#### Project Report on

**PICOBOO - SECURITY SYSTEM WITH MOTION TRACKING USING THERMAL IMAGING AND DEEP LEARNING**

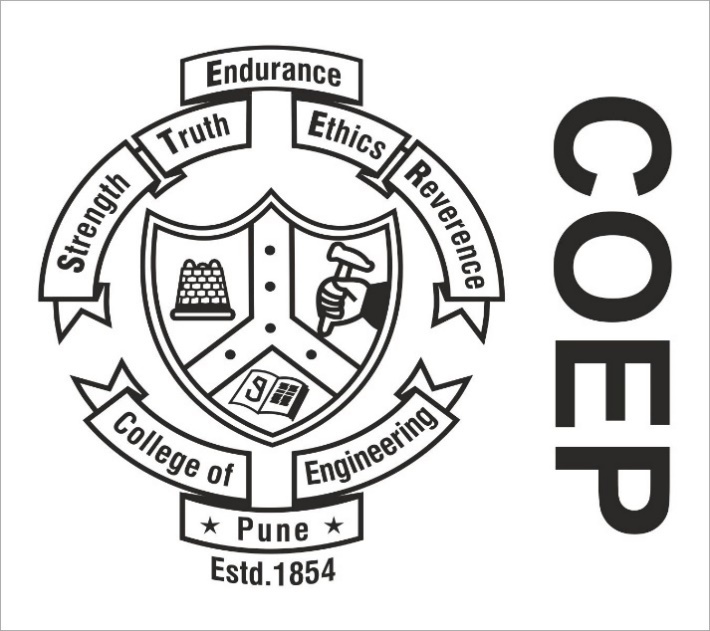
#### by

|  |  |
| --- | --- |
| **Anway Pimpalkar** | **111907066** |
| **Kushagra Shrivastava** | **111907074** |
| **Ishita Rathor** | **111807046** |

#### Under the guidance of

**Dr. Shrinivas Mahajan**

Head of Department, Electronics and Telecommunications, COEP



DEPARTMENT OF

**ELECTRONICS AND TELECOMMUNICATION ENGINEERING**

**COLLEGE OF ENGINEERING PUNE 2021 – 2022**

**Abstract**

This report explores the

**Contents**

1. **Introduction** …………………..………………………………………………………… 1
   1. Literature Review …………………….……………………….…………………….. 1
   2. Aim …………………………………………………………………………….……. 2
   3. Objectives …………………………………………………………….……………... 2
2. **Theoretical Background** ……………………………..…………………………………. 3
   1. Peripheral Components ……………………………………………………………….. 3
      1. MQ-2 Smoke and Flammable Gas Detector …………………………………... 3
      2. LM-35 Temperature Sensor ………………………………………………….... 5
      3. Shift Registers …………………………………………………………………. 5
      4. OV7675 Camera Module …………………………………………………….... 6
   2. Comparator Circuit …………………………………………………………………… 6
   3. Feature Extraction in Signal Processing ……………………………………………… 7
      1. Fast Fourier Transform (FFT) ……………………………………………….… 7
      2. Spectrogram Analysis …………………………………………………………. 8
      3. Mel-Scale and MFCCs ………………………………………………………… 9
   4. Embedded Machine Learning ……………………………………………………….. 10
3. **System Architecture** …………………………………………..………………………. 12
   1. System Overview ……………………………………………………………………. 12
   2. Hardware and Software Overview …………………………………………………... 13
   3. Contactless Operation using KWS and VWW CNN Models ……………………….. 15
      1. Datasets ………………………………………………………………………. 16
         1. ‘COCO (Common Objects in Context)’ Dataset ………………………….. 16
         2. Generic Methodology for the CNN Models ………………………………. 17
      2. Generic Methodology for the CNN Models ………………………………….. 17
         1. VWW Model for Person Detection ……………………………………….. 18
         2. KWS Model for Speech Recognition ……………………………………… 21
      3. Multitenant Architecture for Running KWS and VWW Simultaneously …….. 28
   4. Fire Safety Unit Outside The Elevator ………………………………………………. 29
      1. MQ-2 Smoke/Flammable Gas Sensor …………………………………...…… 29
      2. LM-35 Temperature Sensor ………………………………………………..… 29
      3. System Working ……………………………………………………………… 30
      4. Fire Alarm System and Connection to Main Elevator Interface ……………… 31
   5. Fire Safety Unit Inside The Elevator ………………………………………………… 32
4. **Results, Analysis and Conclusion** ……….…………………………………………..... 34
   1. Hardware Implementation ………………………………………………..…………. 34
      1. Working and Results Achieved …………………………………………….… 35
         1. Setting Up …………………………………………………………………. 35
         2. Running Inference ………………………………………………………… 35
         3. Accuracy and Latency Observations ………………………………...…… 39
      2. Bridging the Gaps Between The Hardware and The Simulations ………….… 39
   2. Software Based Simulations ………………………………………………………… 39
      1. Display and Fire Safety System of the Outside (Floor) Unit on Proteus ……... 40
         1. Working Overview ……...………………………………………………… 41
         2. Limitations of the Simulation …………………………………………...… 44
      2. Fire Safety System Inside The Elevator Unit on TinkerCAD …………….…. 44
   3. PCB Design ………………………………………………………………………….. 46
      1. Schematic …………………………………………………………………….. 46
      2. Bill of Materials ………………………………………………………………. 47
      3. PCB Layout …...……………………………………………………………… 47
      4. 3D Renders of PCB …………………………………………………………... 49
   4. Conclusion …………………………………………………………………………... 50

