FYP-1

Final Evaluation Report

F21-38-D-HospitalAid

Members:

Mr. Hassan Shahzad 18i-0441

Ms. Sana Ali Khan 18i-0439

Supervisor:

Mr. Syed Muhammad H. Mustafa

Dated: 11th December, 2021

Anti-Plagiarism Declaration

This is to declare that the above FYP report produced under the title **Hospital Aid** is the sole contribution of the author(s) and no part hereof has been reproduced on **as it is** basis (cut and paste) which can be considered as **Plagiarism.** All referenced parts have been used to argue the idea and have been cited properly. I/We will be responsible and liable for any consequence if violation of this declaration is determined.

Date: 30th June, 2022

Member 1

Hassan Shahzad

18i-0441

Member 2

Sana Ali Khan

18i-0439

Supervisor

Mr. S. Muhammad H. Mustafa

Abstract

Hospitals are famously busy environments that suffer from always being in demand and yet frequently understaffed. The use of manual supervision for everything takes time and energy that could be better directed elsewhere. Our system will allow for that by providing digital monitoring to observe the environment and detect certain abnormalities.

Our system has three main modules: 1) Detecting abnormalities 2) Generating alerts to appropriate staff 3) Providing incident logs. With these, we will be able to provide a complete system for digital surveillance and alert generation in hospitals.

Table of Contents

2. Introduction

HospitalAid is a multi-platform application that aims to assist hospitals by removing the need for manual oversight of patients and hospital staff. It will monitor the hospital environment through the camera feed, notice medical abnormalities/incidents and alert appropriate personnel to address the situation.

Detecting an anomaly will be done purely through computer vision, by performing real-time analysis of the live video feed coming in from the hospital's existing surveillance infrastructure. Our system will have various deep learning models that will be trained to notice particular types of anomalies. Alerts will be sent to hospital staff via a mobile app, and incident statistics will be reported to the hospital administration through a web portal.

This system will integrate into the hospital's existing infrastructure, enable automatic monitoring of patients and staff and create a better, safer environment.

3. Literature Review

- **YOLOv3: An Incremental Improvement**

  YOLO (you only look once) is an object detection algorithm that works best for real-time detection. This paper discusses the improvements that YOLOv3 has brought upon its predecessors – increasing the layers in the neural network, detection at three scales, more bounding boxes and a faster, more accurate algorithm.

- **Face mask detection using YOLOv3 and faster R-CNN models: COVID-19 environment**

  In this paper, two object detection models are suggested as advanced face detection approaches. Both YOLOv3 and faster R-CNN models are trained on a dataset of masked and unmasked people; the article proposes an efficient real-time deep learning-based technique to automate the process of detecting masked faces, where each masked face is identified in real-time with the help of bounding boxes.

- **Patient Monitoring by Abnormal Human Activity Recognition Based on CNN Architecture**

  One research domain for video analysis and understanding is of human action recognition. This paper presents a real-time application of it – monitoring patients and identifying abnormal activities e.g., vomiting and coughing. YOLO network was utilized as the CNN model, and trained upon a dataset of annotated patient videos.

  Monitoring COVID-19 social distancing with person detection and tracking via fine-tuned YOLO v3 and Deep sort techniques

  This article proposes a deep learning-based framework for automating the task of monitoring social distancing using video. The proposed framework utilizes the YOLO v3 object detection model to segregate humans from the background and deep sort approach to track the identified people with the help of bounding boxes and assigned IDs. The results of the YOLO v3 model are further compared with other popular models, e.g., faster region-based CNN (convolution neural network) and single shot detector (SSD). The pairwise vectorized L2 norm is computed based on the three-dimensional feature space obtained by using the centroid coordinates and dimensions of the bounding box. From the experimental analysis, it is observed that the YOLO v3 with deep sort tracking scheme displayed best results to monitor the social distancing in real-time.

4. Project Vision

4.1. Problem Statement

"How to efficiently monitor a hospital 24/7?"

Consider a hospital environment: always busy and frequently understaffed. There are hundreds of things going on at any given moment, and it is impossible for a hospital to ensure everything is running perfectly – every patient attended to, every anomaly noticed and corrected, every employee error noticed and so on.

It is necessary, however, to monitor patients and employees; patients may require attention or urgent care at any moment and it is vital that hospital staff be available at their post when they are required to be present. This is done manually (e.g., a nurse may notice a patient needs medical attention or the manager might observe the reception is empty.) Manual oversight might be the only option for now, but it is unreliable and at risk of human error.

A patient may suffer from an incident e.g., fainting and no-one may be around to notice they need aid. Perhaps a nurse was supposed to be attending their room but all the nurses are absent from their station. Or a doctor may not be wearing their mask while attending a sick person, and falls ill themselves. These are just a few examples of incidents that could have been easily mitigated, had they just been observed quickly and attended to.

Many hospitals, especially private ones, already have some form of digital surveillance. But what do they do with that? Monitor for possible security incidents, perhaps notice any abnormal medical incidents – but all this is done by manually viewing the footage only after the fact.

4.2. Business Opportunity

- **Assisting Hospital Patients/Staff**: Manual oversight of patients and staff is laborious, time-consuming and not completely reliable. With HospitalAid in place, there would be a tireless system using the live video feed to monitor the environment, detect an anomaly and send out alerts.
- **Report of Incident Statistics**: Noticing an incident/anomaly and sending out alerts is obviously beneficial for patients, staff and the hospital as a whole. HospitalAid would not just be doing this, but would also collect data on incident statistics and present them for the administration to see. For example, a particular hospital might learn how many times in a day the reception was empty.

4.3. Objectives

1. Dataset collection
2. Data pre-processing
3. Training models to detect
    a. masked/unmasked faces
    b. people within vision of interest
    c. certain actions/behaviors of people
4. System for viewing incident
5. Generating logs of incidents

4.4. Project Scope

The digital surveillance system in a hospital will observe the hospital environment, staff and patients; the video feed received will be tested by our system. If it detects any sort of abnormality (choking, empty reception, fainting etc.), an alert message will be generated and sent to the concerned staff/personnel (nurses, doctors, security team, administration etc.) Additionally, logs of every abnormality would be maintained and records would be saved in a database with timestamps for administration to view.

4.5. Constraints

- Detecting a person's actions or behavior is difficult if they are at the edge of vision of interest
- Training a model requires significant amount of time, effort and data – adding new actions to detect would be very hard
- Live video feed will need to be passed to the models, requiring a backend server for them to run on
- If the camera quality is poor, then model accuracy will suffer
- Poor lighting in the hospital environment could also lead to incorrect/missing predictions
- A full-time internet connection will be required for forwarding video feed to server

4.6. Stakeholders

- Project leader
- Project developer team
- Product user
- Product user group
- Product testers
- Product maintenance team
- Companies funding us in future

4.7. Target Audience

Target audience includes all hospitals that have a digital surveillance system and want to monitor the hospital environment more easily and extensively.

5. Software Requirements Specification

5.1. List of Features/Functional Requirements

The major functions/features of the system will be:

- **Object Detection:** The live camera feed is analyzed frame-by-frame and passed through trained models to identify particular objects e.g., person

- **Object Classification:** Some objects e.g., faces mean nothing on their own, they will be further classified into right and wrong categories.

- **Activity Recognition:** System will monitor the environment and observe for the occurrences of specific activities i.e., medical abnormalities such as fainting.

- **Alert Generation:** For any abnormality or incident observed, an alert will be sent to the appropriate hospital staff.

- **Incident Report:** Hospital administration will be provided with a report of incident statistics, generated by the system after a set time interval.

5.2. Quality Attributes/Non-functional Requirements

- **Performance:** The various models that the system will use will have at least 80% accuracy.

- **Usability:** Both the web app and the mobile app will have a clean, minimalistic look that will be easy for users to understand and use.

- **Reliability:** The system will be reliable in terms of predicting – using various machine learning techniques, our application will give a good accuracy.

- **Interoperability:** The system will run on new versions of web browsers as well as on the versions which are old.

- **Maintainability:** The system will be easily maintained because it will be built in modules and if a module is to be changed the whole system doesn't change. The backend model can also be updated and replaced easily.

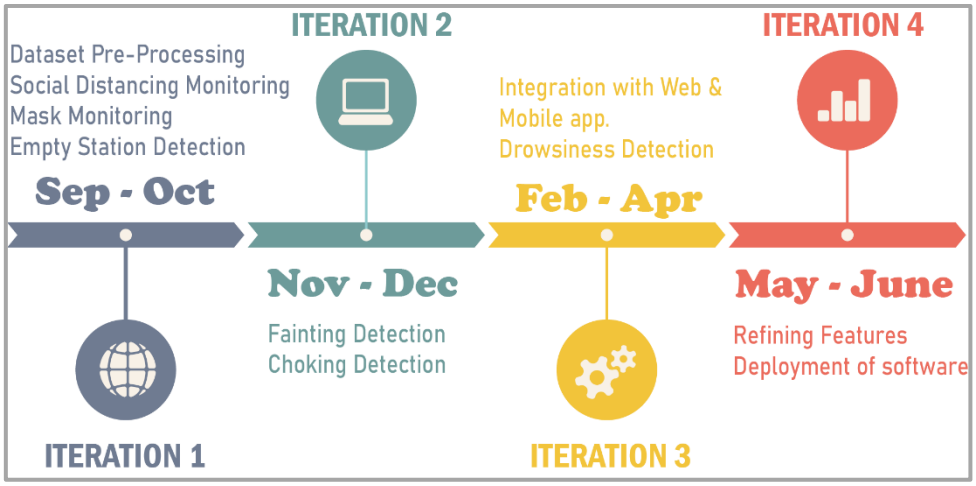
- **Modifiability:** The system will be built in modules in such a way that modules will be independent.

- **Testability:** Thousands of pictures will be used for training each module therefore the system will perform very well under testing environment.

- **Reusability:** Since each module will be independent, therefore with a little change, they can be used in other similar systems. The models we have trained may also be applied to different scenarios using transfer learning.

- **Robustness:** No data is being saved locally so the chances of data loss in worst cases are 0%.

6. Project Timeline



7. High Level Use Cases

7.1. Use Cases List

USE CASE ID	PRIMARY ACTOR	USE CASE
UC-101	Admin	View Logs
UC-102	Admin	View Incident
UC-103	Manager, Nurse	Receive Alert
UC-104	System	Detect Person
UC-105	System	Detect Face
UC-106	System	Detect Hands
UC-107	System	Classify Face
UC-108	System	Classify Hands
UC-109	System	Count People
UC-110	System	Calculate Distances
UC-111	System	Classify Action
UC-112	System	Generate Alert
UC-113	System	Generate Incident Logs
UC-114	System	Generate Report

7.2. Use Cases

Use Case ID:	UC-101
Use Case Name:	View Logs
Actors:	Admin
Type:	Primary
Description:	Admin will be able to view the logs of incidents i.e., the date and time when something abnormal occurred.

Use Case ID:	UC-102
Use Case Name:	View Incident
Actors:	Admin
Type:	Primary
Description:	Admin can view an image/video of any incident detected.

Use Case ID:	UC-103
Use Case Name:	Receive Alert
Actors:	Manager, Nurse
Type:	Primary
Description:	The hospital staff (manager and nurse) will receive an alert on their phones when the system observes an abnormality.

Use Case ID:	UC-104
Use Case Name:	Detect Person
Actors:	System
Type:	Primary
Description:	System will process the camera feed, pass image through trained models and determine if there are people in frame or not.

Use Case ID:	UC-105
Use Case Name:	Detect Face
Actors:	System
Type:	Primary
Description:	System will process the camera feed, pass image through trained models and determine if there are faces in frame.

Use Case ID:	UC-106
Use Case Name:	Detect Hands
Actors:	System
Type:	Primary
Description:	System will process the camera feed, pass image through trained models and determine if there are hands in frame.

Use Case ID:	UC-107
Use Case Name:	Classify Face
Actors:	System
Type:	Primary
Description:	A face detected will be classified as wearing mask or not wearing mask.

Use Case ID:	UC-108
Use Case Name:	Classify Hands
Actors:	System
Type:	Primary
Description:	Hands that are detected will be classified as wearing gloves or not wearing gloves.

Use Case ID:	UC-109
Use Case Name:	Count People
Actors:	System
Type:	Primary
Description:	System will count the number of people in its vision, from a particular video feed.

Use Case ID:	UC-110
Use Case Name:	Calculate Distances
Actors:	System
Type:	Primary
Description:	After detecting multiple people in frame, the distances between them are calculated to determine if they are violating social distancing rules.

Use Case ID:	UC-111
Use Case Name:	Classify Action
Actors:	System
Type:	Primary
Description:	Medical abnormalities e.g., choking, fainting will be observed by the camera and then classified by the system to check if it actually was that action or not.

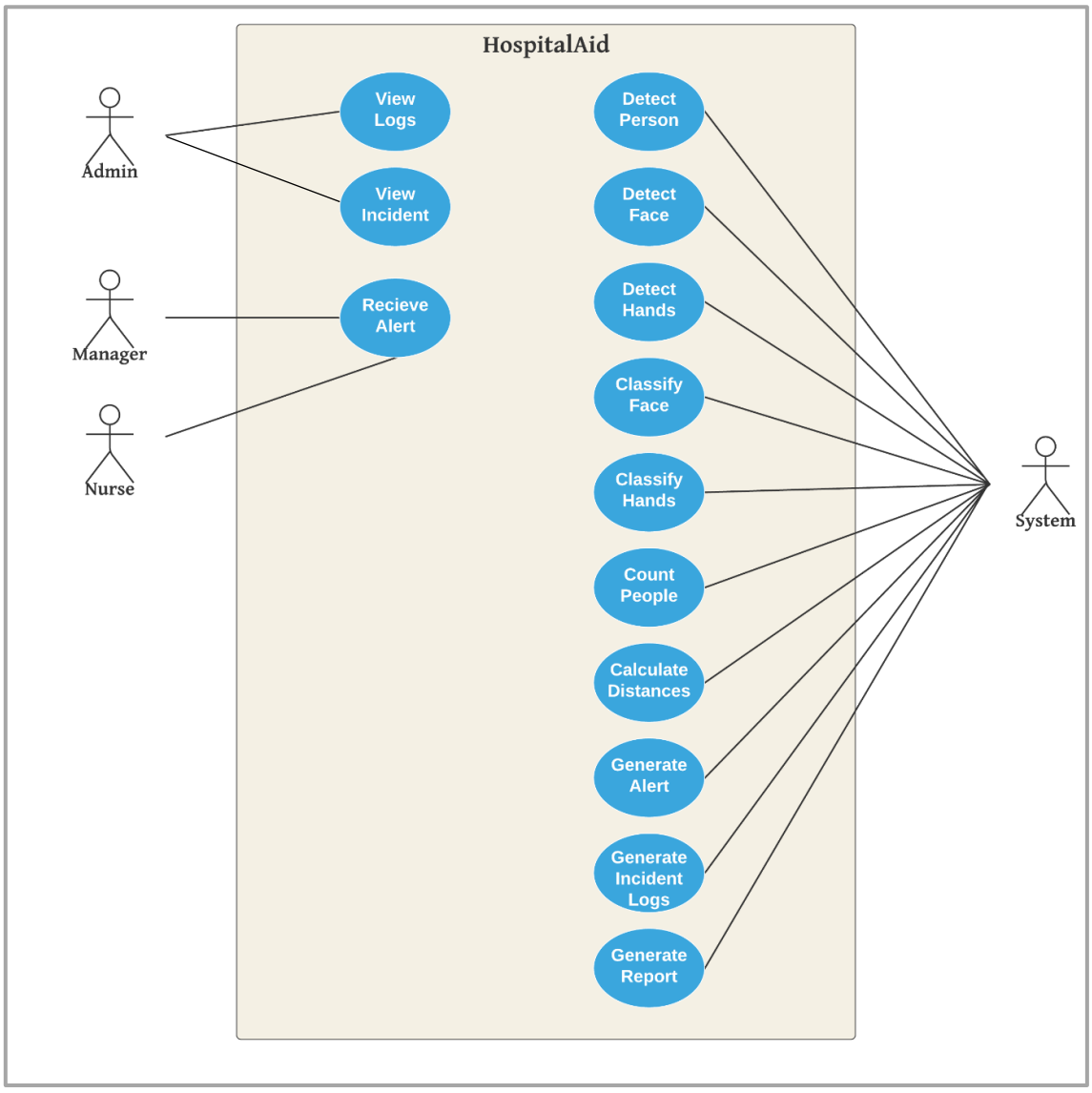
Use Case ID:	UC-112
Use Case Name:	Generate Alert
Actors:	System
Type:	Primary
Description:	After detecting an abnormality or incident, system will generate an alert and send it to hospital staff.

