Smart Elevator

A Graduation Project Thesis

Submitted to the Faculty of Engineering at Helwan University

In Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer and Systems Engineering

Faculty of Engineering, Helwan University, Egypt

By

Noura Medhat Shawky Ahmed

Omar Magdy Yousef Zeneldin

Omar Aladdin Muhammed Attia

Kareem Rabeea Awed Mohamed

Muhammad Fayez Hussien Ahmad

Under the Supervision of

Dr. Maher Mansour

July 2023

Contact Info

Noura Medhat	Medhatnoura90@gmail.com
Omar Magdy	omar_magdyz@yahoo.com
Omar Aladin	omaraladin20@gmail.com
Kareem Rabeea	Kareemrabeea6629@gmail.com
Muhammad Fayez	mohammedfayez691@gmail.com

Table of contents

Table of Figures

Abstract

The convergence of AI systems and embedded systems marks a transformative milestone in technology. Together, they unleash unprecedented potential, revolutionizing industries and enhancing our daily lives. By embedding AI capabilities into hardware and software, intelligent devices emerge, perceiving their surroundings, making real-time decisions, and augmenting human capabilities. This synergy has far-reaching implications, from manufacturing to healthcare, with robots, autonomous vehicles, and AI-powered medical systems reshaping industries. Challenges remain, but the pursuit of this convergence continues to push innovation, promising a smarter, more connected world.

Chapter 1. Introduction

1.1 Overview

In an era of rapid technological advancements, the integration of artificial intelligence (AI) has revolutionized numerous aspects of our daily lives. One such area that has benefited from AI is elevator systems, where traditional methods are being transformed into smart solutions capable of addressing diverse user needs. This project aims to develop a Smart Elevator, leveraging AI technologies such as Text-to-Speech (TTS), object detection, wake word detection, and Speech-to-Text (STT) models. By incorporating these AI capabilities, the Smart Elevator enhances user experience, provides increased accessibility, and tackles challenges posed by various situations, including pandemics like the COVID-19 outbreak and aiding visually impaired individuals.

1.2 Motivation

The need for smart elevator systems stems from the desire to optimize and streamline the elevator usage process. Traditional elevator systems often lack the ability to effectively communicate with users, resulting in inefficiencies, user confusion, and potential safety concerns. Moreover, the COVID-19 pandemic has emphasized the importance of minimizing physical contact and ensuring a safe environment for all individuals using shared spaces. Additionally, visually impaired individuals face unique challenges when interacting with conventional elevator systems, highlighting the need for accessible and inclusive solutions. The incorporation of AI technologies into elevator systems presents a promising opportunity to address these issues effectively.

1.3 Objectives

The main objectives of this project are as follows:

1.3.1 Enhanced User Interaction

By employing a TTS model, the Smart Elevator establishes a natural and interactive communication channel between the elevator system and the user. This facilitates clear and concise communication, eliminating ambiguity and ensuring that the user's intentions are accurately understood.

1.3.2 Intelligent Detection and Response

Utilizing an object detection model, the Smart Elevator can detect the presence of a person outside the elevator. This allows the system to initiate communication with the user, confirming their intention to use the elevator and avoiding accidental activations. The integration of AI-driven decision-making enables the elevator door to open only upon receiving a positive confirmation from the user.

1.3.3 Seamless User Experience

Inside the elevator, the Smart Elevator employs a wake word model to activate the system upon detecting the specific wake-up word. This feature eliminates the need for physical interaction with buttons or control panels, enhancing convenience and reducing the risk of cross-contamination, particularly during pandemics.

1.3.4 Customized Floor Announcement

The Smart Elevator provides floor-specific announcements to the user, ensuring they are informed of each floor they pass. This feature enables users to easily identify their desired destination and reduces the likelihood of missing their stop, enhancing the overall user experience.

1.3.5 Accessibility for All

By utilizing AI and voice-based control, the Smart Elevator extends its usability to individuals with visual impairments. The system's reliance on voice commands allows visually impaired users to navigate the elevator independently and effortlessly, promoting inclusivity and autonomy.

1.4 How Our System Assists Blind People

One of the key focuses of our Smart Elevator is to provide enhanced accessibility and support for individuals with visual impairments. Traditional elevator systems often present significant challenges for blind or visually impaired individuals, making it difficult for them to navigate and operate elevators independently. Our system incorporates AI technologies to address these challenges and ensure that blind users can utilize elevators with ease and confidence.

1.4.1 Voice-Based Interaction

The foundation of our system's assistance to blind people lies in its voice-based interaction capabilities. By leveraging a TTS model and STT model, the Smart Elevator enables blind users to control the elevator solely through their voice commands. This eliminates the need for visual cues or physical interactions, empowering blind individuals to navigate the elevator system independently and safely.