**References** ...……………...………………………………………………………………… 51

**Appendix** …………………………………………………………………………………… 52

[A] PCB Design Schematic on KiCAD EDA ………………………………………. 53

[B] PCB (pcbnew) Layout on KiCAD EDA ……………………………………….. 54

**List of Figures**

Figure 2.1: Outer build of MQ-2 Gas Sensor ................................................................................. 3

Figure 2.2: Sensor build with outer mesh removed …………………………………………….... 4

Figure 2.3: Tubular Sensing element inside MQ-2 ……………………….................................... 4

Figure 2.4 NEB Package of LM-35 Temperature sensor ……………………………….............. 5

Figure 2.5: Working of Shift Registers ………..………………………………………………… 6

Figure 2.6: OV7675 Camera Module …………………………………………..............................7

Figure 2.7: Skeleton circuit of Comparator using Operational Amplifier ...................................... 8

Figure 2.8: Fast Fourier Transform …………………………………............................................. 9

Figure 2.9: Sample spectrogram with a color legend ……............................................................ 10

Figure 2.10: Sample MFCC ……………………………………………………..……………… 11

Figure 3.1: Generic Block Diagram for Implementation ………………….................................. 13

Figure 3.2: Sample images from the COCO dataset with identified human subjects .................. 16

Figure 3.3: VWW block diagram .................................................................................................. 18

Figure 3.4: VWW model architecture as shown using Netron ..................................................... 20

Figure 3.5 KWS block diagram …………………………............................................................ 22

Figure 3.6: Analog visualization of raw audio signals of the keywords ...................................... 23

Figure 3.7: Generated FFTs of the keyword audio signals ........................................................... 24

Figure 3.8: Generated frequency domain spectrograms of the keyword audio signals ................ 25

Figure 3.9: Generated MFCCs of the keyword audio signals ....................................................... 25

Figure 3.10 Architecture of KWS Model ..................................................................................... 26

Figure 3.11 Final Training Results of KWS Model …………….................................................. 27

Figure 3.12 Circuit for MQ-2 Smoke Detector ............................................................................ 30

Figure 3.13 Circuit for LM-35 Temperature Sensor ..................................................................... 30

Figure 3.14:Circuit diagram for Fire Safety Alarm Outside the Elevator ..................................... 32

Figure 3.15:Circuit diagram for Fire Safety Alarm Inside the Elevator ....................................... 33

Figure 4.1 Individual components used for the hardware setup …………………………............ 34

Figure 4.2: Hardware setup for the Arduino Nano 33 BLE Sense and OV7675 Camera ............. 35

Figure 4.3: Sample greyscale QCIF images of size 144 x 176 pixels with human subject ……... 36

Figure 4.4: Output of the VWW model inferences ....................................................................... 36

Figure 4.5: LED Indicators of the VWW and KWS model inferences ......................................... 37

Figure 4.6: Output of the KWS model inferences ......................................................................... 38

Figure 4.7 Outside Unit Simulation on Proteus ............................................................................. 40

Figure 4.8 Displaying the hardware output on the 7 Segment Display of the simulation ………. 41

Figure 4.9 (a) Simulation when all parameters are optimal ......................................................... 42

Figure 4.9 (b) Simulation when smoke detector is set to 0 (smoke present) ................................ 42

Figure 4.9 (c) Simulation when temperature is above 45oC and smoke detector is set to 0 …..... 43

Figure 4.10: Simulation state when an anomaly is detected on other floors …………………..... 43

Figure 4.11: Smoke detector circuit using IC741 …………………….......................................... 44

Figure 4.12 (a) Simulation when smoke is detected inside the elevator below threshold ………. 45

Figure 4.12 (b) Simulation when smoke is detected inside the elevator above threshold ………. 45

Figure 4.13: PCB Schematic on KiCAD EDA ............................................................................. 46

Figure 4.14: Bill of Materials for the PCB .................................................................................... 47

Figure 4.15: Four Copper layers of designed PCB ……………………....................................... 48

Figure 4.16: 3D View of the created PCB ………………………................................................. 49

**List of Tables**

Table 3.1 Arduino Nano 33 BLE Sense Specifications ........................................................ 14

Table 3.2 List of Hardware and Software components used ................................................. 15

Table 3.3: Utterances of the chosen keywords in the KWS dataset ...................................... 17

Table 3.4 KWS model summary specifications ……………………………………………. 27

Table 3.5: Logic Table for Alarm System ............................................................................. 31

**Abbreviations**

CNN: Convolutional Neural Network

KWS: Keyword Spotting

VWW: Visual Wake Word

PCB: Printed Circuit Board

# Introduction

Elevator, also called lift, car that moves in a vertical shaft to carry passengers or freight between the levels of a multistory building. It is a platform that could either be open or closed and is used for lifting or lowering both people and goods to upper and lower floors.

Structurally, elevators have not changed much since the 1800s. However, their control systems have been altered for modern elevators in ways that improve on speed and safety. Modern elevator systems have added technology installed in them. Some have phones that allow the occupant to call for help in the event of an emergency. Others are fitted with a trap door located at the ceiling that make escape possible in emergency situations.

Even though elevators still maintain their original purpose of transporting people across floors, these upgrades that exist in the status quo have agreeably increased the quality and the general user experience of these elevators.