Use Case ID:	UC-113
Use Case Name:	Generate Incident Logs
Actors:	System
Type:	Primary
Description:	After detecting an abnormality or incident, system will generate a record of when it happened and save to database.

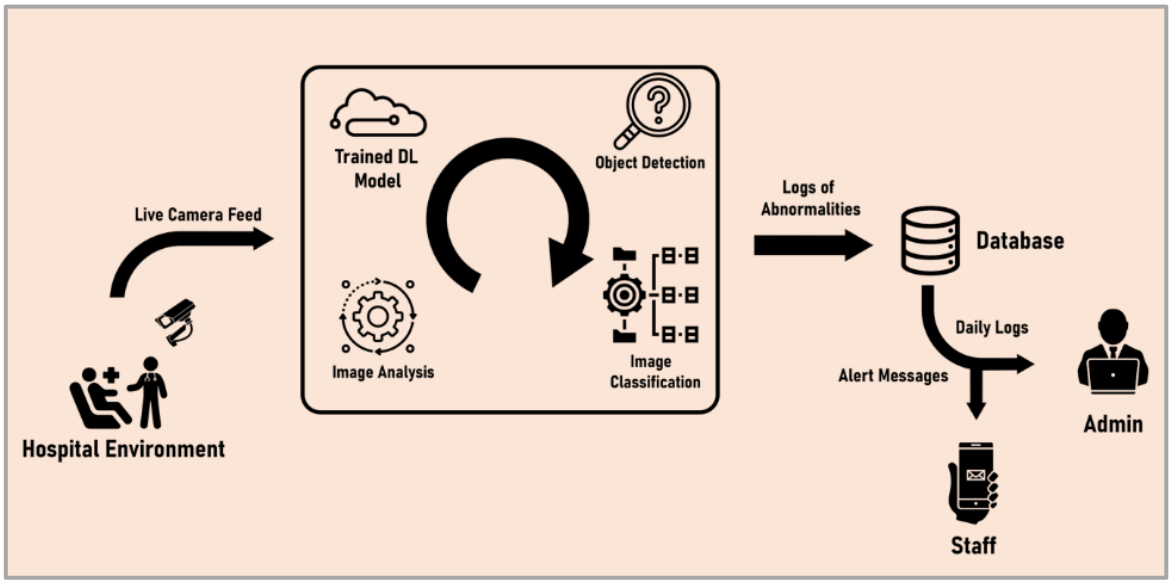
Use Case ID:	UC-114
Use Case Name:	Generate Report
Actors:	System
Type:	Primary
Description:	After a set interval of time e.g., daily, a report will be generated containing the statistics of the incidents that happened during that interval.

Use Case ID:	UC-115
Use Case Name:	Login
Actors:	Admin
Type:	Primary
Description:	Admin can login to the website.

7.3. Use Case Diagram

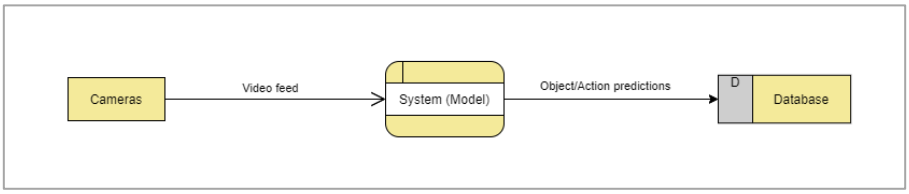


8. Architecture Diagram

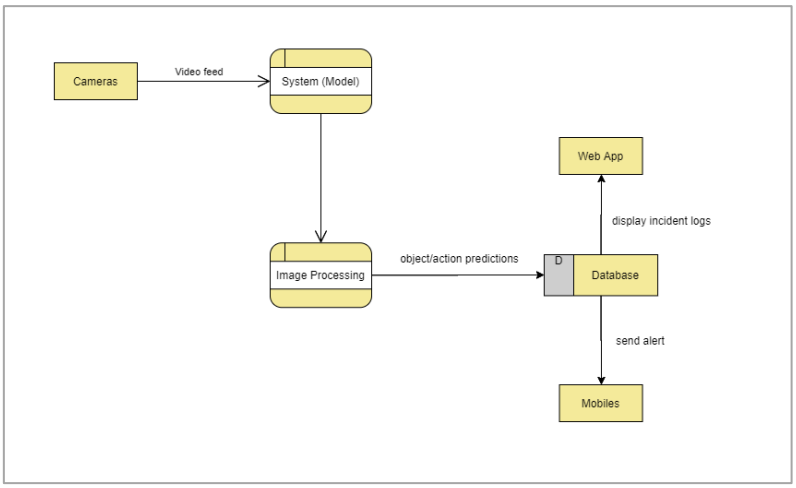


9. Data Flow Diagram

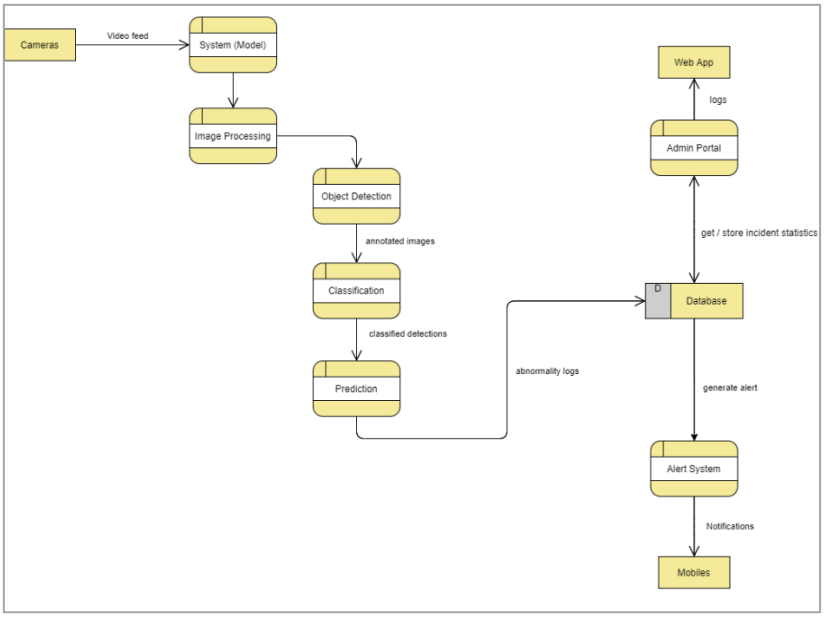
9.1. Level 0



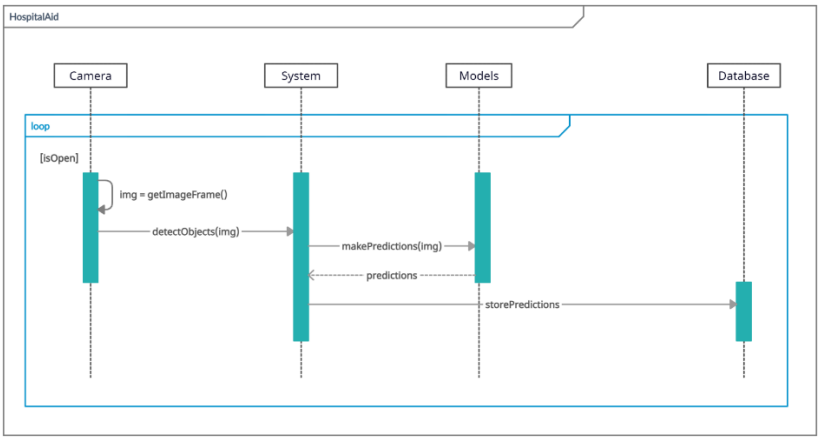
9.2. Level 1



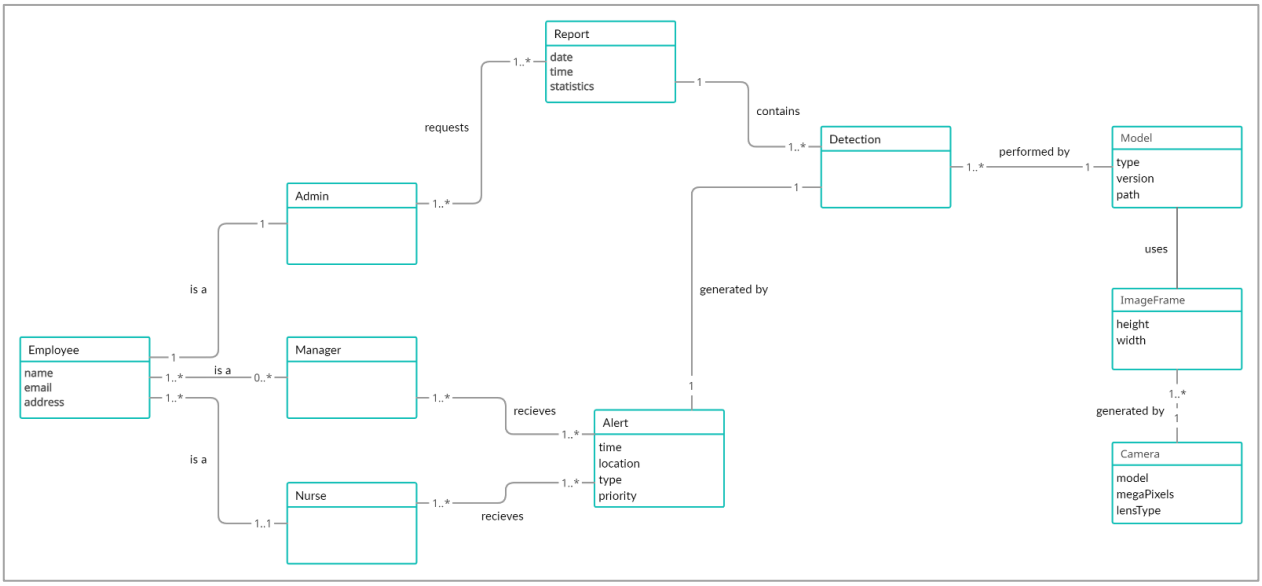
9.3. Level 2



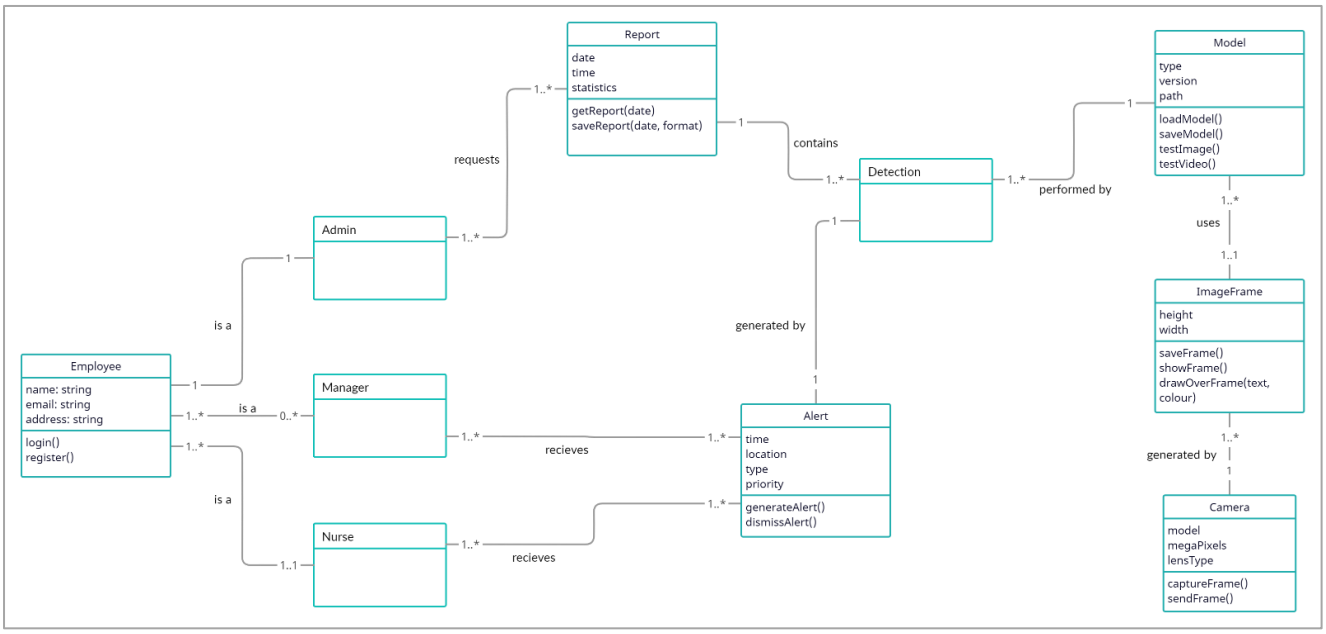
10. Sequence Diagram



11. Domain Model



12. Class Diagram



13. Iteration – 1

13.1. Introduction

In the first iteration, we planned to research and develop some of our simpler features:

- Empty Counter Detection
- Mask SOPs Monitoring
- Social Distance Monitoring

For each feature we had to collect and preprocess our dataset - we used a mix of datasets online, and also filmed our own dataset in an actual hospital environment. All the raw data was appropriately labelled, augmented and saved.