1.4.2 Wake Word Activation

To initiate the system inside the elevator, blind users can simply use the designated wake-up word, followed by the desired floor number. This wake word activation feature eliminates the need for visually locating and pressing buttons or switches, which can be challenging for blind users. By relying on voice commands, our system ensures that blind individuals can effortlessly communicate their desired floor, enabling them to reach their destinations accurately.

1.4.3 Floor-Specific Announcements

During the elevator journey, our system provides floor-specific announcements, informing blind users about the current floor they are passing. These announcements serve as important auditory cues, enabling blind individuals to keep track of their progress and anticipate their desired destination. By leveraging the capabilities of our TTS model, the system delivers clear and concise floor announcements, ensuring that blind users stay informed and aware throughout their elevator experience.

1.4.4 Increased Safety and Confidence

The integration of AI technologies in our system not only enhances accessibility but also contributes to the safety and confidence of blind users. By eliminating the need for physical interactions and incorporating voice-based controls, the risk of accidental button presses or confusion is greatly reduced. Blind individuals can rely on the accuracy and reliability of the system, allowing them to confidently utilize elevators without the fear of making mistakes or getting lost.

1.4.5 Promoting Inclusivity

By catering to the needs of blind users, our Smart Elevator promotes inclusivity and equal access to essential services. It removes barriers that traditionally hindered visually impaired individuals from navigating and using elevators independently. Through the implementation of AI technologies, our system empowers blind users, granting them the freedom to travel and access different floors with ease, thus fostering a more inclusive and equitable environment.

1.5 Literature Review

This part provides a historical overview of the evolution of smart elevator systems, AI integration, and accessibility solutions for individuals with visual impairments. By examining the key milestones and advancements in these areas, we gain insights into the historical context that has shaped the development of our Smart Elevator, with a specific focus on its relevance to assisting blind users.

1.5.1 Smart Elevator Systems [1]

The concept of smart elevator systems emerged in the late 20th century with the advent of computerized controls and sensors. In the 1980s, elevator manufacturers began incorporating microprocessors and advanced algorithms to optimize elevator operations, reducing waiting times and improving energy efficiency. These early developments laid the foundation for the integration of AI technologies in elevator systems

1.5.2 AI Integration in Elevator Systems [2]

The integration of AI technologies in elevator systems gained momentum in the early 21st century. Computer vision algorithms, initially developed for object detection and recognition, found applications in elevator safety and security. In 2001, the introduction of OpenCV (Open Source Computer Vision Library) provided a framework for implementing computer vision capabilities in various domains, including elevator systems. This allowed for the detection of individuals in front of elevators, enabling proactive interactions and enhancing user safety.

1.5.3 Accessibility Solutions for Visual Impairments [3]

The quest for improving accessibility in elevators for individuals with visual impairments has a long history. In the 1960s, braille buttons were introduced to enable tactile interaction with elevator controls. However, it was not until the late 20th century that more comprehensive accessibility solutions began to emerge. In the 1980s, audible floor announcements became more prevalent, providing auditory cues for blind users. These early accessibility features set the stage for further advancements in the integration of AI technologies.

1.5.4 AI-Assisted Navigation for Blind Individuals [4]

Advancements in AI-assisted navigation for blind individuals began to accelerate in the early 2000s. Computer vision algorithms, combined with wearable devices and haptic feedback systems, were utilized to assist blind individuals in detecting obstacles and navigating their surroundings. Simultaneously, speech recognition and synthesis technologies improved, enabling voice-based interactions and navigation assistance. These developments laid the groundwork for the application of AI in elevators to assist blind users.

1.5.5 Integration of AI for Blind-Friendly Elevators

The integration of AI technologies specifically tailored for blind users in elevator systems is a relatively recent focus. Research and development in this area gained traction in the past decade. Researchers started exploring the combination of AI-powered object detection, voice recognition, and natural language processing techniques to create blind-friendly elevator environments. By leveraging AI advancements, elevators can offer enhanced accessibility, independence, and safety for blind individuals.

1.5.6 Summary

The historical review demonstrates the progression of smart elevator systems, AI integration, and accessibility solutions for individuals with visual impairments. From the early incorporation of microprocessors and algorithms in elevator operations to the more recent developments in computer vision and speech technologies, the evolution has paved the way for our Smart Elevator. By integrating AI capabilities

for object detection, voice recognition, and navigation assistance, our system aims to address the unique needs of blind users, contributing to the ongoing efforts to create inclusive and accessible elevator environments.