# Literature Review

Studies have shown that a significant part of the population believes elevators are unreliable and hold certain levels of uncertainty especially during emergencies. Additionally, elevators are not recommended during fire emergencies, which puts the people inside and their safety at a huge risk, the lack of automation is also a big risk factor, particularly during a time of a public health emergency as we are experiencing. To conclude, the current elevator technologies are not intuitive and cognizant of the risk that they may cause to the users and therefore, certain aspects need to be further developed.

The improvements and upgrades we propose are two-fold. One aspect is incorporating a smoke detector system inside and outside the elevator. This subsystem curbs the risk of people getting stuck inside an elevator to a large extent since it informs the users about smoke being in immediate

vicinity and gives them two things based on where the user is:

1. Ample time to get out of the elevator safely if the user is inside.
2. Get help for the people inside if the user is outside.

The second part is a person and speech detector in the elevator system. This subsystem allows a person to verbally communicate to the elevator instead of pressing buttons. This feature is exceptionally useful in the current paradigm, where social distancing and sanitization are quintessential for one’s safety.

Therefore, our system not only maximizes sanitization, but it also ensures user protection to its apex. Our projectis a simple but a significant change to the way things work in the status quo by the increasing the efficiency and safety of the current elevators in use.

# Aim

###### To ideate and model a safer and smarter elevator system using a device which can be deployed to pre-existing elevators.

# Objectives

## To ensure a contactless and thus hygienic elevator operation.

## To ensure safety from fire hazards by using smoke detectors and temperature sensors inside and outside the elevator

1. To understand, analyze and apply the working of different types of sensors
2. To learn effective and functional circuit designing.
3. To understand the crucial memory constraints of a given microcontroller and development environment and attain maximum benefits from the limited architecture.
4. To understand the difference of using pure electronic circuits to achieve a given output versus using a hybrid hardware/software development platform such as Arduino to achieve the same.
5. To understand the types of PCBs and develop our own PCB for the given project.
6. To understand how to apply the gained knowledge to create a holistic deployable product.

# Hardware and Software Design

To build any product end-to-end effectively, it is of utmost importance to understand the components used thoroughly. Before diving into the design specifics of our prototype, we will take an overview of the components we used to build it.

# Peripheral Components

# In this project, we have used available state-of-the-art equipment to build a prototype of the security system. They are detailed in this subsection.

# FLIR T420bx Thermal Imaging Camera

# A picture containing indoor, light Description automatically generatedThe FLIR T420bx Thermal Imaging Camera is a state-of-the-art thermal imaging camera designed and manufactured by Teledyne FLIR *(Wilsonville, Oregon, United States)*. It has a temperature range of -20°C to 350°C and is cased in a portable handheld package.

# *Figure 2.1: FLIR T420bx Thermal Imaging Camera (Courtesy: Teledyne FLIR)*

The camera is powered by Multi-Spectral Dynamic Imaging (MSX) technology which adds visible spectrum definition to IR images in real time for excellent thermal detail. It can provide a Frame Rate of upto 60Hz for video recordings, as we used during our data collection step. By default, it is equipped with a 25° lens. It supports uncompressed colorized video streaming over USB.



# *Figure 2.2: FLIR T 420 bx Thermal Imaging Camera Used for the Project.*

# Raspberry Pi Model 4 (2GB RAM Variant)

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It’s capable of doing everything you’d expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games. The Raspberry Pi 4 is a tiny single-board computer, which means that all of its components, from the memory to the USB ports, fit on one PCB without add-on cards or accessories. It measures 2.2 by 3.4 inches and stands about 0.6 inch tall.

# 

# *Figure 2.3: Raspberry Pi Model 4 used in our project.*

# The specifications of the Raspberry Pi Model 4 are as shown in Table 2.1.

*Table 2.1: Raspberry Pi Model 4 Specifications*

|  |  |
| --- | --- |
| **Feature** | **Specification** |
| Chip | BCM2711 |
| CLK | 1.5 GHz |
| RAM | 2 GB |
| Flash Memory | 16 GB (via microSD Card) |
| CPU | 5V / 3.3V |
| GPU | VideoCore VI |
| USB 2.0 | 2 |
| USB 3.0 | 2 |
| HDMI v2.0 | 2 |
| Ethernet | 1GBps |

# Software Components

# Google Firebase Realtime Database

The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in real-time. Firebase is a Backend-as-a-Service (Baas). It provides developers with a variety of tools and services to help them develop quality apps, grow their user base, and earn profit. It supports authentication using passwords, phone numbers, Google, Facebook, Twitter, and more. The Firebase Authentication (SDK) can be used to manually integrate one or more sign-in methods into an app. Data is synced across all clients in real-time and remains available even when an app goes offline. Firebase Hosting provides fast hosting for a web app; content is cached into content delivery networks worldwide. The application is tested on virtual and physical devices located in Google’s data centres. Notifications can be sent with firebase with no additional coding.

# Heroku

Heroku is a container-based cloud Platform as a Service (PaaS). Developers use Heroku to deploy, manage, and scale modern apps. Our platform is elegant, flexible, and easy to use, offering developers the simplest path to getting their apps to market. Heroku lets developers scale applications instantly. The simple way to scale applications makes working with Heroku easy and convenient. Projects created in Heroku are bound to repositories in GitHub. Heroku is known for running apps in dynos – which are just virtual computers that can be powered up or down based on how big your application is.