For each feature, we trained different models:

- Empty counter detection required us to check whether there is at least one person present at the area being monitored. We filmed our own dataset of hospital nursing counters and receptions, with a variable number of people. The dataset was then labelled to refer to the people inside the frame. Our model was trained on this dataset so it could predict the presence of a person.

- Mask SOPs Monitoring was done in two phases. The first part required us to identify a face in the image, and the second part was classifying that face as wearing a mask or not. We used a built-in python framework in open-cv to isolate faces, and passed them through a model which was trained on masked and unmasked faces.

- Social distance monitoring used a similar model as in empty counter detection. The goal was to detect people in frame, calculate the distances between them and mark them as either appropriately distanced or not.

13.2. Expanded Use Cases

Use Case ID:	UC-104		
Use Case Name:	Detect Person		
Created By:	Team HospitalAid	Last Updated By:	Team HospitalAid
Date Created:	Oct 20, 2021	Last Revision Date:	Oct 22, 2021

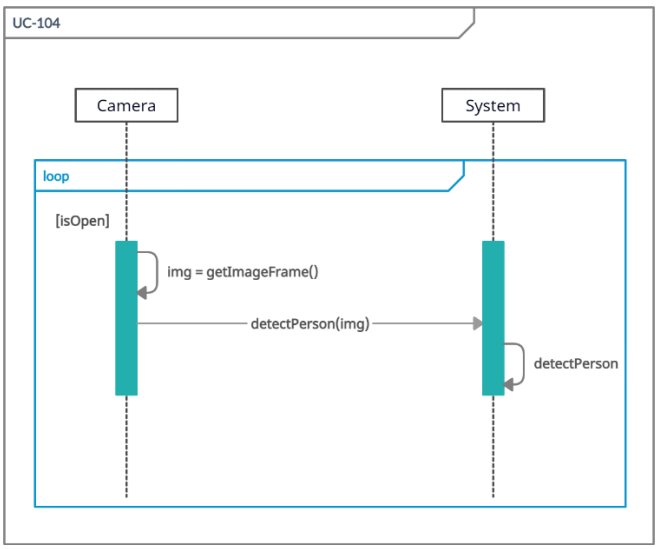
Actors:	System (Primary)	
Description:	System will process the live camera feed, pass image through trained models and determine if there are people in frame or not.	
Pre-conditions:	• Camera should be working • Person(s) is visible in the vision of interest of the camera	
Post-conditions:	System detects the person(s) in the vision of interest of the camera.	
Main Success Scenario:	Actor 1. Person comes in vision of interest of camera	System 2. System detects the person
Alternative Flows:		
Exceptions:	• Camera is not working • Person is at edge of camera's vision of interest • Camera quality is very poor • Lighting is very dim	
Frequency of Use:	This may be used for further development where person detection is required.	
Special Requirements:	Sufficient lighting is required for the camera to view the persons(s) clearly, and the camera should be of adequate quality.	
Assumption:	Camera is working.	

Use Case ID:	UC-105		
Use Case Name:	Detect Face		
Created By:	Team HospitalAid	Last Updated By:	Team HospitalAid
Date Created:	Oct 20, 2021	Last Revision Date:	Oct 22, 2021

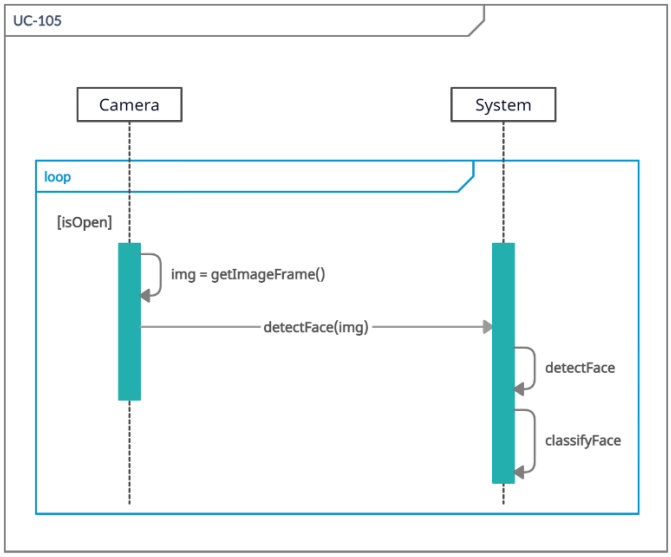
Actors:	System (Primary)	
Description:	System will process the live camera feed, pass image through trained models and determine if there are faces in frame or not.	
Pre-conditions:	• Camera should be working • Human face(s) is visible in the vision of interest of the camera • Faces should not be at very edge of interest or very far from camera	
Post-conditions:	System detects the face(s) in the vision of interest of the camera.	
Main Success Scenario:	Actor 1. Person comes in vision of interest of camera	System 2. System detects the person's face
Alternative Flows:		
Exceptions:	• Camera is not working • Person is at edge of camera's vision of interest • Camera quality is very poor • Lighting is very dim	
Frequency of Use:	This may be used for further development where facial detection is required.	
Special Requirements:	Sufficient lighting is required for the camera to view the face (s) clearly, and the camera should be of adequate quality.	
Assumption:	Camera is working.	

13.3. System Sequence Diagrams

13.3.1. Use case id: UC-104



13.3.2. Use case id: UC-105



13.4. Operation Contracts

13.4.1.Contract use case id: UC-104

Operation:	Detect Person
Cross References:	UC-105 Detect Face, UC-109 Count People, UC-110 Calculate Distances
Pre-Conditions: • The camera is working. • A person is visible in the camera's vision of interest.	
Post Conditions: • Person is detected correctly. • Face is being detected too • Number of people is added into the logs • Distances are calculated	

13.4.2.Contract use case id: UC-105

Operation:	Detect Face
Cross References:	UC-107 Classify Face
Pre-Conditions: • The camera is working. • A person is completely inside the frame. • Face is completely visible. • Lighting conditions are good.	
Post Conditions: • Face is classified correctly. • Log is maintained	

14. Implementation details

As we had three main features for this iteration, hence, the implementation details for each of the feature is as follows:

14.1. Empty Counter Detection

The implementation of empty counter detection was divided into two phases:

- Dataset Pre-Processing
- Training and Testing of Model

14.1.1.Dataset Pre-processing:

For the pre-processing of dataset, we first needed to gather the dataset. For this, we used CCTV video feeds from the hospitals along with some self-recorded videos. The pre-processing steps are as follows:

- **Division:**
    - **Train Set** = 12K images
    - **Valid Set** = 509 images
    - **Test Set** = 99 images
- **Pre-Processing:**
    - Auto-Orientation
    - **Resize** = Stretch to 128x128
- **Augmentations:**
    - **Flip** = Horizontal
    - **Rotation** = Between -5° and +5°
    - **Grayscale** = Apply to 10% of images
    - **Hue** = Between -10° and +10°
    - **Saturation** = Between -7% and +7%
    - **Brightness** = Between -9% and +9%
    - **Exposure** = Between -6% and +6%

14.1.2. Training and testing of model

Once we were done with the pre-processing of dataset, the next thing we needed to do was to train our model and design an implementation logic for our model. The logic that we decided was that if even a single person is detected on the counter, then this will be a normal activity for us but if our model is unable to detect any person for a reserved period of time (>= 5 mins), then this will be considered as an abnormal activity and an alert will be generated along with the log being maintained in a file.

Some key points of our implementation are as follows:

- **Framework used:**
    - PyTorch
    - YOLOv5
    - GitHub of Ultralytics

- **System Configurations:**
  - **Processor:** Core i5 7th Generation
  - **RAM:** 64GB
  - **GPU:** 2 – NVIDIA GTX 1070 SLI
  - **OS:** Ubuntu 20.04
  - **Language:** Python 3.8.0
- **Training:**
  - All images cached before the first epoch.
  - **Epochs:** Ran for 52 Epochs. (Started overfitting after that)
  - **Accuracy:** 97.3
  - **Avg Loss:** 0.01593
- **Testing:**
  - Used random videos from hospital and manually recorded ones.
- **Threshold used:** 0.4

Fig 13.1.2.1: When the counter is not empty

Fig 13.1.2.2: When the counter is empty

14.2. Face Mask Monitoring

The implementation of face mask detection was divided into two phases:

- Dataset Pre-Processing
- Training and Testing of Model

14.2.1.Dataset Pre-processing:

For the pre-processing of dataset, we first needed to gather the dataset. For this, we used datasets from Kaggle and Roboflow. The pre-processing steps are as follows:

- **Division:**
    - **Train Set** = 815 images
    - **Valid Set** = 129 images
    - **Test Set** = 50 images
- **Pre-Processing:**
    - Auto-Orientation
    - **Resize** = Stretch to 128x128
- **Augmentations:**
    - **Flip** = Horizontal
    - **Rotation** = Between -5° and +5°
    - **Grayscale** = Apply to 10% of images
    - **Hue** = Between -10° and +10°
    - **Saturation** = Between -7% and +7%
    - **Brightness** = Between -9% and +9%
    - **Exposure** = Between -6% and +6%

14.2.2.Training and testing of model

Once we were done with the pre-processing of dataset, the next thing we needed to do was to train our model and design an implementation logic for our model. The logic that we decided was that we had 2 classes : Mask and No-Mask. Our model detects faces and then passes them through the classifier. It will predict the class above a certain threshold which we had set to 0.5 for better predictions.

Some key points of our implementation are as follows:

- **Framework used:**
    - PyTorch
    - YOLOv5
    - GitHub of Ultralytics
- **System Configurations:**
    - **Processor:** Core i5 7th Generation
    - **RAM:** 64GB
    - **GPU:** 2 – NVIDIA GTX 1070 SLI

- **OS:** Ubuntu 20.04
- **Language:** Python 3.8.0

- **Training:**
  - All images cached before the first epoch.
  - **Epochs:** Ran for 34 Epochs. (Started overfitting after that)
  - **Accuracy:** 91.2
  - **Avg Loss:** 0.493
- **Testing:**
  - Manually asked some fellow classmates to help us in recording the demo video.
- **Threshold used:** 0.5

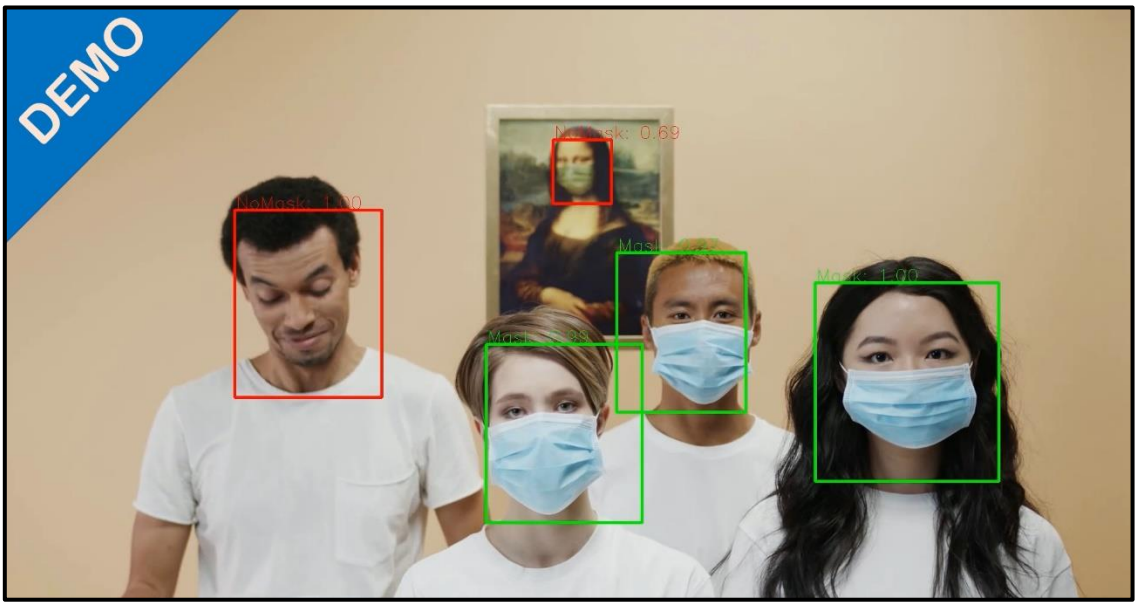


Fig 13.2.2.1: Face Mask Monitoring Output

14.3. Social Distancing Monitoring

The implementation of social distancing monitoring was divided into two phases:

- Dataset Pre-Processing
- Training and Testing of Model

14.3.1.Dataset Pre-processing:

For the pre-processing of dataset, we first needed to gather the dataset. For this, we used Microsoft's COCO dataset . The pre-processing steps are as follows:

- **Division:**
    - **Train Set** = 118K images
    - **Valid Set** = 5K images
    - **Test Set** = 41K images
- **Pre-Processing:**
    - Auto-Orientation
    - **Resize** = Stretch to 128x128
- **Augmentations:**
    - **Flip** = Horizontal
    - **Rotation** = Between -5° and +5°
    - **Grayscale** = Apply to 10% of images
    - **Hue** = Between -10° and +10°
    - **Saturation** = Between -7% and +7%
    - **Brightness** = Between -9% and +9%
    - **Exposure** = Between -6% and +6%

14.3.2.Training and testing of model

Once we were done with the pre-processing of dataset, the next thing we needed to do was to train our model and design an implementation logic for our model. The logic that we decided was that first of all we needed to detect every person in the given frame. So, for this we used COCO's pre-trained weights to detect people. Once every person was detected, we were calculating the center point of each person. Then the next step was to calculate the Euclidian Distance between two center points. If the distance is less than the set threshold i.e., 150, then it will be marked as a violation of social distancing, else it's perfectly normal.

Some key points of our implementation are as follows:

- **Framework used:**
    - PyTorch
    - YOLOv5
    - GitHub of Ultralytics
- **System Configurations:**
    - **Processor:** Core i5 7th Generation

- **RAM:** 64GB
- **GPU:** 2 – NVIDIA GTX 1070 SLI
- **OS:** Ubuntu 20.04
- **Language:** Python 3.8.0

- **Testing:**
  - Used demo video from : https://www.epfl.ch/labs/cvlab/data/data-pom-index-php/ and some self-recorded videos.