Chapter 2. System Block Diagram

2.1 System Block Diagram

Our system provides two scenarios, one inside the elevator and the other outside it. Both scenarios are illustrated as follows:

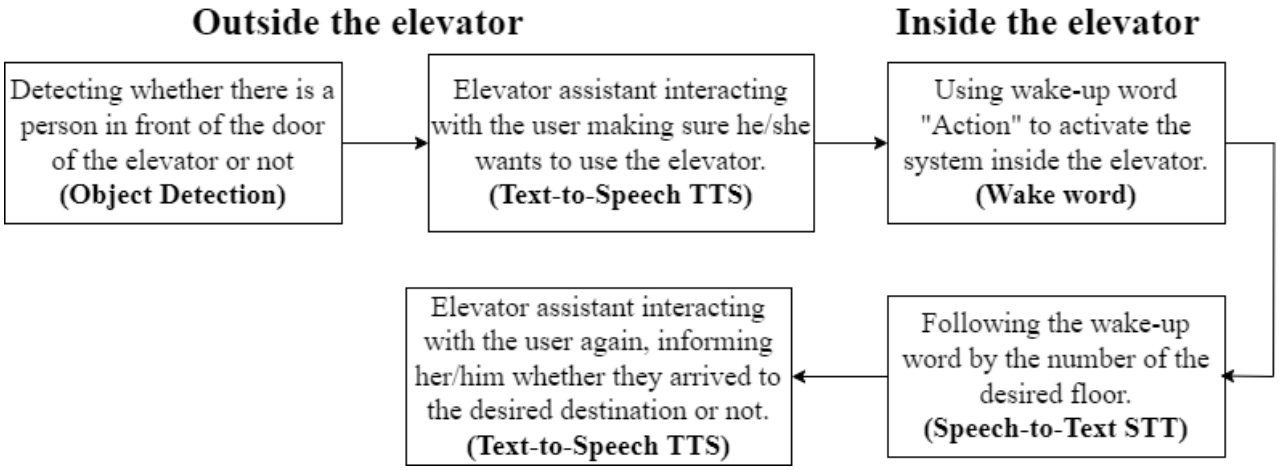

Figure 2.1: System Description

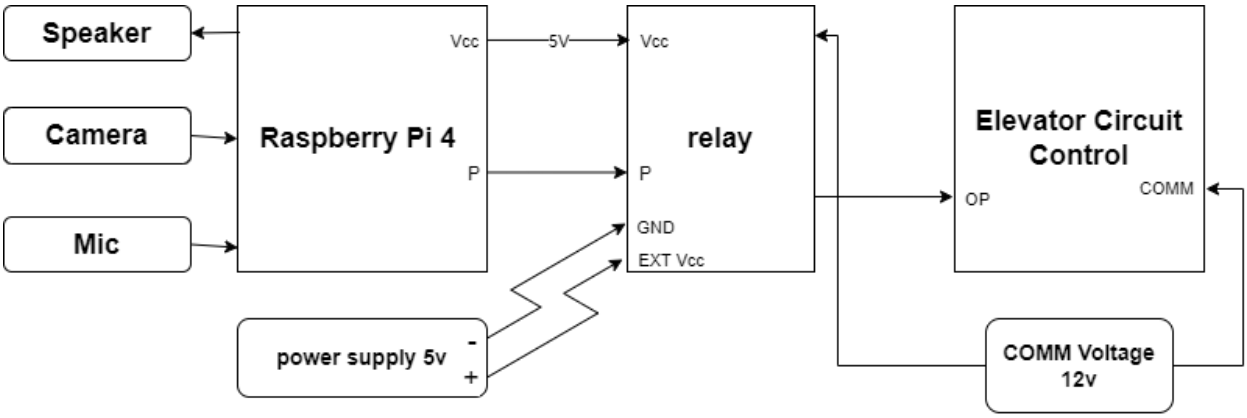

Figure 2.2a System Block Diagram "Outside the Elevator"

