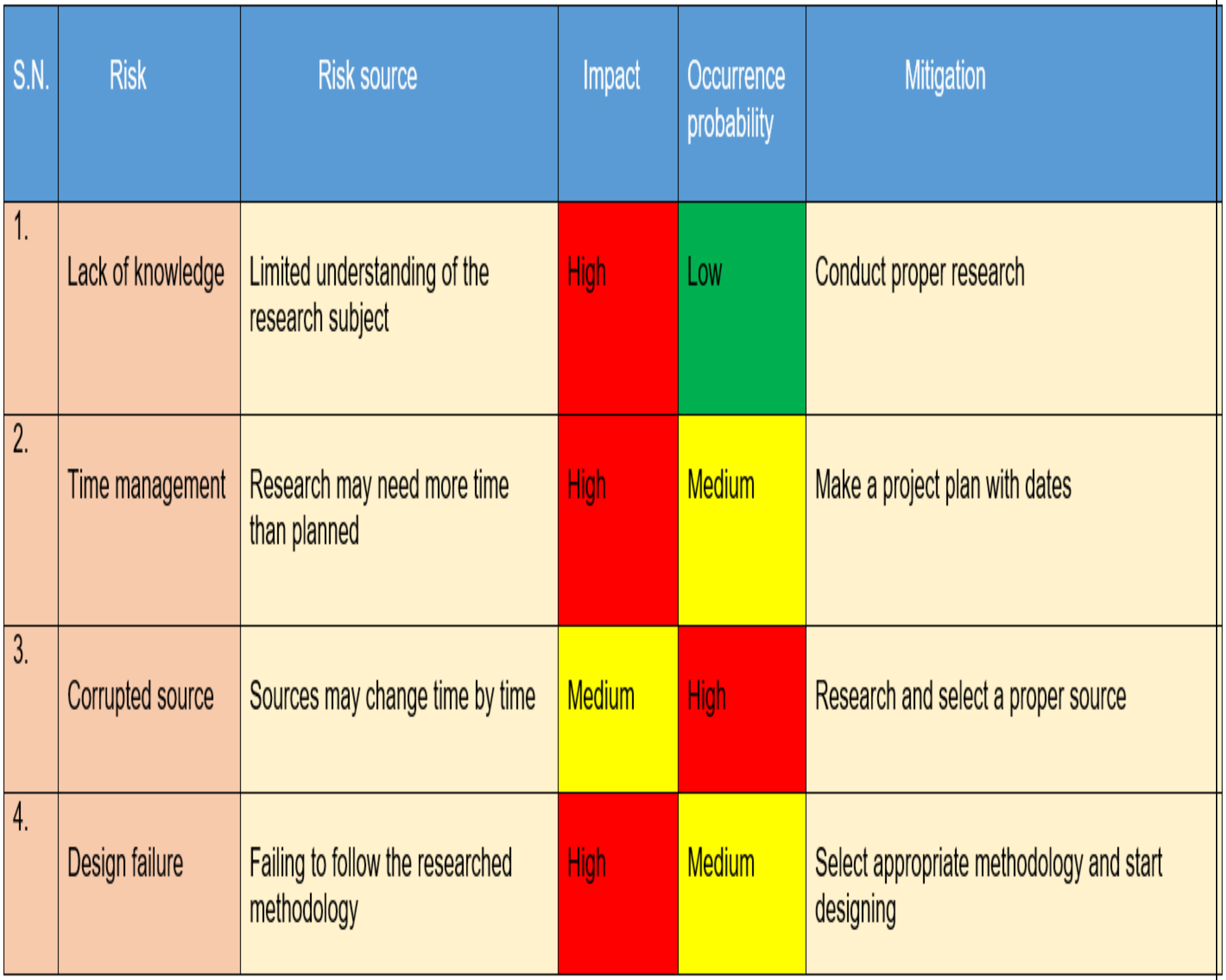
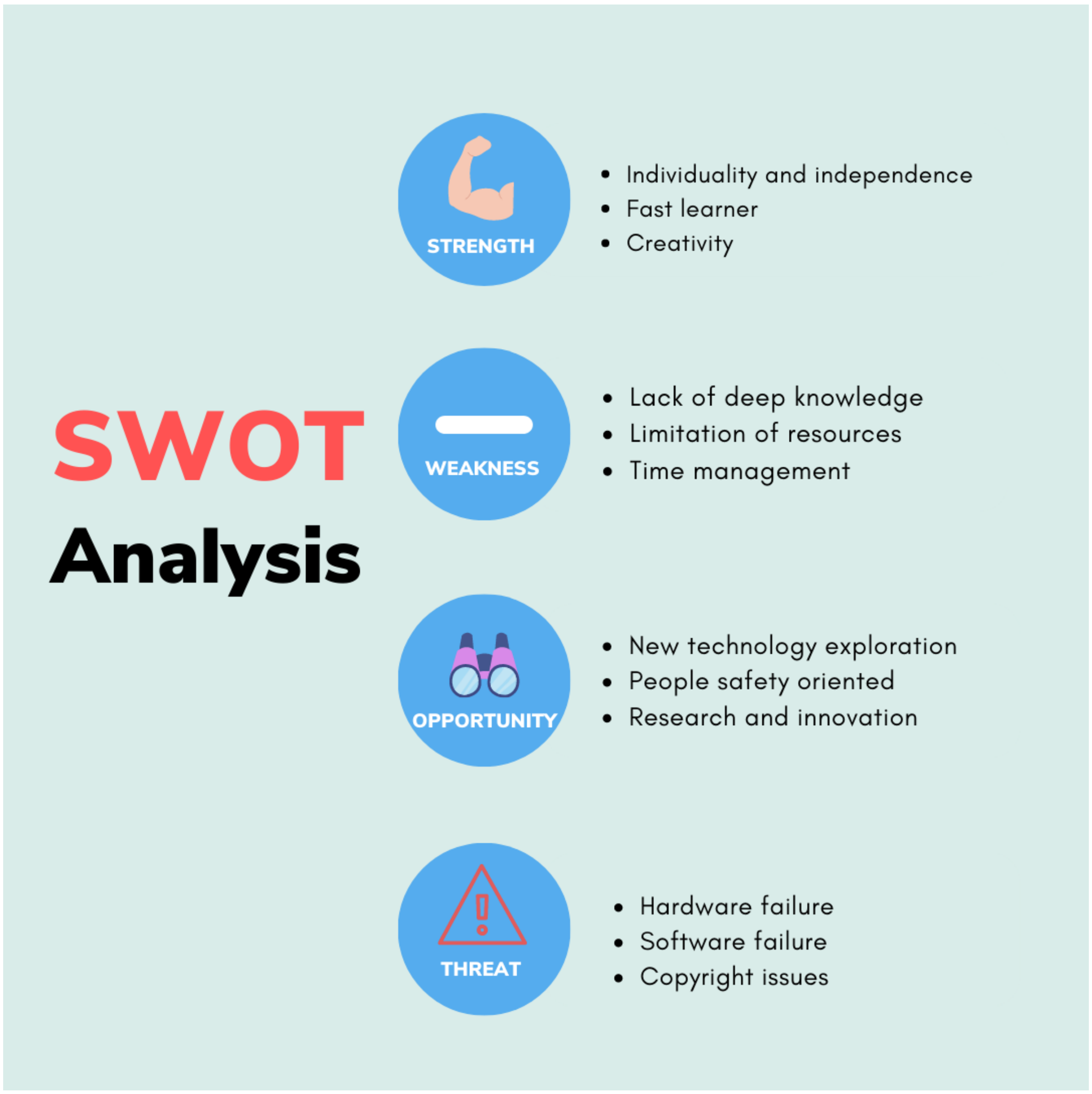


Figure 38: Gantt chart



S.N.	Risk	Risk source	Impact	Occurrence probability	Mitigation
1.	Lack of knowledge	Limited understanding of the research subject	High	Low	Conduct proper research
2.	Time management	Research may need more time than planned	High	Medium	Make a project plan with dates
3.	Corrupted source	Sources may change time by time	Medium	High	Research and select a proper source
4.	Design failure	Failing to follow the researched methodology	High	Medium	Select appropriate methodology and start designing

Figure 39: Risk management planning



SWOT Analysis

STRENGTH
- Individuality and independence
- Fast learner
- Creativity

WEAKNESS
- Lack of deep knowledge
- Limitation of resources
- Time management

OPPORTUNITY
- New technology exploration
- People safety oriented
- Research and innovation

THREAT
- Hardware failure
- Software failure
- Copyright issues

Figure 40: SWOT Analysis