- **Threshold used:** 0.5

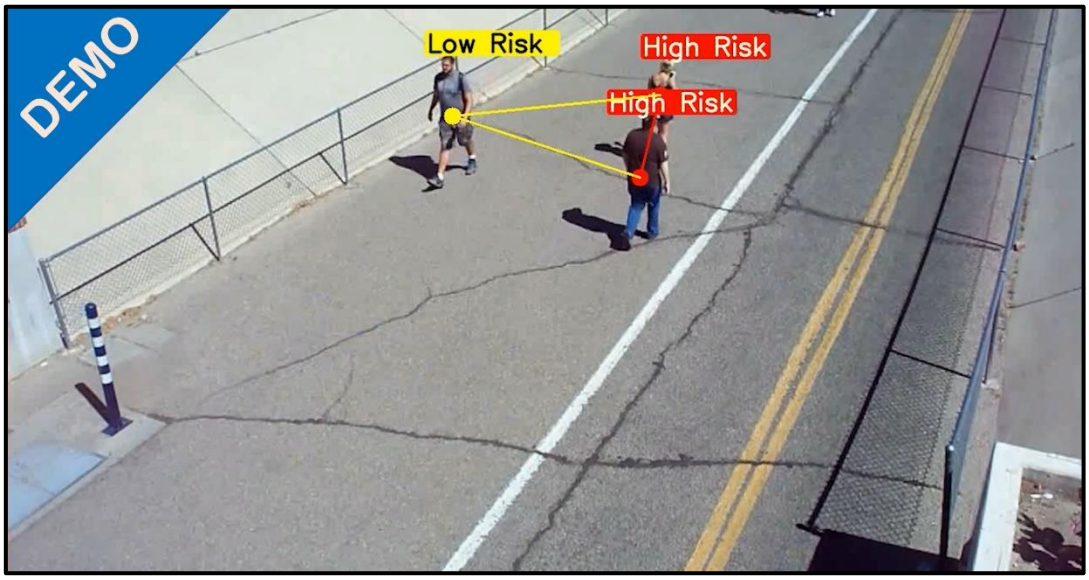


Fig 13.3.2.1: Real-time social distancing monitoring

15. Iteration – 2

15.1. Introduction

In the second iteration, we planned to research and develop some of our complex features:

- Choking Detection
- Fainting Detection

For each feature we had to collect and preprocess our dataset - we used a mix of datasets online (some public and other filmed in a research collaboration of several universities), and also filmed our own dataset. All the raw data was appropriately labelled, augmented and saved.

For each feature, we trained different models:

- Choking detection requires the system to be able to predict from an image if a person is choking or not. Naturally, an incident like this required immediate medical attention. Our model would be able to identify the activity, and an alert would be sent to the hospital staff.
- Fainting detection requires the system to be able to predict from an image if a person is choking or not. Someone fainting (within the camera vision) would be noticed immediately and reported to the appropriate medical personnel.

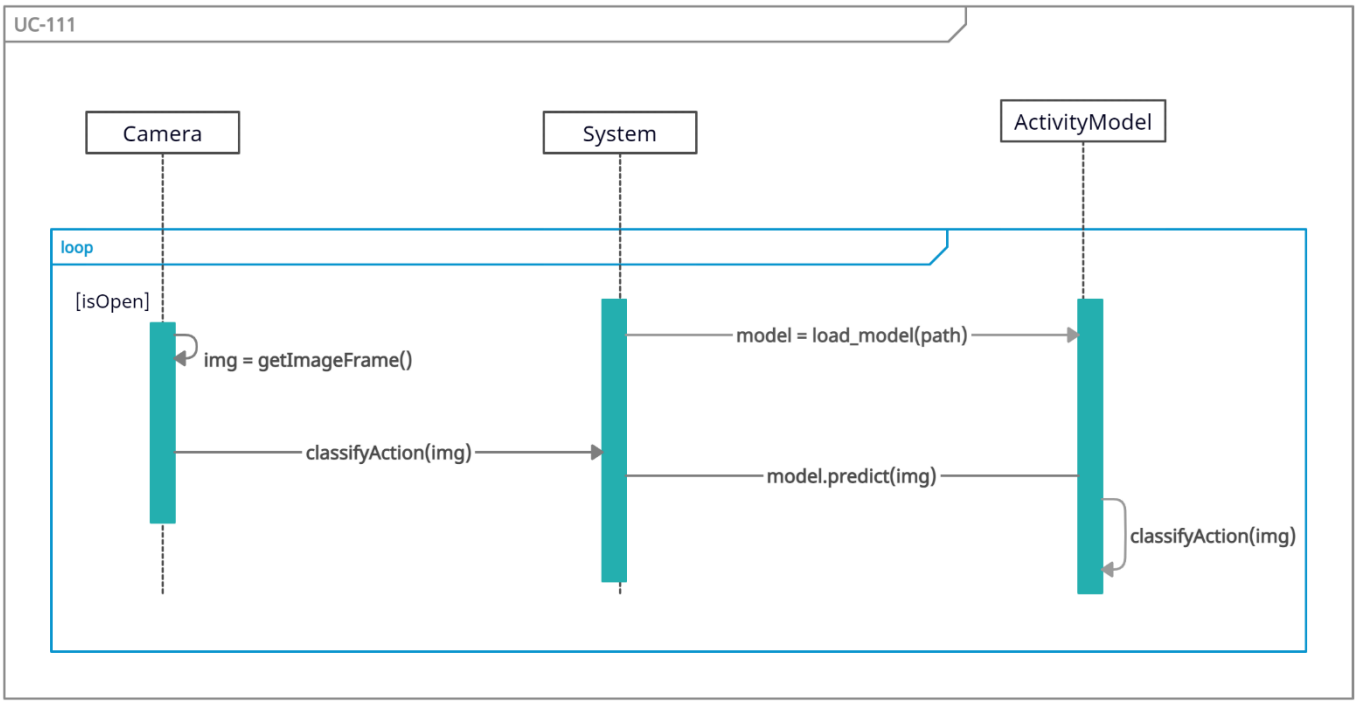
15.2. Expanded Use Cases

Use Case ID:	UC-111		
Use Case Name:	Classify Action		
Created By:	Team HospitalAid	Last Updated By:	Team HospitalAid
Date Created:	Dec 10, 2021	Last Revision Date:	Dec 14, 2021

Actors:	System (Primary)	
Description:	System will process the live camera feed, pass image through trained models and determine if the specified action(s) are occurring or not.	
Pre-conditions:	• Camera should be working • Person(s) is visible in the vision of interest of the camera • Majority of person(s)' body is visible in the vision of interest of the camera	
Post-conditions:	System detects if the specified activity is being performed in the vision of interest of the camera.	
Main Success Scenario:	Actor 1. Person comes in vision of interest of camera	System 2. System classifies the person as performing the specified action or not
Exceptions:	• Camera is not working • Person is at edge of camera's vision of interest • Camera quality is very poor • Lighting is dim • The person performing the action is being obscured	
Frequency of Use:	This is a generic feature which may be used for further development of action classification.	
Special Requirements:	Sufficient lighting is required for the camera to view the persons(s) clearly, and the camera should be of adequate quality.	
Assumption:	Camera is working.	

15.3. System Sequence Diagrams

15.3.1.Use case id: UC-111



15.4. Operation Contracts

15.4.1.Contract use case id: UC-111

Operation:	Classify Action
Cross References:	
Pre-Conditions: • The camera is working. • A person is visible in the camera's vision of interest. • Majority of person(s)' body is visible in the vision of interest of the camera	
Post Conditions: • The action is observed • Action is classified correctly • Incident time is noted in the logs	

15.5. Choking Detection

The implementation of choking detection was divided into two phases:

- Dataset Pre-Processing
- Training and Testing of Model

15.5.1.Dataset Pre-processing:

For the pre-processing of dataset, we first needed to gather the dataset. For this, we gathered some data from a research paper[1] (conducted in collaboration with several universities) and some were recorded in a controlled environment. The pre-processing steps are as follows:

- **Division:**
  - **Train Set** = 7K images
  - **Valid Set** = 800 images
  - **Test Set** = 1.5K images
- **Pre-Processing:**
  - Auto-Orientation
  - **Resize** = Stretch to 128x128
- **Augmentations:**
  - **Flip** = Horizontal
  - **Rotation** = Between -5° and +5°
  - **Grayscale** = Apply to 10% of images
  - **Hue** = Between -10° and +10°
  - **Saturation** = Between -7% and +7%
  - **Brightness** = Between -9% and +9%
  - **Exposure** = Between -6% and +6%

15.5.2.Training and testing of model

Once we were done with the pre-processing of dataset, the next thing we needed to do was to design an implementation logic for our model and train it. For this purpose, we used a convolutional neural network. However, there was a major issue present - **flickering**. As our program was reading almost 30 frames in a single second, it was applying the predictions on each frame, which was not only computationally expensive but was inaccurate too. This is every frame is classified completely independent of the rest, which meant several consecutive frames might be classified differently and the classification of the action being performed would appear to 'flicker'. The solution we came up with was using **queues** to implement a rolling average over predictions. We specify the number of frames in the start which are then stored in the queue. We then apply predictions on each of the frame in the queue and then take the average of the predictions. The answer is then displayed on the screen.

[1] https://www.mdpi.com/2079-9292/9/12/1993/htm

The overall algorithm can be summarized as:

- Loop over all frames in the video file
- For each frame, pass the frame through the CNN
- Obtain the predictions from the CNN
- Maintain a list of the last K predictions
- Compute the average of the last K predictions and choose the label with the largest corresponding probability
- Label the frame and write the output frame to disk

Fig 13.4.2.1: When a person is not choking

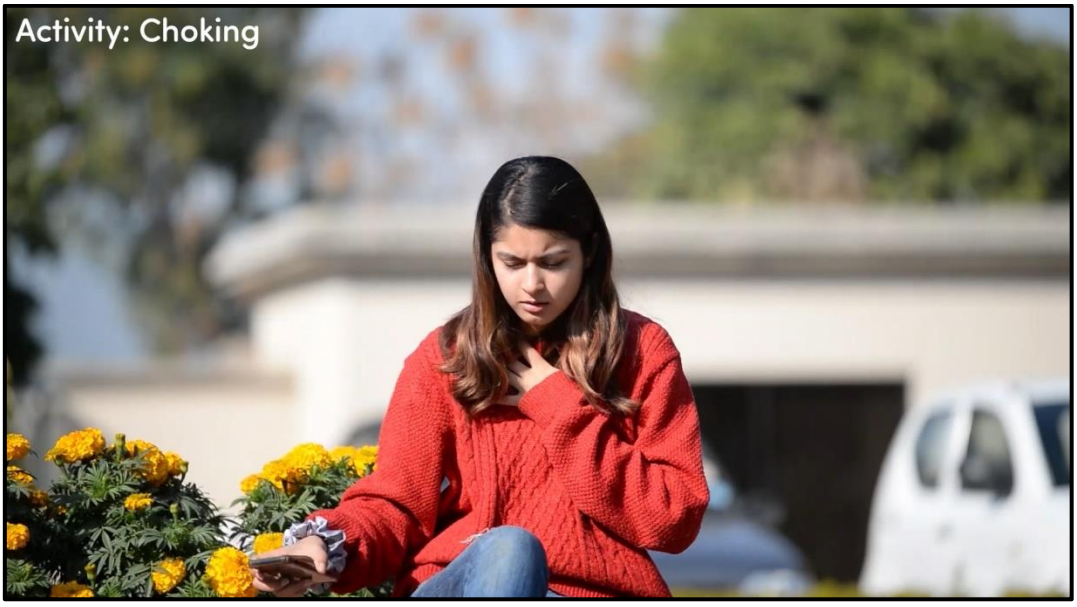


Fig 13.4.2.2: When a person is choking

15.6. Fainting Detection

The implementation of fainting detection was divided into two phases:

- Dataset Pre-Processing
- Training and Testing of Model

15.6.1.Dataset Pre-processing:

For the pre-processing of dataset, we first needed to gather the dataset. For this, we gathered some dataset from a research paper mentioned in the references and some were recorded in a controlled environment. The pre-processing steps are as follows:

- **Division:**
  - **Train Set** = 24K images
  - **Valid Set** = 5K images
  - **Test Set** = 3K images
- **Pre-Processing:**
  - **Resize** = Stretch to 128x128
- **Augmentations:**
  - **Grayscale** = Apply to 15% of images
  - **Hue** = Between -5° and +5°
  - **Saturation** = Between -5% and +5%
  - **Brightness** = Between -5% and +5%
  - **Exposure** = Between -5% and +5%

15.6.2.Training and testing of model

Once we were done with the pre-processing of dataset, the next thing we needed to do was to design an implementation logic for our model and train it. For this purpose, we used a convolutional neural network. However, there was a major issue present - **flickering**. As our program was reading almost 30 frames in a single second, it was applying the predictions on each frame, which was not only computationally expensive but was inaccurate too. This is every frame is classified completely independent of the rest, which meant several consecutive frames might be classified differently and the classification of the action being performed would appear to 'flicker'. The solution we came up with was using **queues** to implement a rolling average over predictions. We specify the number of frames in the start which are then stored in the queue. We then apply predictions on each of the frame in the queue and then take the average of the predictions. The answer is then displayed on the screen.

The overall algorithm can be summarized as:

- Loop over all frames in the video file
- For each frame, pass the frame through the CNN
- Obtain the predictions from the CNN
- Maintain a list of the last K predictions
- Compute the average of the last K predictions and choose the label with the largest corresponding probability
- Label the frame and write the output frame to disk

Fig 13.5.2.1: When a person is not fainting

Fig 13.5.2.2: When a person is fainting

16. Iteration – 3 & 4

16.1. Introduction

In the third iteration, we had planned one feature:

- Drowsiness Detection

However, we were able to incorporate iteration 4 features into iteration 3:

- Website Development

Due to this, by the time iteration 4 started, our project was already complete – iteration 4 had no features developed then, and only testing and documentation was done.

For drowsiness detection, we had to collect and preprocess our dataset. All the data was already labelled, but we cleaned the dataset to remove irrelevant images. Then we trained our model on the final dataset.

After completing all our machine learning features, we developed a complete website to interact with out models. Initially website frontend was built, then backend to utilize our models, then frontend was integrated with backend.