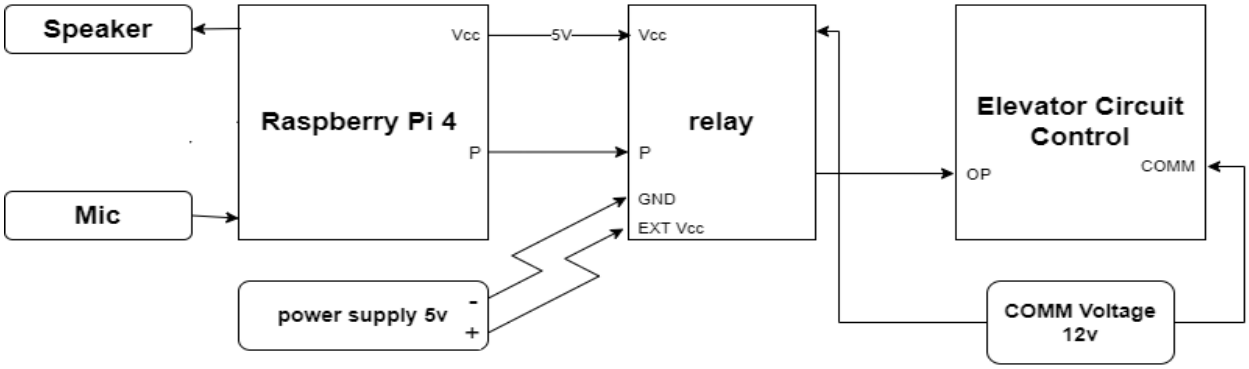

Figure 2.2b System Block Diagram "Inside the Elevator"

2.2 Descriptive Scenario for the System

- **Outside the elevator:**

  An object detection model built using OpenCV is used to detect whether there is a person in front of the elevator or not. In case of detection, a Text-to-Speech model will start to interact with the user, making sure that he wants to use the elevator and that he was not detected by accident. If the answer given by the user was "yes" to the question "Do you want to use the elevator?", the elevator will be requested and the user will be informed with its arrival.

- **Inside the elevator:**

  The user has to use the wake-up word, which in our system is action, in order to activate the system inside the elevator using the wake word model. The wake-up word should be followed by the desired floor number.

  Then, the user will be informed whether he has reached his desired destination or not.

Our old system used an object detection model in order to detect whether the number of people inside the elevator has increased or not, and if it has, a TTS model will start interacting with the user to get the floor number from him. This caused latency when interacting with more than one user at the same time. So, instead, we have developed a wake-word model to handle the case of multiple users interacting with the elevator at the same time.

In order to save power and protect the system from misuse, the object detection model and the TTS were used to only activate the system after making sure that there is a user and that this user wants to use the system.