# Front-End Web

# The Front-End side is also called as the “Client-Side” of the application which includes everything the user sees and experiences. It includes text, navbars, colour-styles, images, buttons, etc. These help the user to understand and interact with the webpage.

# The Front end Development Program for Complete Beginners | Arcskill

# *Figure 2.4: Main Technologies used to Create the Front-End Web system.*

# Back-End Web

# The Back-End side is also called as the “Server Side” of application. This side of the code doesn’t come in contact of the user, but it manages and makes sure that everything on the client side works perfectly. Working with Backend involves storing/arranging data, writing APIs, creating libraries and working with the server requests.

# 

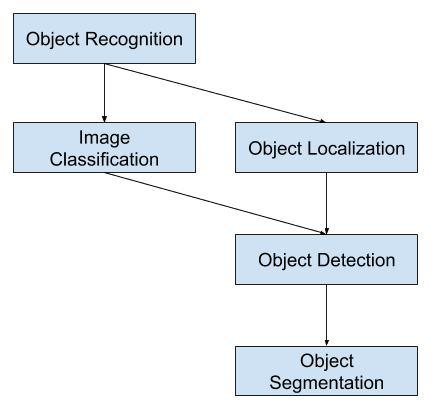
# *Figure 2.5: Most Popular Back-End Frameworks of January 2022.*

# Object Detection using Deep Learning on Embedded Systems

# Machine learning is the science of getting computers to act without being explicitly programmed. These processes require a lot of computing power and memory to be present in the system on which it runs. A CNN model once compiled is in the order of MBs if not GBs. A typical GPU consumes power in the order of Watts. Deploying such models to embedded microcontrollers is a challenge because of the severe power and memory constraints.

# The solution is to make use of various quantization methods on the models to shrink down the size and power requirements to be deployable to embedded microcontrollers. This is done by using various methods such as quantization, pruning, etc. Some accuracy is lost in the process due to the reduces computes. The key is to find the balance between the size, latency, and accuracy for any given application.

# TensorFlow provides multiple APIs to streamline this process of shrinking down models to make them deployable to embedded systems. For this reason, TensorFlow has been used extensively throughout this project for building and deploying neural networks to Raspberry Pi.



# *Figure 2.6: Basic Object Detection Algorithm.*

# Object detection is a computer vision technique that works to identify and locate objects within an image or video. Specifically, object detection draws bounding boxes around these detected objects, which allow us to locate where said objects are in (or how they move through) a given scene.

# To implement Object Detection, *EfficientNet* is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient. Unlike conventional practice that arbitrary scales these factors, the EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. The compound scaling method is justified by the intuition that if the input image is bigger, then the network needs more layers to increase the receptive field and more channels to capture more fine-grained patterns on the bigger image.

# We have used the quantized version of this architecture, *EfficientNet-Lite* for our model.

# System Architecture

This section elaborates the methodologies employed in our work for building a security system based on thermal imaging. A brief overview of this section is as follows. We begin by introducing the basic structure of the project we chose to implement. Then we briefly describe the approach to implement the object detection aspect of the system, and connecting it to a web interface accessible to anyone on the globe.

## System Overview

The main aim of the system is to be robust and serve as a working prototype for a system which can be deployed in genuine security scenarios. The entire prototype is based on a Raspberry Pi Model 4, which has an array of features in-built into the board, of which we can take maximum advantage. A generic block diagram of the entire proposed system is represented in Figure 3.1.

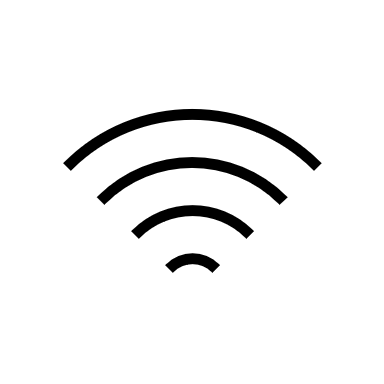
Thermal Camera

Stepper Motor

Machine Learning Inference

Driver Code

Alarm



Google Firebase Realtime Database

Communication Module

Heroku App



***Raspberry Pi***

User

*Figure 3.1: Generic Block Diagram for Implementation*

The project is a three-part system encompassing the following subsystems:

1. Human detection and tracking using a thermal imaging camera using TensorFlow Lite.
2. Web interface to show the status of the security system and stream the thermal camera stream to the application/interface.
3. Drive a stepper motor to follow the motion of the human detected in the frame based on the position of the human in the frame.

To implement the project, we selected components and controllers based on their availability, suitability, feasibility, cost, and ease of use. Most components were available within our college laboratories, which we used to avoid unnecessary expenses.

## Human Detection and Tracking

To implement this, we worked with TensorFlow. TensorFlow is a free and open-source software library for deep learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. We used the concept of object detection to complete the human detection and tracking subsystem. We can summarize our work related to human detection and tracking as the following steps:

1. Collecting data using the *FLIR T 420 bx Thermal Imaging Camera*.
2. Cleaning the data, labeling it, and building a dataset suitable to train a detection model.
3. Building a machine learning model capable of detecting humans in the frames.

## Dataset

At the foundation of every artificially intelligent system is the data on which it is trained. We chose to build a strong dataset which can later be diversified to be annotated into more than one type of objects. The data was collected using the *FLIR T 420 bx Thermal Imaging Camera*. The data was collected using two methods, either by taking pictures and videos on the camera itself and extracting the data from the SD card, or by connecting the camera to a computer and accessing the stream via the *FLIR Thermal Studio*.