16.2. Expanded Use Cases

Use Case ID:	UC-101		
Use Case Name:	View Logs		
Created By:	Team HospitalAid	Last Updated By:	Team HospitalAid
Date Created:	June 10, 2022	Last Revision Date:	June 30, 2022

Actors:	Admin (Primary)	
Description:	Admin will be able to view logs of incidents i.e., date and time when it was detected	
Pre-conditions:	• Admin should be logged in • Database should be online • Internet connection required	
Post-conditions:	Admin views logs of incidents detected.	
Main Success Scenario:	Actor 1. Admin accesses website and enters credentials 4. Admin views incident logs	System 2. System authenticates admin 3. System gets incident data from database and sends to website
Exceptions:	• Internet connection not working • Database not online • Backend server not online	
Frequency of Use:	This will be used very frequently; every time admin logs in and accesses surveillance page.	
Special Requirements:	Internet connection is required for system to get data from database and send to website.	
Assumption:	• Internet connection is working • Database is online	

Use Case ID:	UC-107		
Use Case Name:	Classify Face		
Created By:	Team HospitalAid	Last Updated By:	Team HospitalAid
Date Created:	June 12, 2022	Last Revision Date:	June 30, 2022

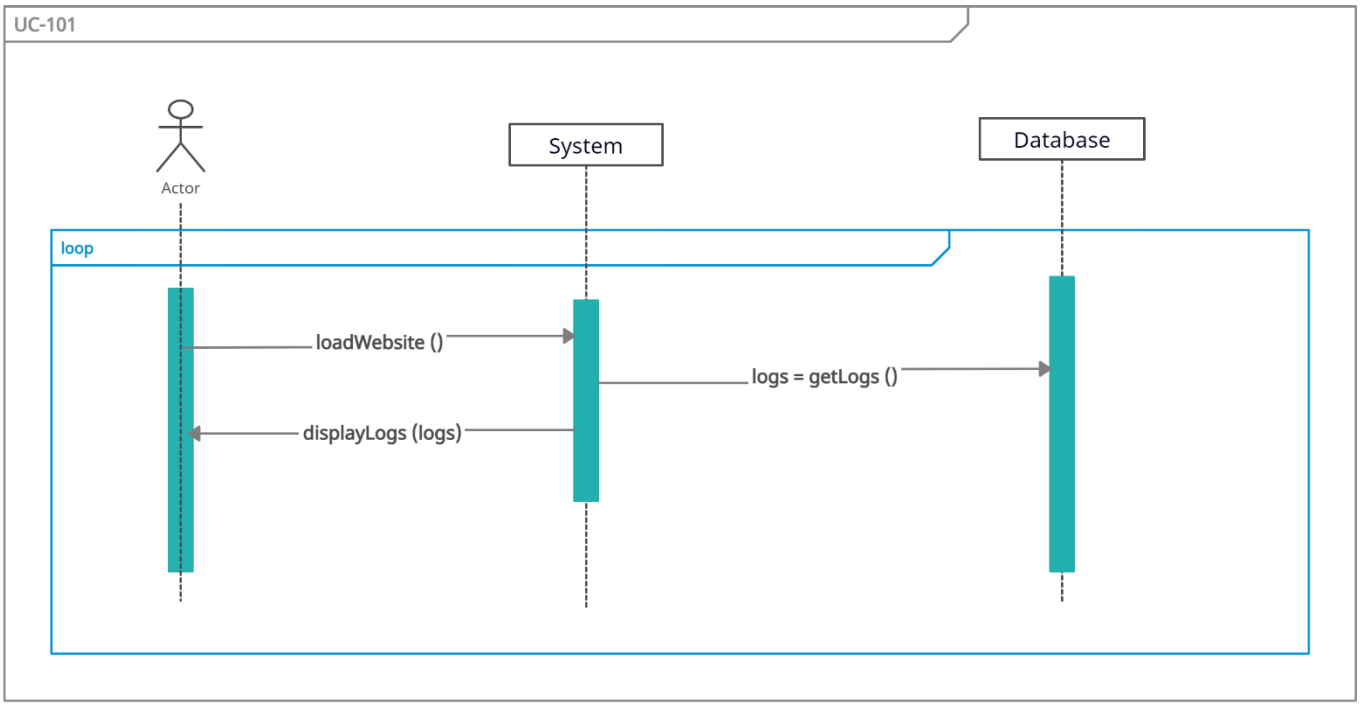
Actors:	System (Primary)	
Description:	System will process the live camera feed, pass image through trained models and determine if the specified action(s) are occurring or not.	
Pre-conditions:	• Camera should be working • Face(s) is visible in the vision of interest of the camera • Majority portion of face(s) is visible in the vision of interest of the camera	
Post-conditions:	System detects if a face is present and performing the specified activity in the vision of interest of the camera.	
Main Success Scenario:	Actor 1. Person's face comes in vision of interest of camera	System 2. System detects face 3. System classifies face as performing the specified action or not
Exceptions:	• Camera is not working • Face is at edge of camera's vision of interest • Camera quality is very poor • Lighting is dim • The faces in the frame are being obscured	
Frequency of Use:	This is a generic feature which may be used for further development of action classification of faces.	
Special Requirements:	Sufficient lighting is required for the camera to view the faces(s) clearly, and the camera should be of adequate quality.	
Assumption:	Camera is working.	

Use Case ID:	UC-113		
Use Case Name:	Generate Incident Logs		
Created By:	Team HospitalAid	Last Updated By:	Team HospitalAid
Date Created:	June 15, 2022	Last Revision Date:	June 30, 2022

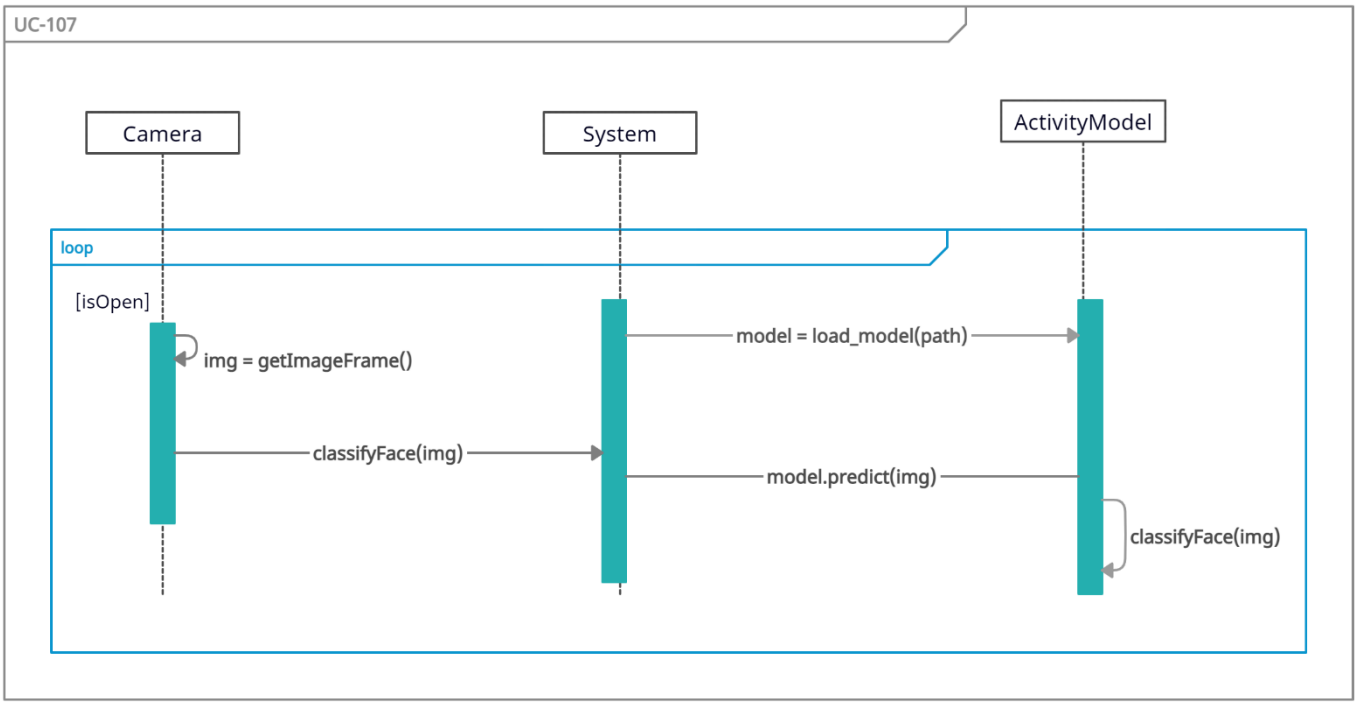
Actors:	System (Primary)	
Description:	After detecting an abnormality or incident, system will generate a record of when it happened and save to database.	
Pre-conditions:	• Backend server should be running • Database should be online • Internet connection working	
Post-conditions:	System generates log of incident and saves it to database.	
Main Success Scenario:	Actor 1. Performs action/activity	System 2. System detects incident or abnormality 3. System saves record of it to database with current date and time
Exceptions:	• Database is not online • Backend server is down • Internet connection not working	
Frequency of Use:	This is expected to be performed frequently; every time an abnormality or incident is detected.	
Special Requirements:	Internet connection is required to send and save data in database.	
Assumption:	• Database is online • Backend server is running • Internet connection is working	

16.3. System Sequence Diagrams

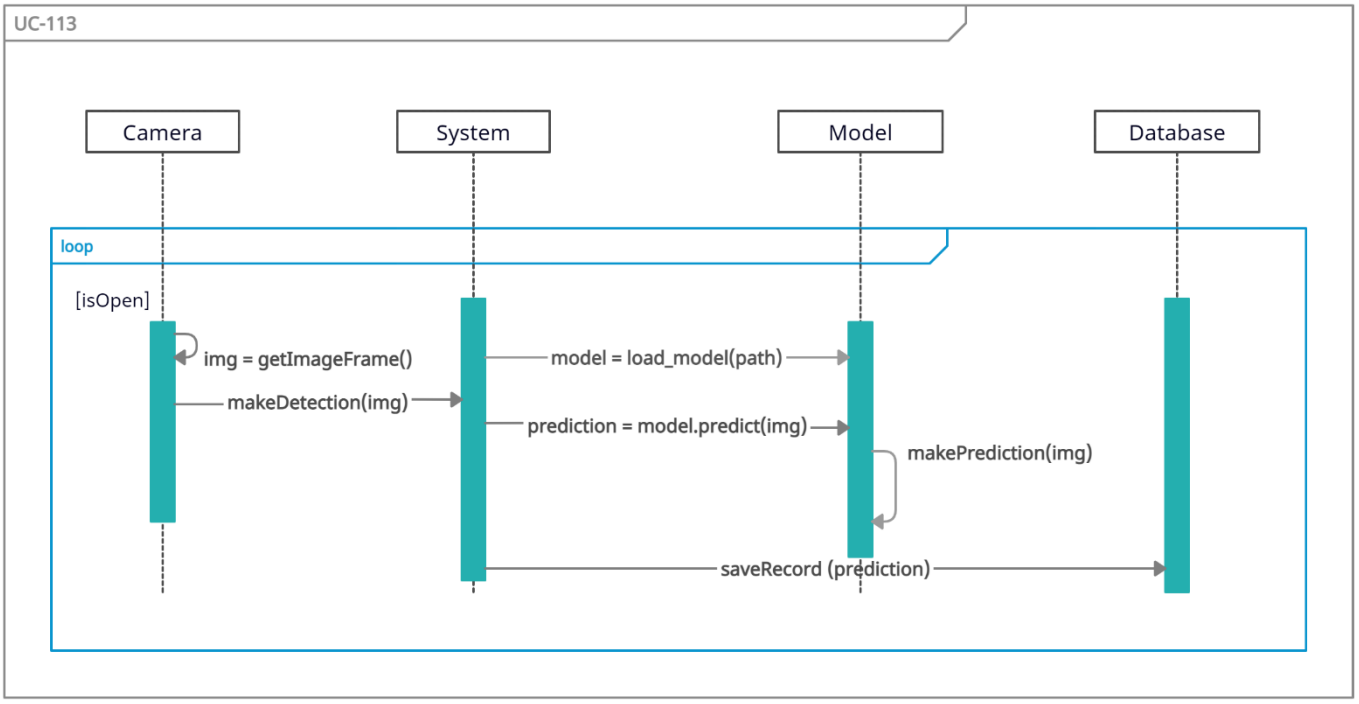
16.3.1. Use case id: UC-101



16.3.2. Use case id: UC-107



16.3.3.Use case id: UC-113



16.4. Operation Contracts

16.4.1.Contract use case id: UC-101

Operation:	View Logs
Cross References:	UC-113 Generate Incident Logs, UC-115 Login
Pre-Conditions: • Admin should be logged in • Database should be online • Internet connection required	
Post Conditions: • Admin views logs of incidents detected.	

16.4.2.Contract use case id: UC-107

Operation:	Classify Face
Cross References:	UC-105 Detect Face
Pre-Conditions: • Camera should be working • Face(s) is visible in the vision of interest of the camera • Majority portion of face(s) is visible in the vision of interest of the camera	
Post Conditions: • System detects if a face is present and performing the specified activity in the vision of interest of the camera.	

16.4.3.Contract use case id: UC-113

Operation:	Generate Incident Logs
Cross References:	UC-113 Generate Incident Logs
Pre-Conditions: • Backend server should be running • Database should be online • Internet connection working	
Post Conditions: • System generates log of incident and saves it to database.	

16.5. Drowsiness Detection

For the pre-processing of dataset, we first needed to gather the dataset. For this, we gathered some dataset from a research paper mentioned in the references (Ghoddosian, 2019). The pre-processing steps are as follows:

- **Division:**
  - **Train Set** = 70K images
  - **Valid Set** = 20K images
  - **Test Set** = 10K images
- **Pre-Processing:**
  - **Resize** = Stretch to 128x128
- **Augmentations:**
  - **Grayscale** = Apply to 15% of images
  - **Hue** = Between -5° and +5°
  - **Saturation** = Between -5% and +5%
  - **Brightness** = Between -5% and +5%
  - **Exposure** = Between -5% and +5%

16.5.1.Training and testing of model

Once we were done with the pre-processing of dataset, the next thing we needed to do was to design an implementation logic for our model and train it. For this purpose, we used a convolutional neural network. However, there was a major issue present - **flickering**. As our program was reading almost 30 frames in a single second, it was applying the predictions on each frame, which was not only computationally expensive but was inaccurate too. This is every frame is classified completely independent of the rest, which meant several consecutive frames might be classified differently and the classification of the action being performed would appear to 'flicker'. The solution we came up with was using **queues** to implement a rolling average over predictions. We specify the number of frames in the start which are then stored in the queue. We then apply predictions on each of the frame in the queue and then take the average of the predictions. The answer is then displayed on the screen.