2.3 System Flowchart

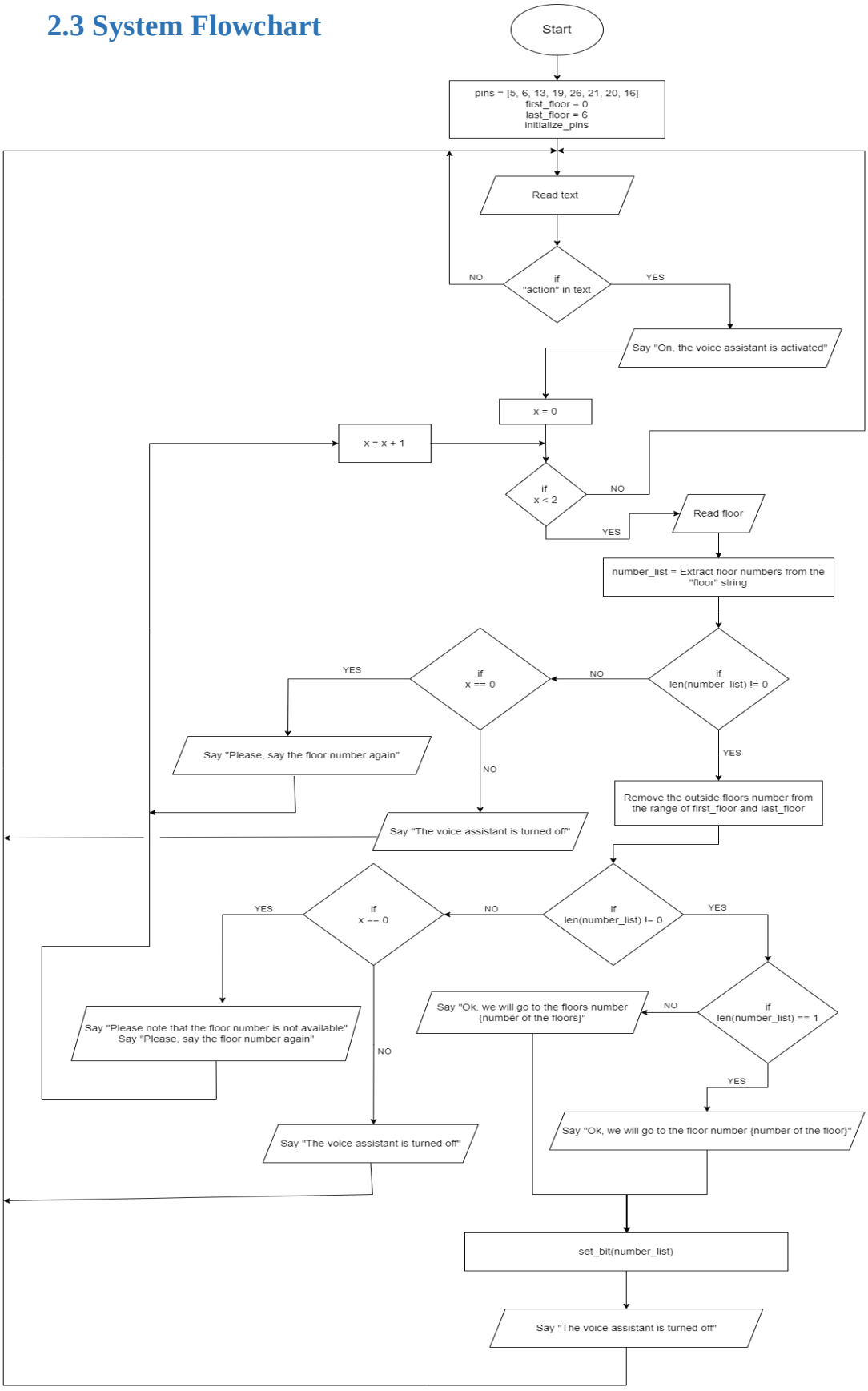

Figure 2.3a: System Flowchart

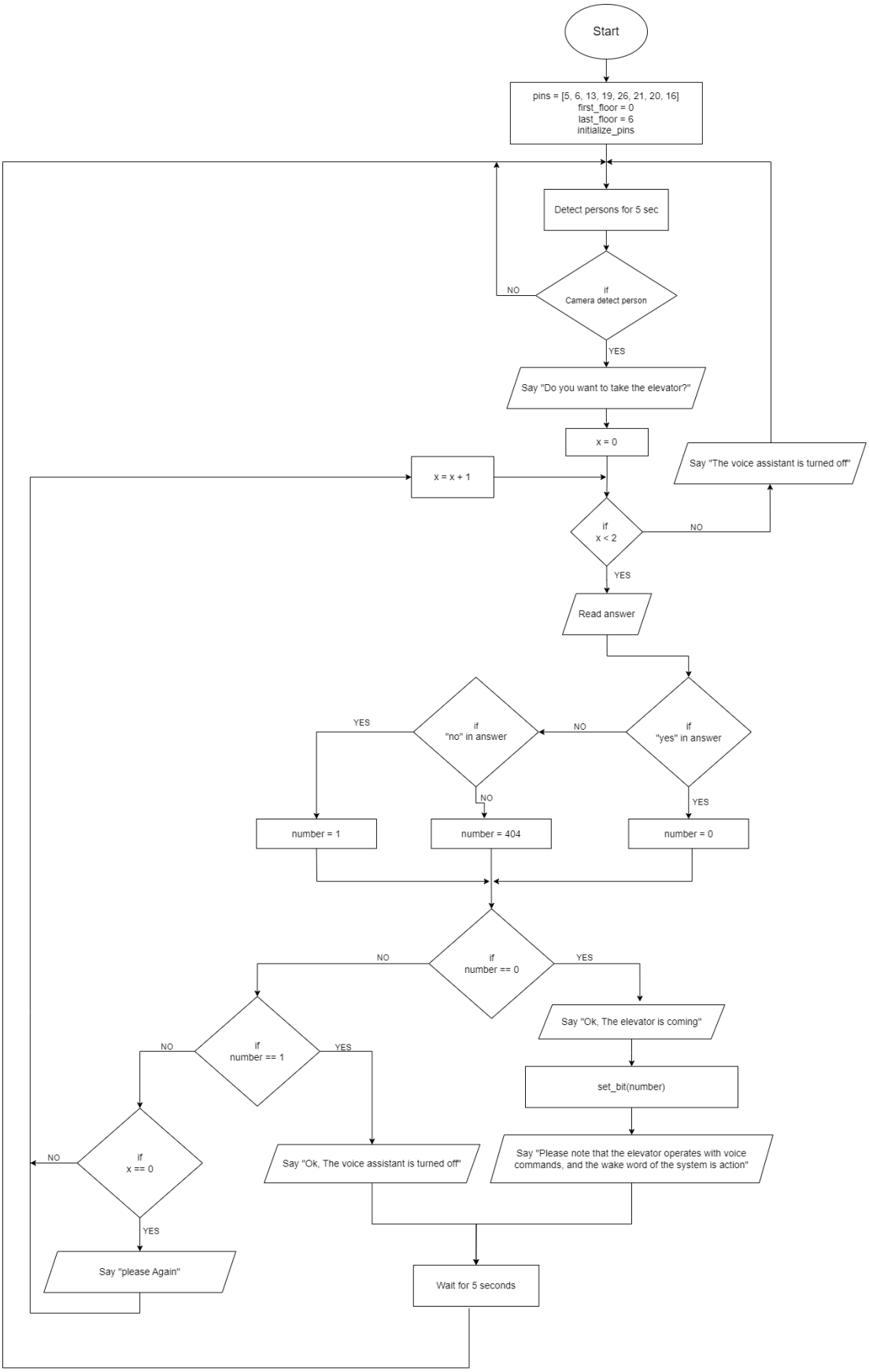

Figure 2.3b: System Flowchart

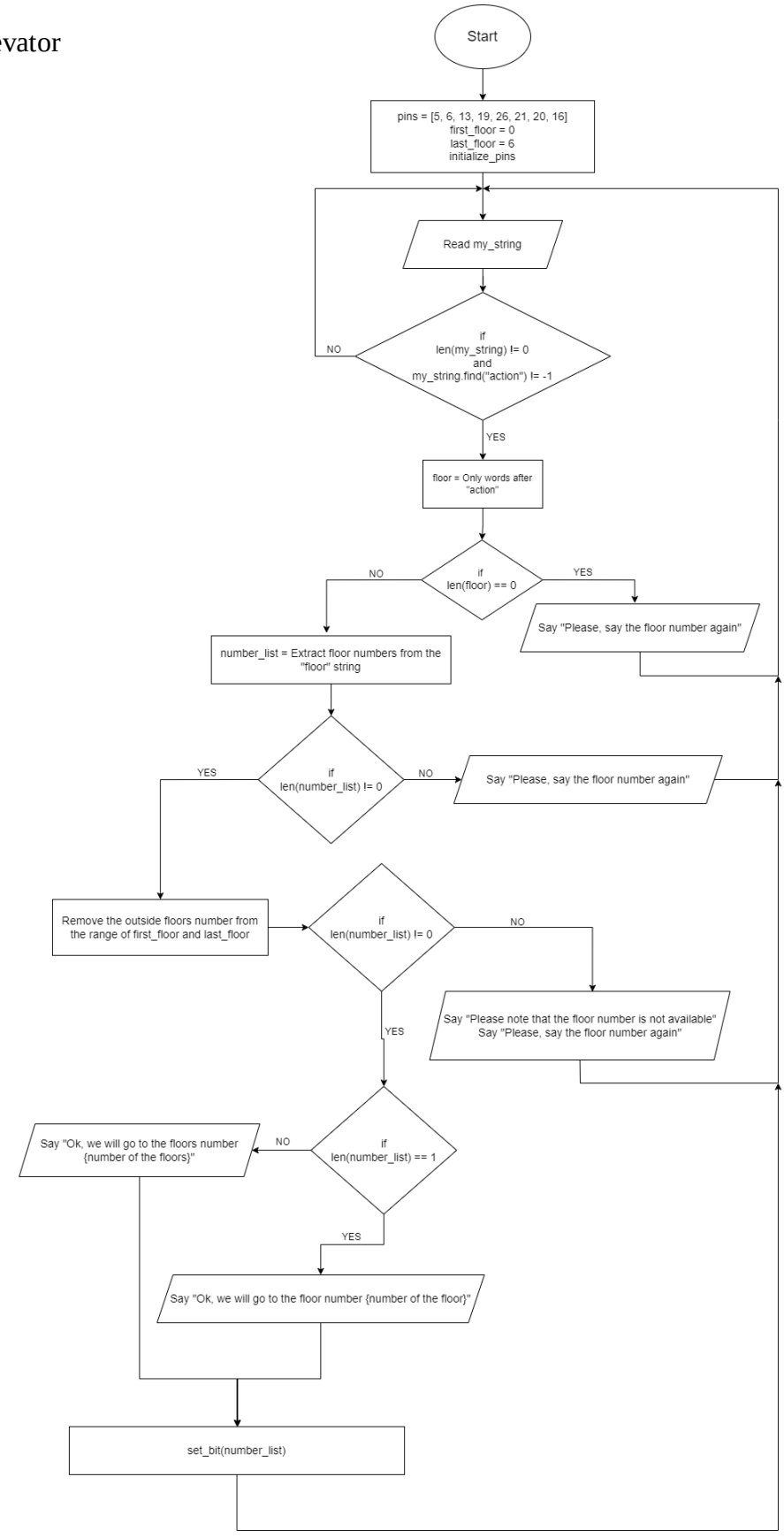

Figure 2.3c: System Flowchart

Chapter 3. AI Models

3.1 Text-to-Speech models

3.1.1 Python Test-to-Speech Version 3 (Pyttsx3) [5]

Pyttsx3 is a Python library for offline text-to-speech conversion. It provides a simple and cross-platform interface to convert written text into spoken words using the text-to-speech engines available on the user's computer.