Since the system we developed was for human intruder detection, we chose to take images and videos focusing on two main labels:

1. Human subjects.
2. Backgrounds

A person standing in front of a whiteboard

Description automatically generatedA picture containing text, colorful

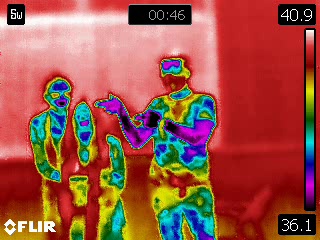
Description automatically generated

A person and person standing in a room with a whiteboard and chairs

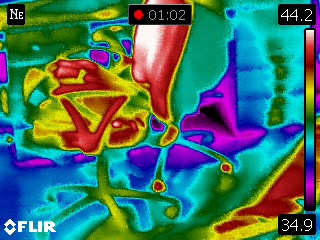
Description automatically generated with low confidenceA group of people in clothing

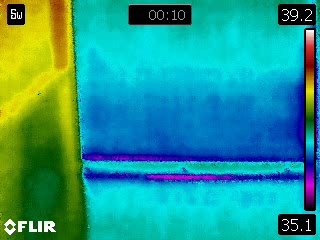
Description automatically generated with low confidence

*Figure 3.2: Sample images collected using the FLIR T 420 bx Thermal Imaging Camera.*

*  *

*Figure 3.3: Images with multiple human subjects in a single frame.*

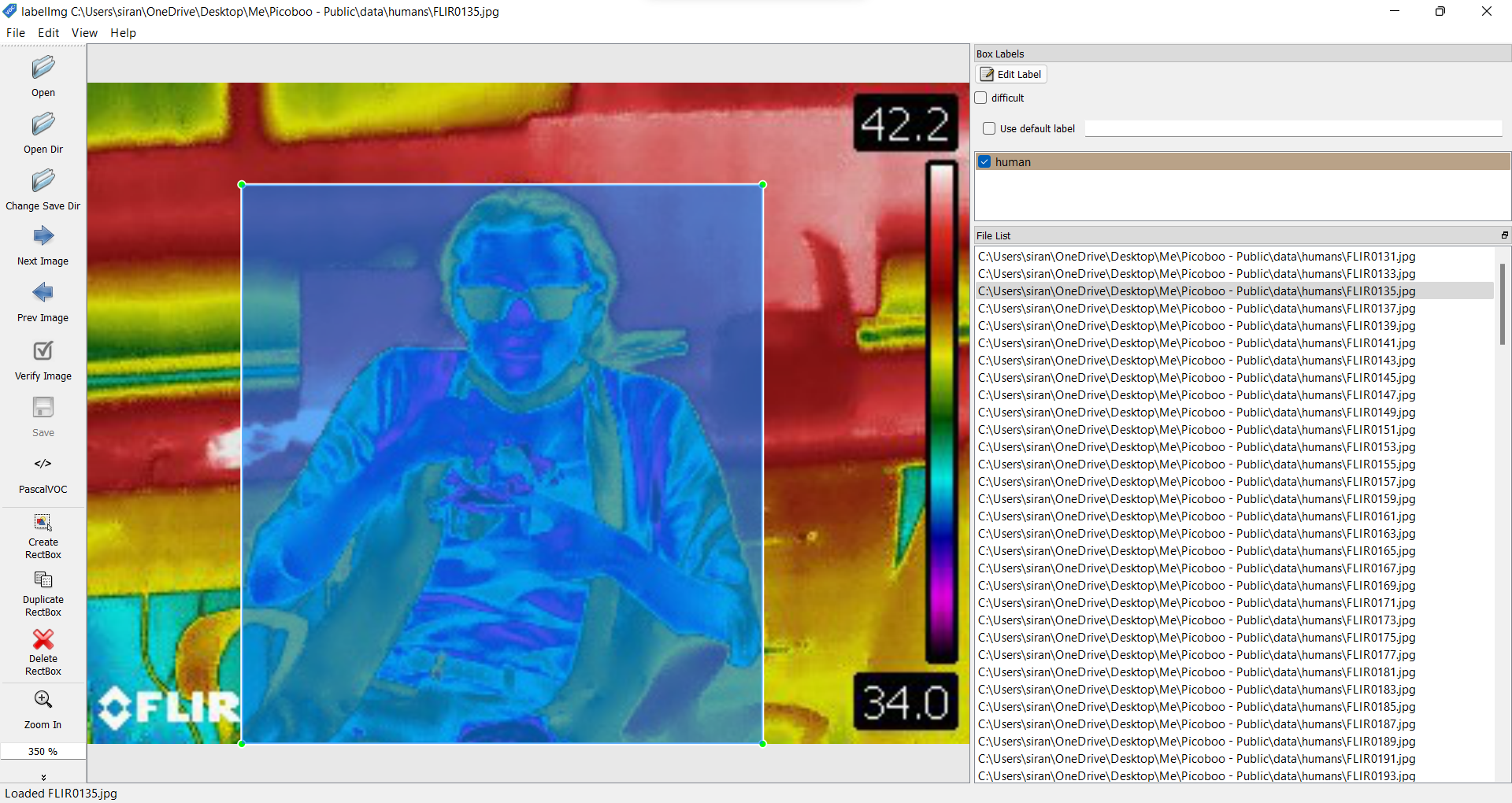
 

*Figure 3.4: Images with no humans (only background).*

A total of 90 photos and videos of human subjects, and 2 videos of background views were recorded and obtained. The videos were then split frame-by-frame using a Python script to create images which can be used to train the model. Post this split and some preliminary data cleaning, we had a total of 6,067 images with human subjects and 4,937 instances of backgrounds.