The overall algorithm can be summarized as:

- Loop over all frames in the video file
- For each frame, pass the frame through the CNN
- Obtain the predictions from the CNN
- Maintain a list of the last K predictions
- Compute the average of the last K predictions and choose the label with the largest corresponding probability
- Label the frame and write the output frame to disk

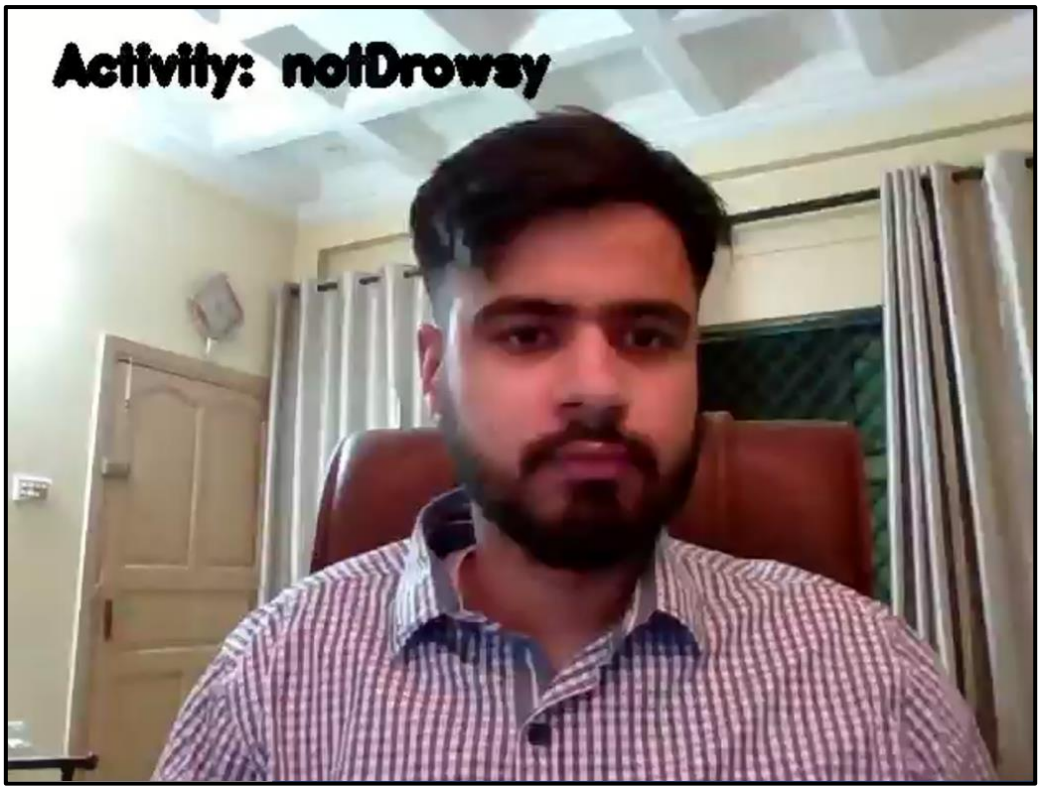


Fig 16.5.1.1: When a person is not drowsy

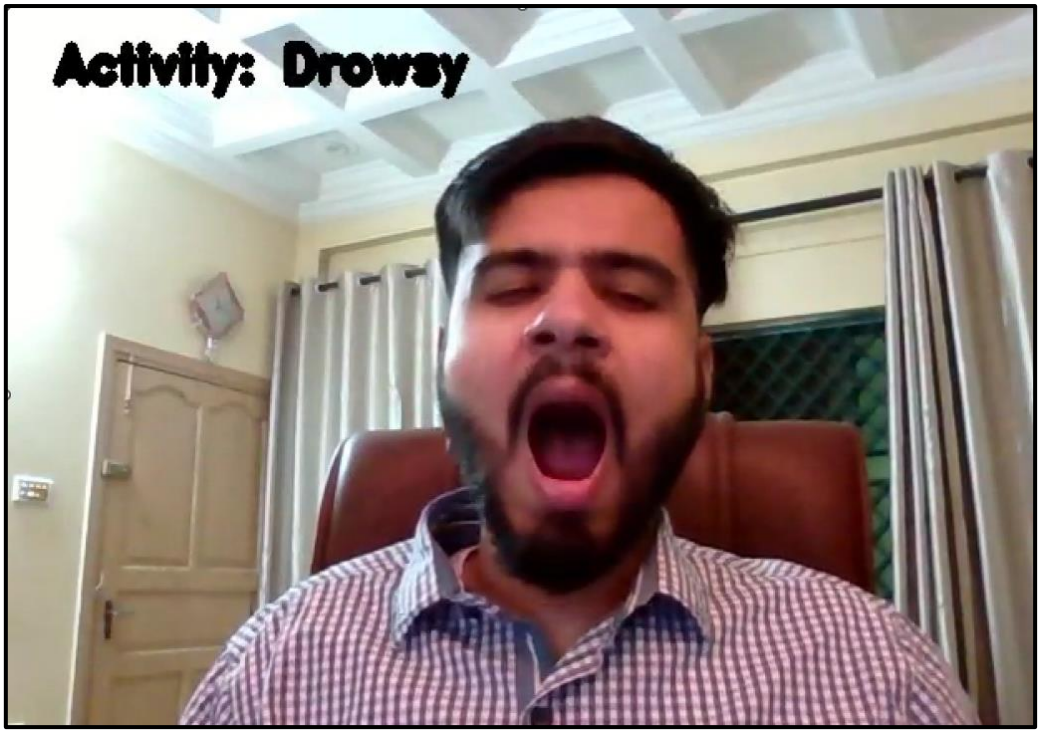


Fig 16.5.1.2: When a person is drowsy

16.6. Website Development

16.6.1. Frontend

Our website development started with designing and implemented the frontend – the look and feel of the website. To that end, the website was built as a multi-page interactive website. The website displays the product details, and has a surveillance page where an authorized admin can view the live video stream, select a model to apply, view results, and observe real-time log of incident detections.

The UI/UX of the website was designed to provide the best and easiest user experience for product users and admins.

Tools and Technologies: ReactJS

16.6.2. Backend

Website backend was developed using Flask and Python. Flask is a middleware framework that acts as a go-between the website frontend and the machine learning models. Using Flask, we set up a server that listens for requests – which will be sent by React once the user selects a relevant option on the website. Flask uses Python, and once it intercepts the request sent by the website, it will call the selected model on the image frame, and return the result to the website.

Tools and Technologies: Flask, Python

16.6.3.Database

The database used for the project is Firebase, a NoSQL database program that is hosted online on Google servers. We used its authentication module as well as the Cloud Firestore module to store data of incidents detected.

When any incident/abnormality is detected by the system, a record is saved in the database with date and time of detection, and type of incident. On the website, these logs are fetched and displayed to the admin.

Tools and Technologies: Firebase

16.6.4. Overall Integration

Overall integration of the complete website was done by building the frontend, backend, database and connecting them all together using Flask. Once complete, the website displays the real-time video feed, allows user to select a model, view model results and view real-time incident logs.

16.6.5. Issues Faced

1. Delay in returning predicted images to website
2. Image frames not being sent to website
3. React version changes led to dependency issues

16.6.6. Solutions

1. Hardware constraint, somewhat improved by using PC with GPUs
2. Resolved by encoding frame properly to specify data type in content header of http return message
3. Restrict React updates

17. Bonus Feature

As a bonus feature, we also worked on a mobile application to integrate with the overall system and provide easier access for users. To that end, we designed and built the UI of the mobile app.