- **Advantages of Pyttsx3:**

1. Platform Independence: It is designed to work seamlessly across different operating systems, including Windows, macOS, and Linux. This allows developers to write text-to-speech applications that can run on various platforms without requiring significant modifications.
2. Multiple Speech Engines: Pyttsx3 supports multiple text-to-speech engines, such as Microsoft Speech Platform, eSpeak, and macOS's built-in speech synthesis. This gives users the flexibility to choose the engine that best suits their needs or preferences.
3. Customization Options: The library provides various parameters for customization, including voice selection, speech rate, volume, and more. Users can tailor the speech output according to their requirements.
4. Event-driven Architecture: Pyttsx3 follows an event-driven architecture, allowing users to attach event handlers for specific events like the completion of speech or errors. This enables developers to create interactive applications that respond to events during the speech synthesis process.
5. Offline Speech Synthesis: Unlike some online text-to-speech services, Pyttsx3 performs the text-to-speech conversion offline. This ensures that the process can be carried out without relying on an internet connection or external APIs.
6. Open Source and Free: Pyttsx3 is an open-source library released under the MIT license. It is freely available for use and can be modified to suit specific project requirements.

- **Limitations of Pyttsx3:**

1. Limited Language Support: Pyttsx3 relies on platform-specific speech synthesis engines, such as Microsoft Speech API (SAPI) on Windows and NSSpeechSynthesizer on macOS. The availability and language support of these engines may vary. As a result, the range of supported languages and accents may be limited compared to other text-to-speech solutions
2. Voice Variety: The choice of voices in pyttsx3 can be dependent on the underlying platform's speech synthesis engine. This may result in a limited selection of voices, especially when using cross-platform applications.
3. Speech Quality: The speech quality of pyttsx3 is dependent on the capabilities and characteristics of the underlying speech synthesis engine. The quality may vary among different platforms and voices. In some cases, the synthesized speech may sound somewhat robotic or lack the naturalness and expressiveness found in premium or dedicated text-to-speech engines.
4. Customization Options: While pyttsx3 allows limited customization options, such as adjusting the speech rate and volume, it may lack advanced customization features found in more specialized text-to-speech solutions. Fine-grained control over prosody, pitch, or pronunciation adjustments may be limited.
5. Platform Dependencies: Pyttsx3 relies on platform-specific speech synthesis engines, which may introduce dependencies and compatibility issues across different operating systems. Care should be taken to ensure compatibility when deploying applications using pyttsx3 on different platforms.
6. Multithreading Limitations: Pyttsx3 may have limitations when used in multithreaded environments. Some users have reported issues related to threading, such as voice interruptions or freezing when used concurrently with other threads or processes.

3.1.2 eSpeak Text-to-Speech (eSpeak TTS) [6]

eSpeak is a compact and open-source software speech synthesizer that converts written text into spoken words. It is designed to be lightweight and efficient, making it suitable for a wide range of applications, including embedded systems, text-to-speech synthesis, screen readers, and accessibility tools.

eSpeak supports multiple languages and offers various features, including different speech synthesis techniques, pronunciation rules, and customizable voices. It uses a

formant synthesis method to generate speech by modeling the vocal tract and producing the corresponding sound signals.

- **Advantages of eSpeak TTS:**

1. Pronunciation Rules: eSpeak incorporates pronunciation rules to improve the accuracy of speech synthesis. These rules help handle pronunciation variations, specific language rules, and proper intonation.
2. Compact and Efficient: eSpeak is designed to be lightweight and resource-efficient, making it suitable for use in resource-constrained environments and low-power devices.
3. Customizable Voices: The software provides options for customizing the characteristics of the generated voices, such as pitch, speed, and volume. This allows developers to tailor the speech output to their specific needs.
4. Text Processing: eSpeak supports various text processing features, including word capitalization, punctuation interpretation, and the ability to specify pauses, emphasis, and other speech attributes through specially formatted text.
5. Open Source: eSpeak is released under the GNU General Public License (GPL), making it open-source software. This allows users and developers to access and modify the source code as per the terms of the license.