Considering the time it would have taken to train our model, we chose a small portion of this data to build the model, of 184 instances of humans, and 222 instances of backgrounds.

To label this data, we used a free open-source data labeler, *LabelImg*. The software allows for easy rectangular bounding of subjects in images, with annotations saved in XML files. This software had a limitation, it cannot add empty annotations to images with no human subjects. To fix this, we used a Python script to generate the annotations automatically for the background images from the dataset.



Text

Description automatically generated

*Figure 3.5: Annotations creating using LabelImg for the dataset.*

In the data, we left the additional information captured as is, such as the temperature scale, the FLIR watermark and the current temperature reading. We hoped to create a model that would learn to ignore these features as background features. A summary of the created dataset is given in Table 3.1.

*Table 3.1: Dataset specifications.*

|  |  |
| --- | --- |
| Human Instances Captured | 6,067 |
| Background Instances Captured | 4,937 |
| Human Instances Used for Training | 184 (3.03%) |
| Background Instances Used for Training | 222 (4.4%) |
| Total Size | 39.6 MB |

### Object Detection Model using EfficientNet-Lite

The Object Detection model must have two key features – high accuracy with low latency. With extensive use of TensorFlow, we were able to build multiple models which took excellent advantage of the TensorFlow Lite library and performed well under limited power and memory environments such as the Raspberry Pi.

Chart, scatter chart

Description automatically generatedIn May 2019, Google released a family of image classification models called *EfficientNet*, which achieved state-of-the-art accuracy with an order of magnitude of fewer computations and parameters. They looked to optimize these models for deployment at the edge, and in 2020 they launched *EfficientNet-Lite* which runs on TensorFlow Lite and is designed for performance on mobile CPU, GPU, and EdgeTPU. EfficientNet-Lite brings the power of EfficientNet to edge devices and comes in five variants, allowing users to choose from the low latency/model size option *(EfficientNet-Lite0)* to the high accuracy option *(EfficientNet-Lite4)*. Below are how the quantized *EfficientNet-Lite* models perform compared to similarly quantized version of some popular image classification models.

*Figure 3.6: Comparison of EfficientNet-Lite models to different Image Classification models. (Courtesy: Google TensorFlow)*

EfficientNet has a reputation for achieving high accuracy with minimal parameters and FLOPS (Floating Point Operations Per Second). It is suitable for use with the Raspberry Pi, which has limited processing power. We implemented transfer learning using the learned weights of EfficientNet from the ImageNet dataset since both FER-2013, and the ImageNet are image classification datasets. The architecture of the EfficientNet-Lite0 network consists of 1x1 Convolution, Average Pooling, Convolution, Dense Connections, Dropout, Inverted Residual Block, Batch Normalization, and ReLU6.

To train our models, we used this *EfficientNet-Lite* series and checked their performance for our use-case. We started with the *EfficientNet-Lite0* baseline model using the *TFLite Model Maker*. The TFLite Model Maker library simplifies the process of adapting and converting a TensorFlow neural-network model to particular input data when deploying this model for on-device ML applications. After training the model on the *TFLite Model Maker*, it had to be quantized post-training to shrink it down. This created a flatbuffer for the model in TensorFlow Lite. Post-training quantization is a conversion technique that can reduce model size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy.

A picture containing text, indoor

Description automatically generated

*Figure 3.6: Quantization Techniques. (Courtesy: Google TensorFlow)*

*Table 3.2: Comparison of Model Hyperparameters.*

|  |  |  |
| --- | --- | --- |
| **Hyperparameter** | **EfficientNet-Lite0** | **EfficientNet-Lite2** |
| Learning Rate | 0.0090 to 2.2931e-05 | 0.0140 to 4.6005e-05 |
| Batch Size | 8 | 16 |
| Epochs | 20 | 20 |
| Size of Model | 4,341 KB | 7,212 KB |
| Training Time (No Hardware Acceleration) | 3,977 secs | 11,123 secs |

The accuracy of Object Detection Models is measured using two main metrics:

1. Average Precision
2. Average Recall

Precision measures how accurate is your predictions. i.e. the percentage of your predictions are correct. Recall measures how good you find all the positives.

Average Precision is defined as the area under the precision-recall curve (PR curve). IoU is a good way of measuring the amount of overlap between two bounding boxes or segmentation masks. If the prediction is perfect, IoU = 1, and if it completely misses, IoU = 0. A degree of overlap will produce a IoU value between those two. This IoU can be kept fixed, for example, at 50% or 75%, which are called AP50 and AP75, respectively. When this is the case, it is simply the AP value with the IoU threshold at that value. The idea is similar for Average Recall as well.

Ideally, we must choose a model with maximum mAP and mAR with minimum latency. This is discussed at length in the Results section of the report.

Based on the detections returned by the inference model, two actions are triggered:

1. Rotation of the stepper motor, turning in the direction of the detections made in the image.
2. Update the detections record in the *Google Firebase Realtime Database*.