Tools and Technologies: React Native

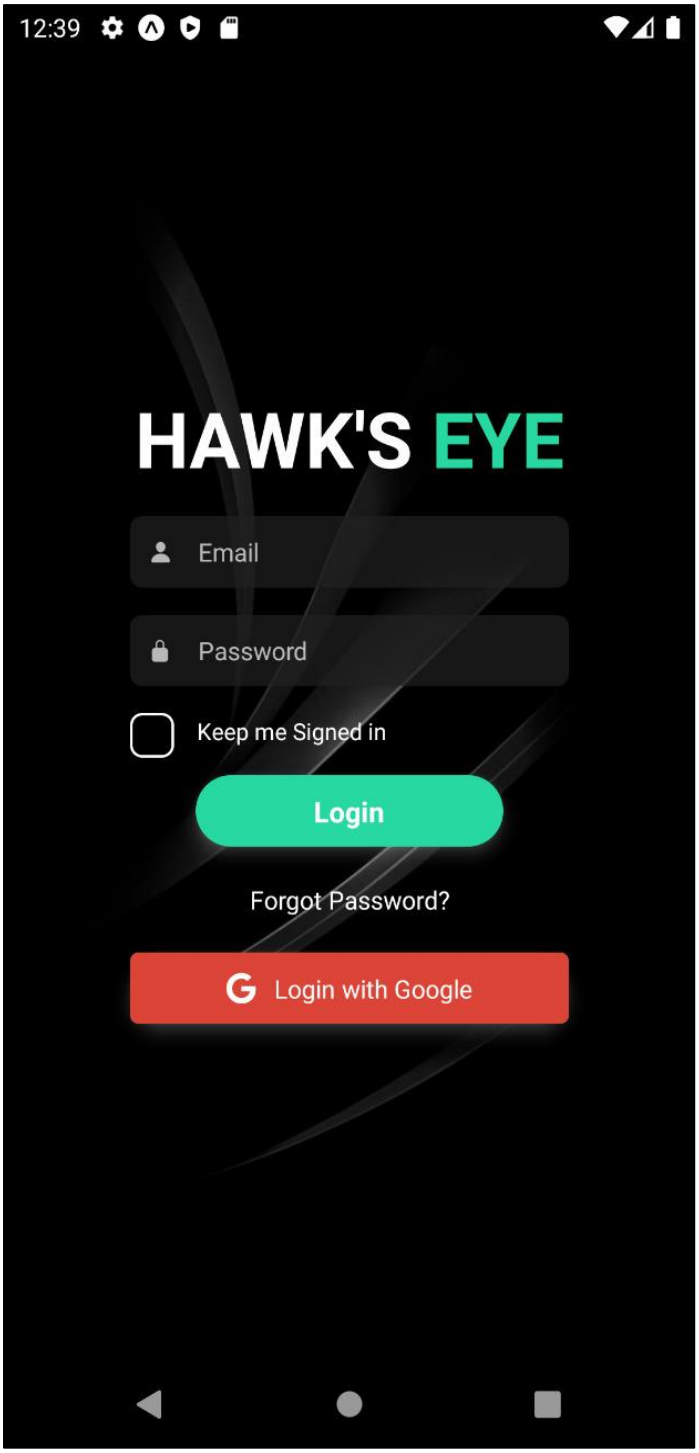


Fig 17.1: App landing page

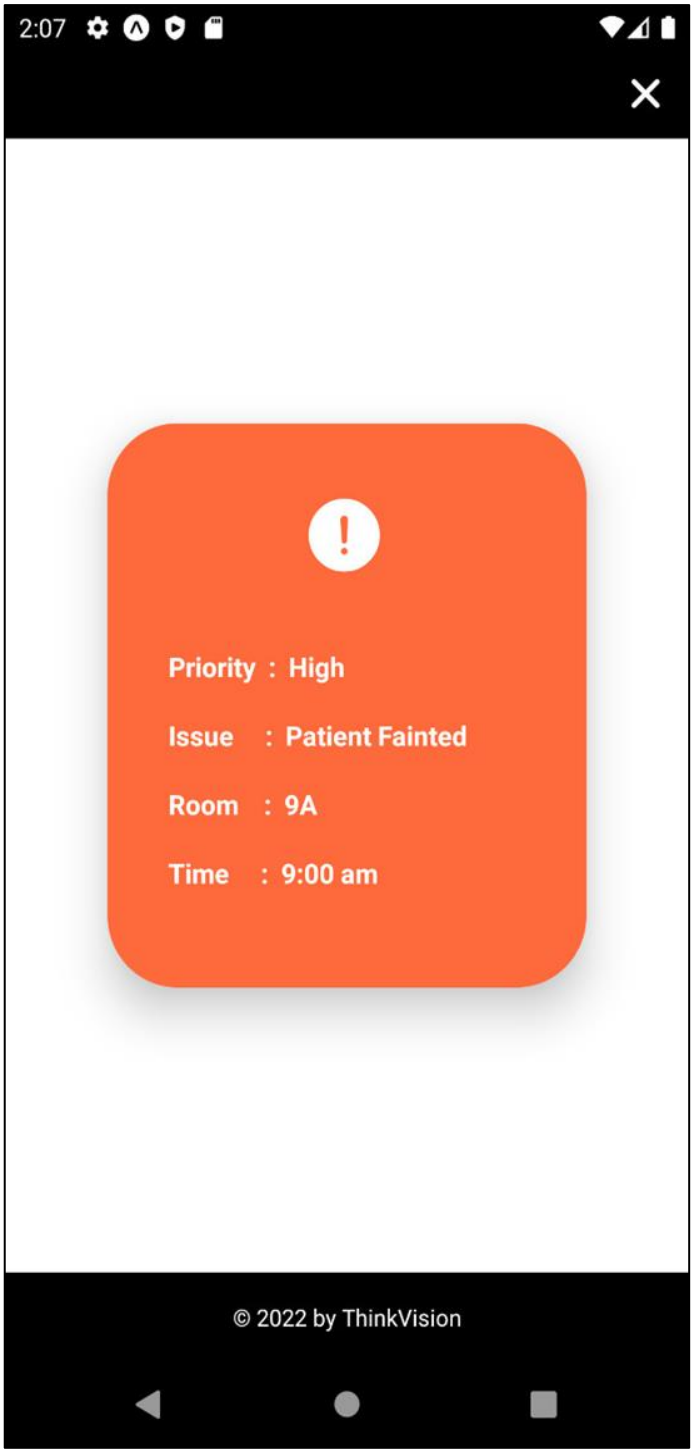
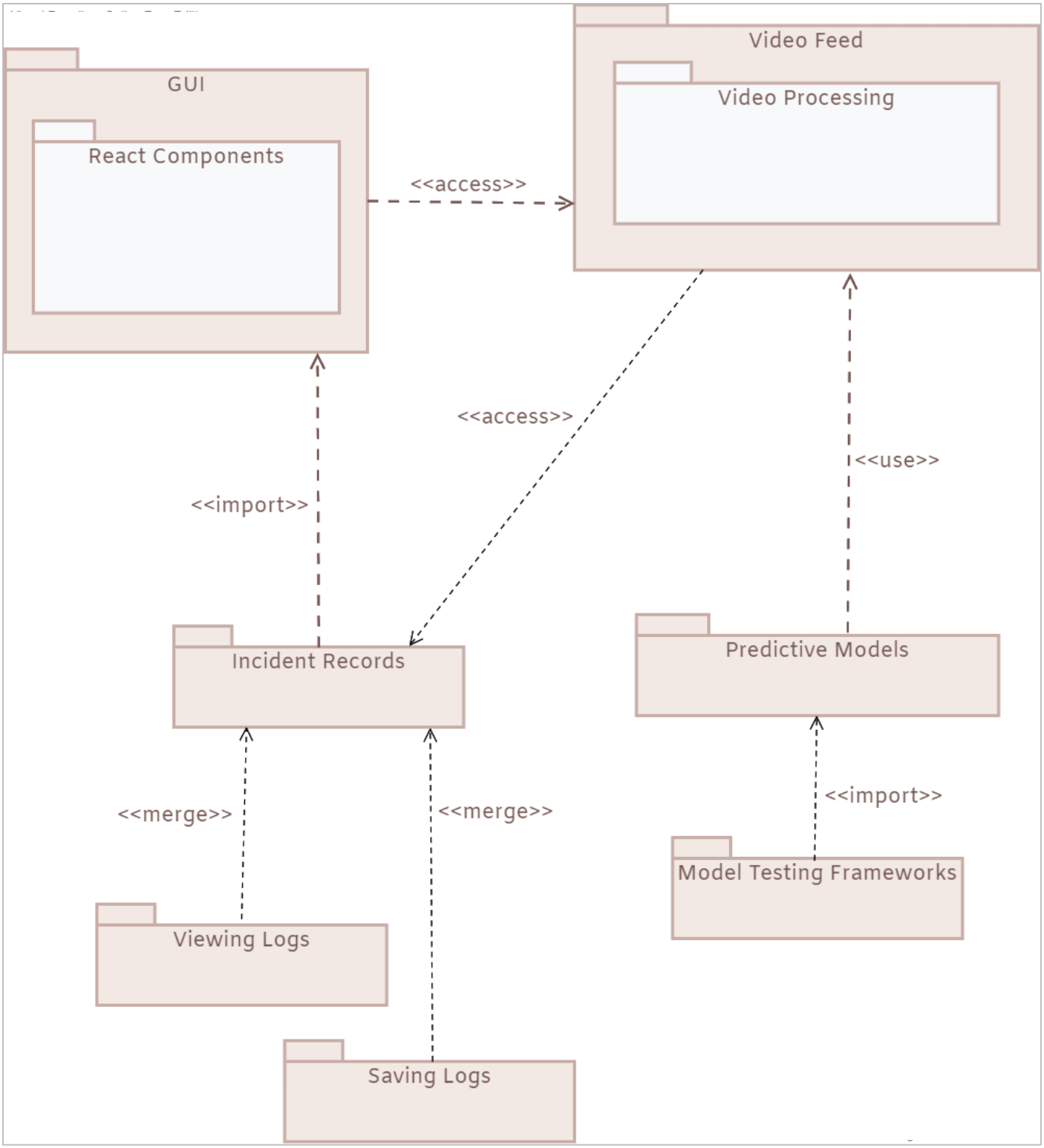
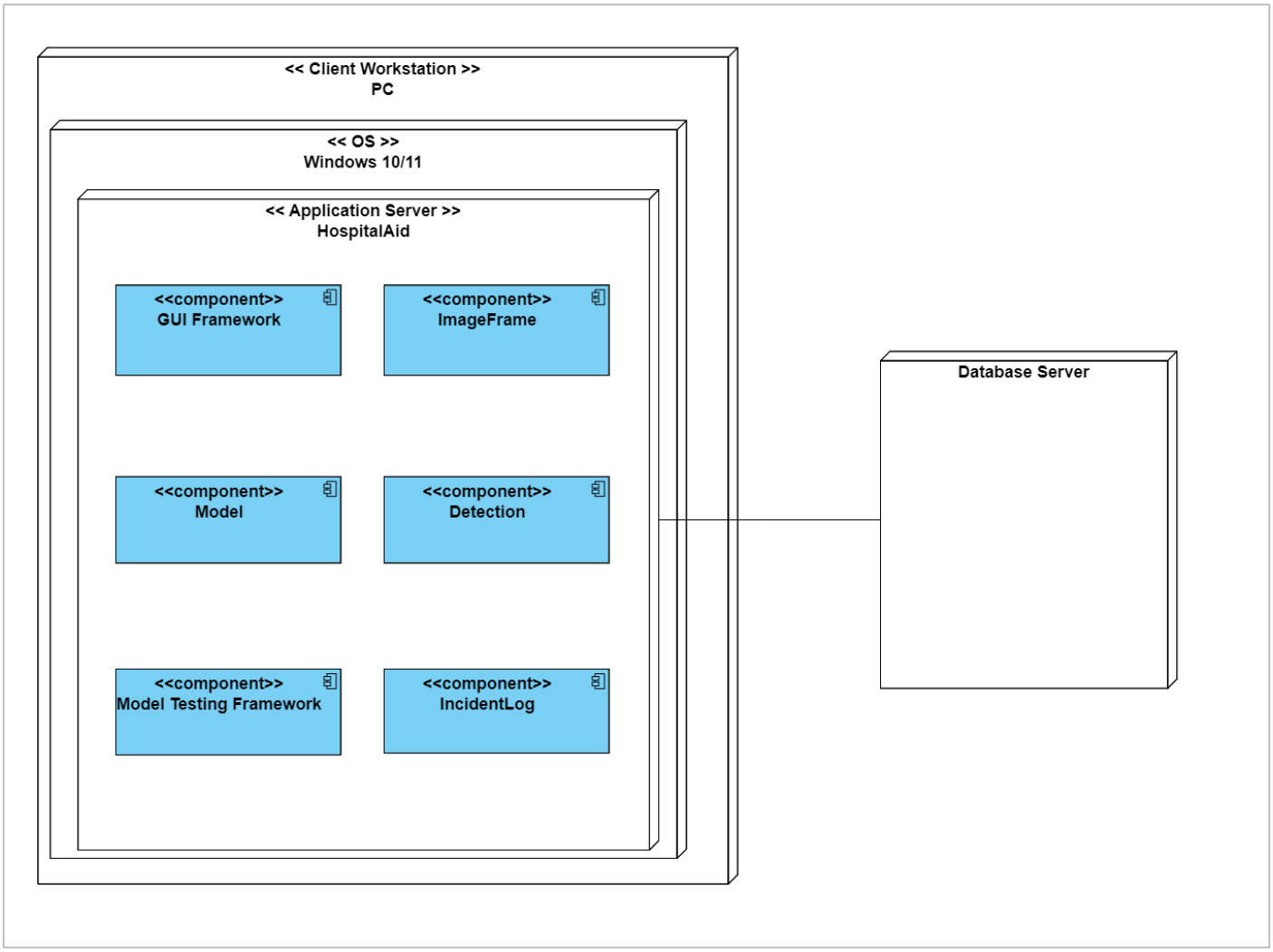


Fig 17.1: Alert UI

18. Package Diagram



19. Deployment Diagram



20. User Manual

20.1. Configuration/Setup

All code files are essentially divided into two folders – frontend and backend.

For starting the backend server:

- Open backend folder
- Open command prompt
- Run ‘flask run’
- Wait for the message on the terminal that informs server is up

For starting website:

- Open frontend folder
- Open command prompt
- Run ‘npm start’
- Browser will automatically open to the website

20.2. Website Walkthrough

Once the website is up and running, user can open browser to view landing page of browser. Details about product are shown on the landing page, and a login button on the top right (highlighted in the following figure) will take the user to the login page.

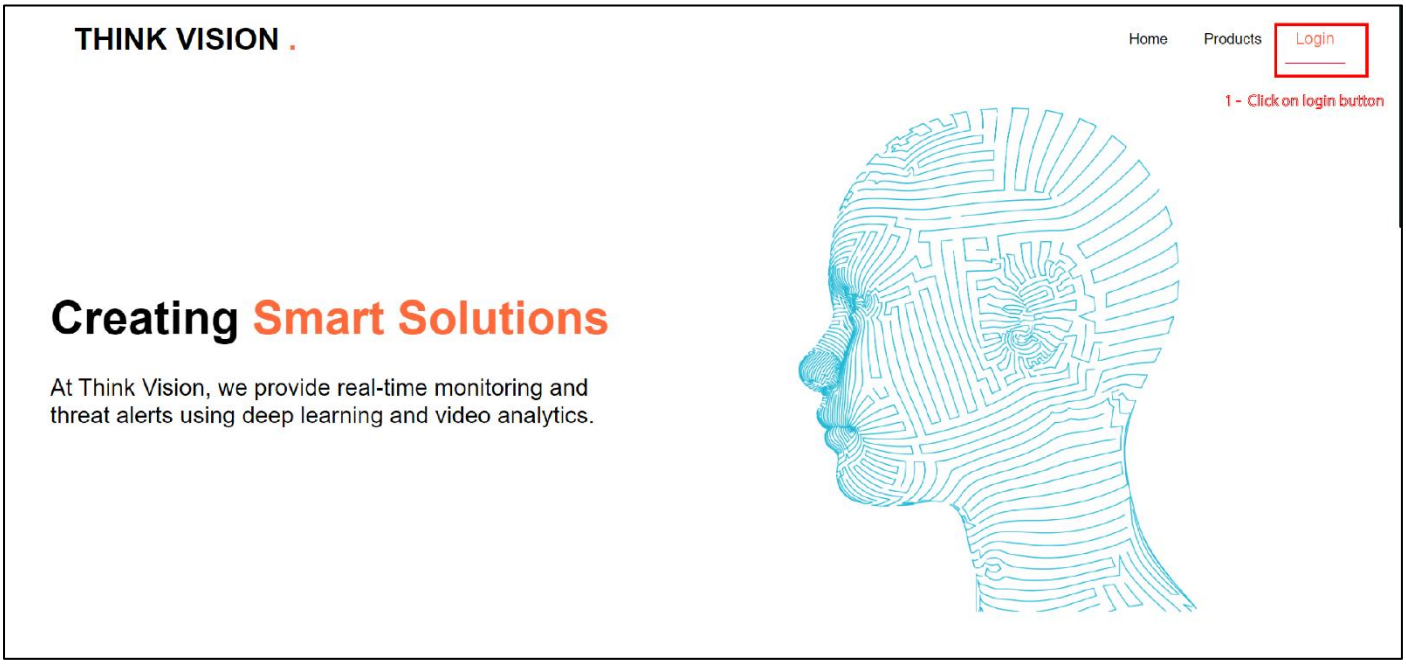


Fig 18.2: Landing page of website

At the login page, user has to enter their email and password, and press the login button. All authorized users will be able to progress forward then.

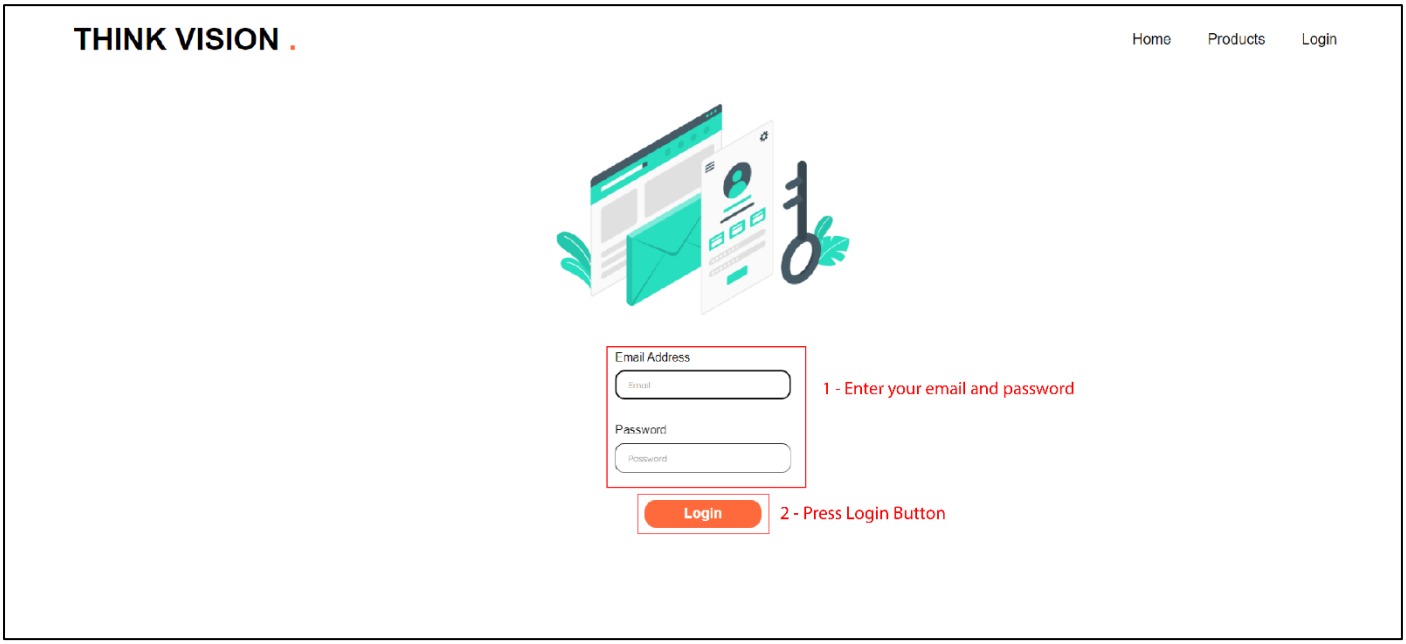


Fig 18.2: Login page

Once logged in, user can view the live video feed at the left. From the bottom bar, user can select any type of model to be applied to the video stream. And on the right is the dynamic, updated list of real-time logs generated of the abnormalities/incidents detected in the video.