### Updating Detections Count to Google Firebase Realtime Database

Google Firebase has provided extensive APIs through which one can connect to the database through a multitude of languages and setups, including Python. The Realtime Database is excellent for backend server services which require data updating and recalling in real time applications such as ours.

### Human Detection Pseudocode

### The algorithm which was followed to run inference on the video stream is as follows:

### continuously run:

### start capturing video input from the camera

### set visualization parameters

### initialize the object detection model

### continuously capture images from the camera:

### run inference

### if a human is detected in the image:

### update firebase count

### draw key points and edges on input image

### calculate and show the FPS

*.*

## Web App

The web application we built has the basic functionality required to run the project, as described below.

###### Front-End Using Vanilla HTML, CSS, and JS

###### In our application, we used HTML5 (Hypertext Markup Language) to make the structure of our site and to create a link between web pages. To style our website we used CSS (Cascading Style Sheets) which focuses on the presentation of the document and deals with text-font/style, background-colour, spacing and similar styling objects.

###### With the help of HTML and CSS we were able to make a static page, but to make the page Dynamic, we used JavaScript. JS is used to execute complex actions and enables user interaction with the web page. In our case, JS was used to change the state of the page (by changing the HTML content and CSS styling) and inform the user whether they are safe or not.

###### Back-End Using Node.JS

# For our application we used Node.JS to set up our Local Host server, manage the Back-End Data and work with Real-Time Firebase data. Node.JS is a free JavaScript Open-Source cross-platform for server-side programming that allows users to build network applications quickly. The definition of Node.js as mentioned in its [official documentation](https://nodejs.org/) – “Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

# Text Description automatically generated

# *Figure 3.7: Snapshot of NPM dependencies used.*

# Along with Node.JS, we used it framework – Express.JS which is a lightweight open-source web application which helps to organize web applications on the server-side into a more organized MVC architecture.

# Express JS is one of the most used Backend Framework and the reason behind it is its pre-defined libraries which reduces the complex programming to build efficient APIs. It has made programming in Node.JS effortless.

# A screenshot of a computer Description automatically generated with medium confidence

# *Figure 3.8: A snapshot of Console Window showing Detections.*

# As all of this was done Locally on LocalHost:3000, we wanted to shift it to a server so that anyone can access it. So, we used the services provided by Heroku and Git to shift our server from Local Host to Heroku’s Server.

# Results, Analysis and Conclusion

The implementation of this project was carried out in two stages based on the availability of parts and components. The VWW and KWS models for the contactless elevator system were implemented using hardware on an Arduino Nano 33 BLE Sense module. The rest of the subsystems were implemented on simulation softwares including Proteus Design Suite and TinkerCAD based on the necessities and additional features which are required.

## Human Detection and Tracking Analysis

As discussed, we used the *EfficientNet-Lite* family for classification and tracking of humans using a thermal imaging camera. The *EfficientNet-Lite0 model* gave a *mean Average Precision (mAP)* of 67.7% on the validation dataset. Thanks to the TensorFlow Model Optimization Toolkit, we easily quantized the model via integer-only post-training quantization. This reduced the model size by 4x and improved inference speed by 2x. The mAP post-quantization was 69.09%, surprisingly higher than our pre-quantization mAP.

When the model ran on a given video stream, it ran at approximately 6 FPS. This was acceptable given our memory constraints and latency expectations. In an attempt to increase the accuracy, we also trained the model using *EfficientNet-Lite2.* This took considerably longer to train, but gave a higher mAP of 75.3%, which reduced to 72.6% post-quantization. Upon running inferences on our video stream, the frame rate dropped to approximately 2 FPS, which was impractical for our application.

The sizes of the models were considerably small. The *EfficientNet-Lite0* was a total of 4.2 MB, and the *EfficientNet-Lite2* was a total of 7.0 MB on the disk. This was an expected model size, considering the extensive architecture of the networks as observed on Netron. The complete data for the training of the models is given in Table 3.2.

*Table 4.1: Comparison of Human Tracking Model Accuracies.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Parameter** | **EfficientNet-Lite0** | **EfficientNet-Lite2** |
| Outputs | Size of Model | 4,341 KB | 7,212 KB |
| Training Time (no hardware acceleration) | 3,977 secs | 11,123 secs |
| Pre Quantization Metrics | mean Average Precision (mAP) | 0.6770454 | 0.75306594 |
| Average Precision with 50% IoU | 0.9646272 | 0.9970297 |
| Average Precision with 75% IoU | 0.8850385 | 0.91438556 |
| Average Precision with Medium Objects | 0.08974359 | 0.2025 |
| Average Precision with Large Objects | 0.70565706 | 0.7883744 |
| Average Recall with Medium Objects | 0.5 | 0.3 |
| Average Recall with Large Objects | 0.75555557 | 0.84444445 |
| Average Recall with Max 1 Object | 0.5 | 0.5736842 |
| Average Recall with Max 10 Objects | 0.74210525 | 0.81578946 |
| Post Quantization Metrics | mean Average Precision (mAP) | 0.69065887 | 0.7267763 |
| Average Precision with 50% IoU | 0.9431037 | 0.9970297 |
| Average Precision with 75% IoU | 0.8883388 | 0.93584985 |
| Average Precision with Medium Objects | 0.044444446 | 0.22777778 |
| Average Precision with Large Objects | 0.7259452 | 0.7535018 |
| Average Recall with Medium Objects | 0.4 | 0.7 |
| Average Recall with Large Objects | 0.76111114 | 0.7888889 |
| Average Recall with Max 1 Object | 0.52105266 | 0.5473684 |
| Average Recall with Max 10 Objects | 0.74210525 | 0.7842105 |

A picture containing graphical user interface

Description automatically generated Graphical user interface

Description automatically generated

*Figure 4.1: Image Outputs of EfficientNet-Lite0 (left) and EfficientNet-Lite2 (right).*

After using both the models, we decided to proceed with the *EfficientNet-Lite0* model. Using OpenCV to drive the video stream into the inference system, the model was deployed to the Raspberry Pi. The output images for both the models are shown in Figure 3.8. As seen in the top-left corner of the images, the latency of the *EfficientNet-Lite0* model is much lesser than the *EfficientNet-Lite2* model.