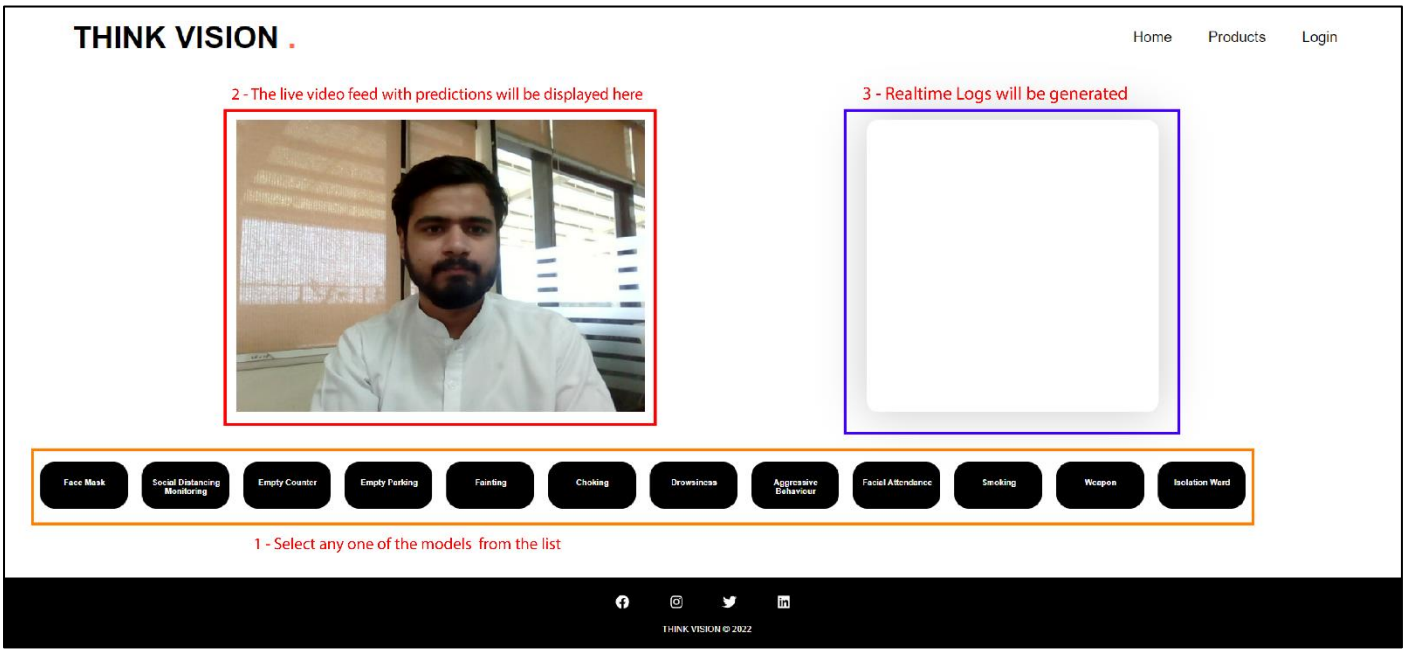


Fig 18.3: Surveillance page

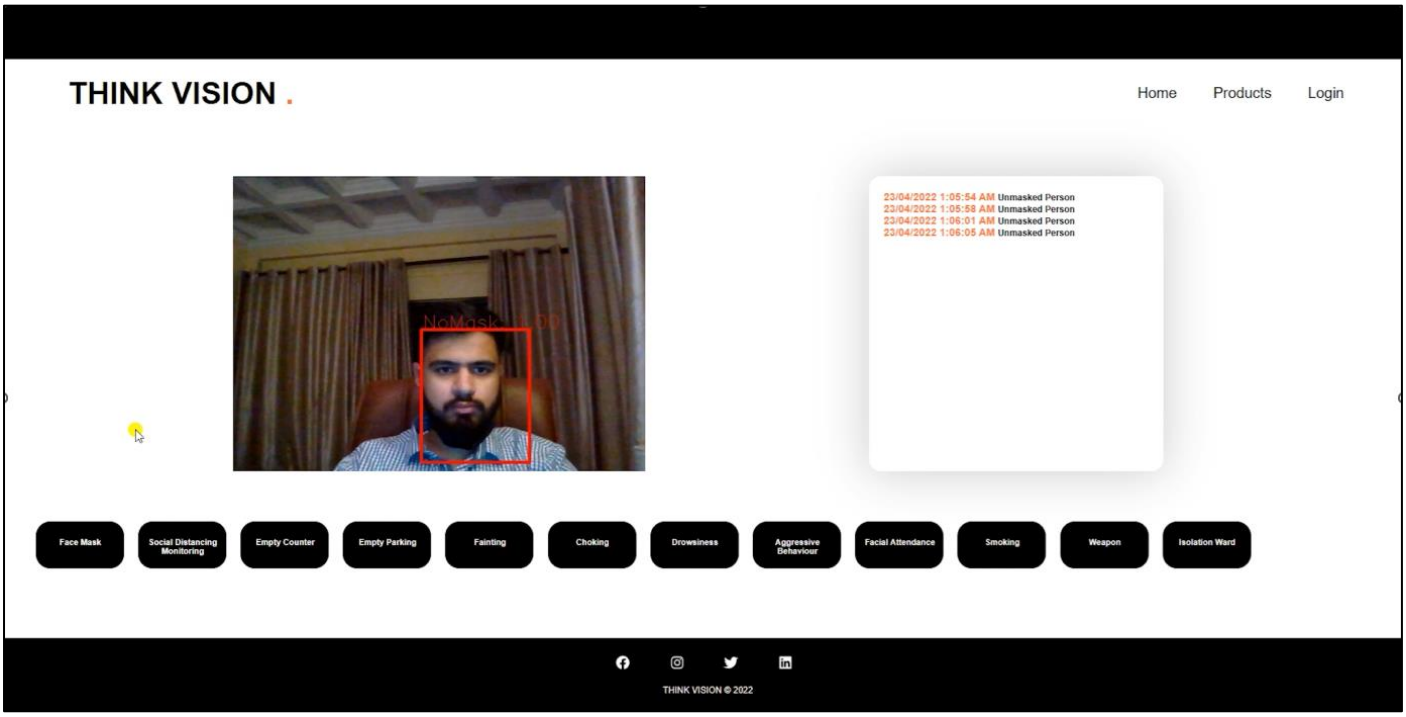


Fig 18.4: Surveillance page with detection and logs

Additionally, user can click on “Products” heading in navbar to navigate to products page, which shows all project modules.

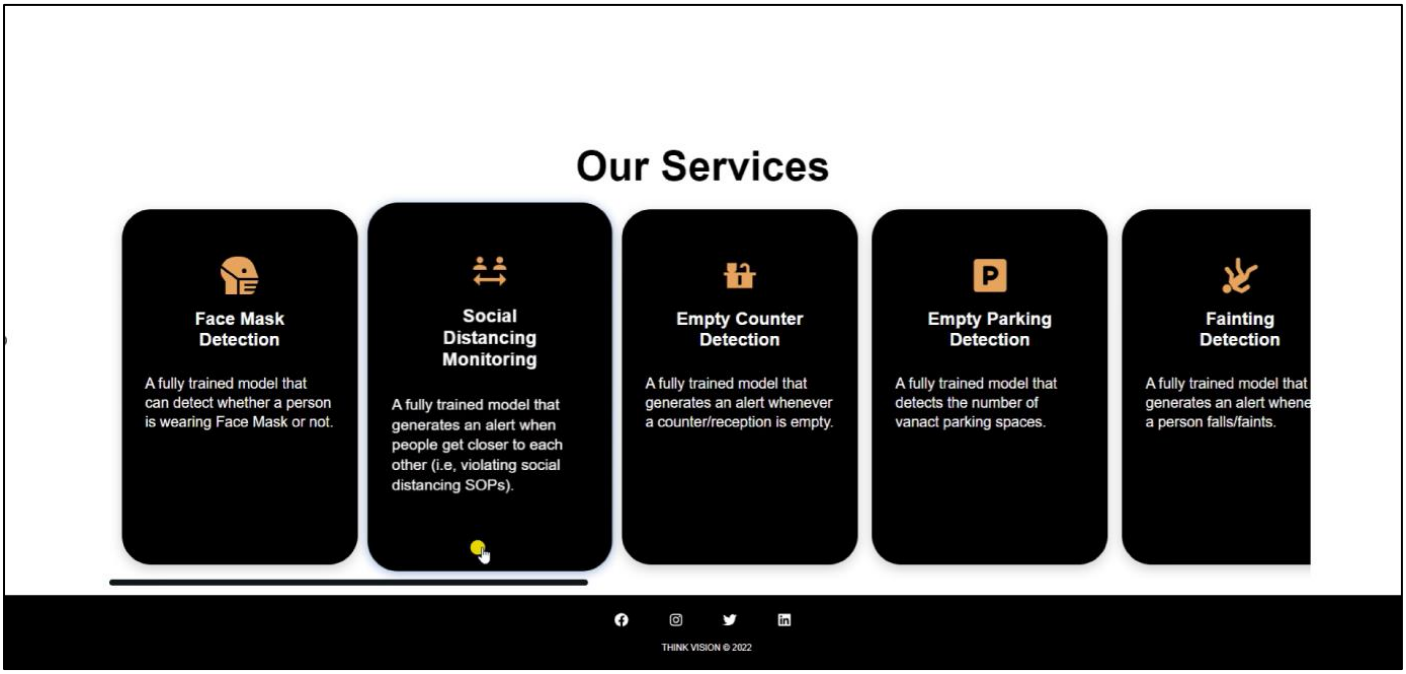


Fig 18.5: List of project modules/services

Clicking on any service card will lead user to a detailed overview of that feature, along with a demo video.

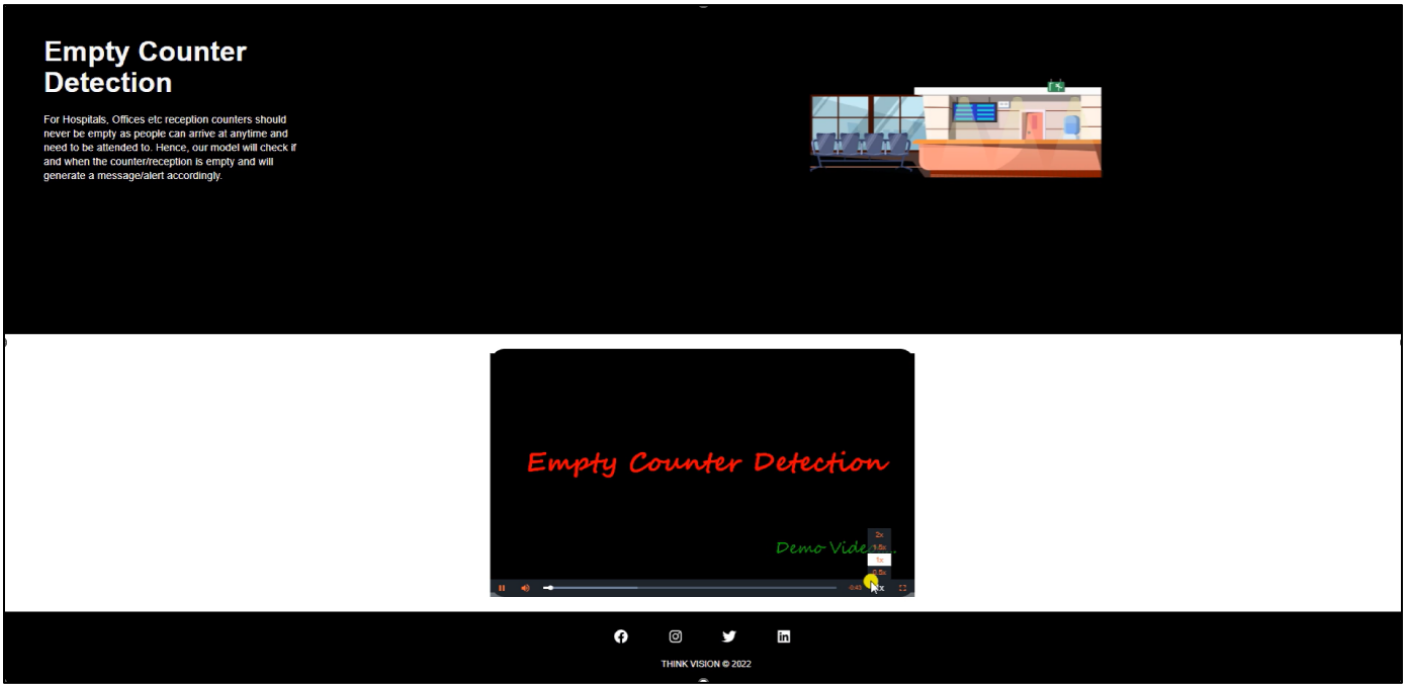


Fig 18.6: Empty Counter module details

21. References

AIZOOTech. (n.d.). AIZOOTech/facemaskdetection: detect faces and determine whether people are wearing mask. GitHub. Retrieved October 22, 2021, from https://github.com/AIZOOTech/FaceMaskDetection.

AlexeyAB. (n.d.). Alexeyab/Darknet: Yolov4 / scaled-yolov4 / yolo - neural networks for object detection (windows and linux version of darknet). GitHub. Retrieved October 22, 2021, from https://github.com/AlexeyAB/darknet.

Balajisrinivas. (n.d.). Balajisrinivas/face-mask-detection: Detecting face masks using python, keras, opencv on real video streams. GitHub. Retrieved October 22, 2021, from https://github.com/balajisrinivas/Face-Mask-Detection.

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020, April 23). YOLOv4: Optimal Speed and Accuracy of Object Detection. Retrieved from https://arxiv.org/abs/2004.10934.

Cabani, A., Hammoudi, K., Behnhabiles, H., & Melkemi, M. (2021). MaskedFace-Net – A dataset of correctly/incorrectly masked face images in the context of COVID-19. Smart Health, 19. https://doi.org/https://doi.org/10.1016/j.smhl.2020.100144

Cocodataset. (n.d.). Cocodataset/cocodataset.github.io. GitHub. Retrieved October 22, 2021, from https://github.com/cocodataset/cocodataset.github.io.

Fall detection. imvia. (2020, April 20). https://imvia.u-bourgogne.fr/en/database/fall-detection-dataset-2.html

Ghoddoosian, R., Marnim, G., & Athitsos, V. (2019, April 15). A Realistic Dataset and Baseline Temporal Model for Early Drowsiness Detection. Retrieved from https://arxiv.org/abs/1904.07312.

Gul, M. A., Yousaf, M. H., Nawaz, S., Ur Rehman, Z., & Kim, H. W. (2020). Patient monitoring by abnormal human activity recognition based on CNN Architecture. *Electronics*, *9*(12), 1993. https://doi.org/10.3390/electronics9121993

K, G. (2020, June 11). COVID-19: AI-Enabled Social Distancing Detector using OpenCV. towards data science. Retrieved from https://towardsdatascience.com/covid-19-ai-enabled-social-distancing-detector-using-opencv-ea2abd827d34.

Khan, W., Nawaz, F., & Hussain, A. (2020, November 12). Video Dataset for COVID-19 Social Distancing and Human Detection Validation. Retrieved from DOI: 10.17632/xh6m6gxhvj.1.

Loey, M. (2021, February 15). COVID-19 Medical Face Mask Detection Dataset. kaggle. https://www.kaggle.com/mloey1/medical-face-mask-detection-dataset.

Redmon, J., & Farhadi, A. (2018). (tech.). *YOLOv3: An Incremental Improvement*. Retrieved from https://arxiv.org/abs/1804.02767.

Singh, S., Ahuja, U., Kumar, M., Kumar, K., & Sachdeva, M. (2021). Face mask detection using yolov3 and faster R-CNN models: COVID-19 environment. *Multimedia Tools and Applications*, *80*(13), 19753–19768. https://doi.org/10.1007/s11042-021-10711-8

Yang, Y., Sarkis, R. A., El Atrache, R., Loddenkemper, T., & Meisel, C. (2021). Video-Based Detection of Generalized Tonic-Clonic Seizures Using Deep Learning. IEEE Journal of Biomedical and Health Informatics, 25(8), 2997–3008. https://doi.org/10.1109/JBHI.2021.3049649

Yang, D., Yurtsever, E., Renganathan, V., Redmill, K. A., & Özgüner, Ü. (2021, July 5). A Vision-based social distancing and critical density detection system for covid-19. MDPI. Retrieved October 21, 2021, from https://www.mdpi.com/1424-8220/21/13/4608.

Ultralytics. (n.d.). Ultralytics/yolov5: Yolov5 in PyTorch > ONNX > CoreML > TFLite. GitHub. Retrieved October 22, 2021, from https://github.com/ultralytics/yolov5.