Using the API provided for Python, we send the number of humans detected in the frame to the Firebase Realtime Database, which was immediately reflected in the database as shown in Figure 3.8.

Graphical user interface, text, application, email

Description automatically generated

*Figure 4.2: Updating detections count in Google Firebase Realtime Database*

## Access to Security Status Through Web App

If the number of detections on the Firebase Realtime Database is more than zero, then the page indicates that the security system is not safe and sends an alert of the intruder. This is triggered immediately due to the event handler which is set off as soon as there is a change in the Database reference point. Figures 4.3 and 4.4 show the changes in the page.

Graphical user interface, text, application

Description automatically generated

*Figure 4.3: Webpage when there is an intruder present in the frame.*

Graphical user interface, application, PowerPoint

Description automatically generated

*Figure 4.4: Webpage when there is no intruder present in the frame.*

# Conclusion

# Smart Elevator Systems Using Embedded Machine Learning and Fire Safety Mechanisms contributes to the domain of conventional elevators to improve current safety systems on various levels. The standard ways of implementing safety mechanisms include trap doors, phone systems inside the elevator for emergencies. Our project upgrades these systems in accordance to current needs. We have implemented machine learning to cater to the tasks based on keyword spotting and computer vision. Taking advantage of the TinyML system we used, we could do more with less cost incurred. We built a VWW model for the purpose of person detection. To create a speech recognition model, we had to extract certain features from the microphone signals, for which we employed a couple of ways, namely FFTs, Spectrograms and MFCCs. Once the user speaks, the raw data is captured and its FFT is generated, The Discrete ­ Fourier Transform of the data sampled over in a small unit of time makes up a small slice of the spectrogram therefore after taking multiple such Fourier Transforms of different timestamps of the incoming signal, we are able to form a single spectrogram which represents the variation of volume and frequency over time much better. We then generate a MFCC of the audio signal, and feed this to our neural network to get a much higher accuracy because of the extracted features than we would have using just the raw audio data. Thus, our model with its increased efficiency enables more automation for the comfort and safety of the users, allowing them to experience a touch-free elevator ride. Studies have shown that elevators elicit a form of fear in the population despite being normalized to a huge extent. This fear is caused because of the possible safety issues that they may compromise on in the status quo. In our model, we aimed to change that. We have also inculcated a smoke detector within the elevator which detects smoke and alerts the people inside the elevator, giving them buffer time to move out of the elevator and our fire safety system outside the elevator, which detects smoke and safety hazards outside the elevators and alerts the people. We therefore provide a holistic solution to everyday elevator-related problems faced.

# References

|  |  |
| --- | --- |
| [1] | Prof. Omkar M. Shete, Ms. Amrita A. Vitekar, Ms. Ashwini N. Patil, “Design & Control of an Elevator Control System using PLC,” *International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering,* vol. 5, 2017. |
| [2] | Max T. Kinateder, Hidemi Omori, Erica D. Kuligowski “The Use of Elevators for Evacuation in Fire Emergencies in International Buildings”, July 2014 |
| [3] | Arduino Nano 33 BLE Sense Schematics: https://content.arduino.cc/assets/NANO33BLE\_V2.0\_sch.pdf |
| [4] | Arduino Nano 33 BLE Sense Datasheet:  https://content.arduino.cc/assets/Nano\_BLE\_MCU-nRF52840\_PS\_v1.1.pdf |
| [5] | OV7675 Camera Module Datasheet:  https://www.uctronics.com/download/Image\_Sensor/OV7675\_DS.pdf |
| [6] | MQ-2 Gas Sensor Datasheet:  https://cdn.sparkfun.com/assets/3/b/0/6/d/MQ-2.pdf |
| [7] | LM-35 Temperature Sensor Datasheet:  https://www.ti.com/lit/ds/symlink/lm35.pdf |
| [8] | Common Anode Seven Segment Display:  https://docs.broadcom.com/docs/AV02-2553EN |
| [9] | 74HC595 Shift Register Datasheet:  https://www.ti.com/lit/ds/symlink/sn74hc595.pdf |
| [10] | 74AHC1G86 EX-OR Gate IC Datasheet:  https://www.diodes.com/assets/Datasheets/74AHC1G86.pdf |
| [11] | IC741 Operational Amplifier Datasheet:  https://www.ti.com/lit/ds/symlink/lm741.pdf |
| [12] | IC741 Operational Amplifier Datasheet:  https://www.ti.com/lit/ds/symlink/lm741.pdf |
| [13] | TensorFlow and Embedded Machine Learning:  https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning |
| [14] | Daniel Situnayake, Pete Warden *“TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers”*, December 2019 |