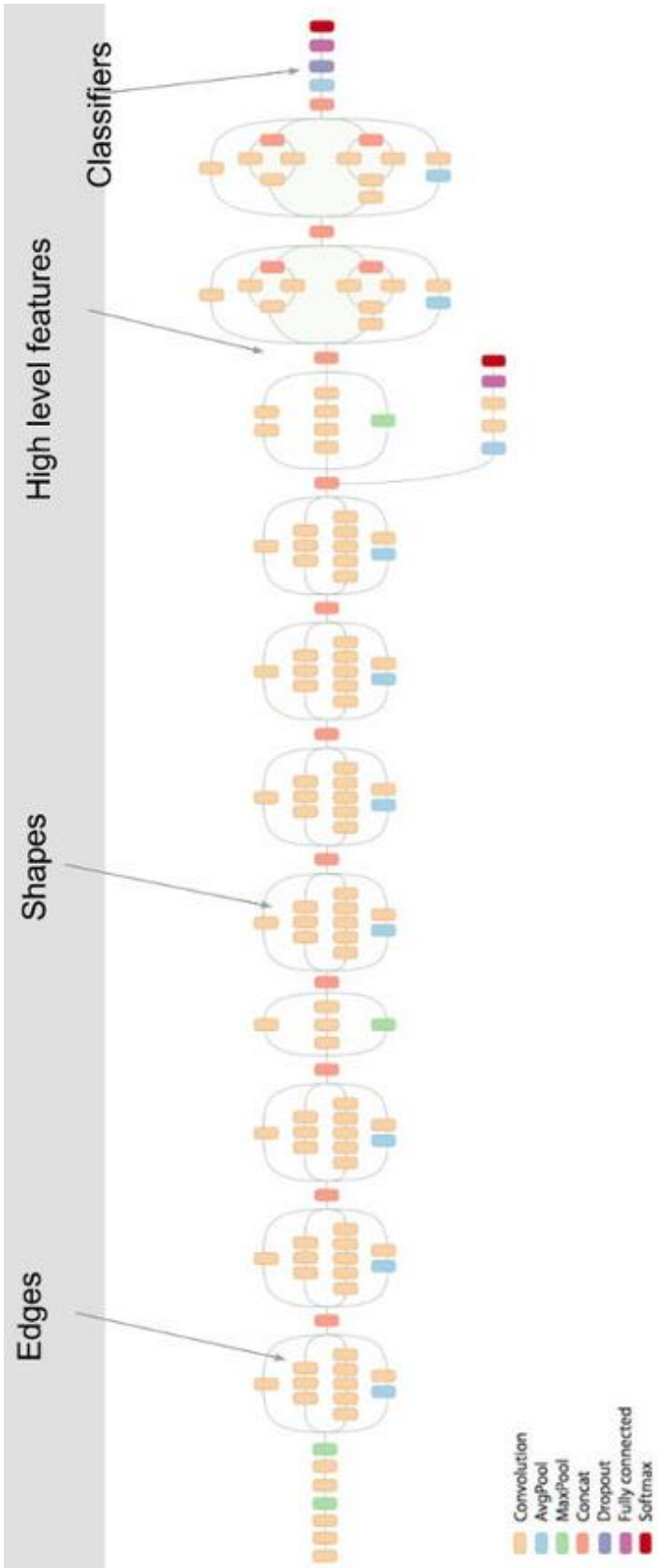

Figure 3.4 eSpeak TTS Architecture

- **Limitations of eSpeak TTS:**

1. Speech Quality: While eSpeak provides intelligible speech, the overall quality of the synthesized voices may not match the naturalness and expressiveness of more advanced or commercial text-to-speech engines. The generated speech can sound somewhat robotic and lack the nuances and variations found in human speech.

2. Limited Language Support: Although eSpeak supports multiple languages, the quality and coverage of different language voices may vary. Certain languages may have more limited support, resulting in potential challenges with accurate pronunciation or handling of specific language features.

3. Pronunciation Accuracy: eSpeak uses a rule-based approach to pronunciation, which means it may encounter difficulties in accurately pronouncing certain words or handling complex or irregular pronunciations. This can result in mispronunciations or unnatural-sounding speech output, especially for non-standard or domain-specific vocabulary.

4. Limited Voice Variety: eSpeak offers a limited selection of pre-defined voices. While it covers multiple languages and accents, the range of available voices might be more limited compared to other text-to-speech solutions. This can impact the variety and diversity of voice options for different applications or user preferences.

5. Customization Options: eSpeak provides limited customization options for adjusting voice characteristics, such as pitch, speed, or intonation. Users have minimal control over modifying the speech output to suit specific preferences or application requirements.

6. Lack of Development and Updates: eSpeak's last official release was in 2011, and development activity has significantly slowed down since then. This means that it may not benefit from ongoing updates, bug fixes, or performance improvements. As a result, it may lack certain features or improvements found in more actively developed text-to-speech systems.

3.1.3 Festival Text-to-Speech (Festival TTS) [7]

Festival TTS (Text-to-Speech) is a free and open-source speech synthesis system developed at the Centre for Speech Technology Research (CSTR) at the University of Edinburgh. It provides a flexible and extensible framework for converting written text into spoken words.

Festival TTS offers a wide range of features and capabilities, making it a popular choice for research, development, and production use in the field of speech synthesis.

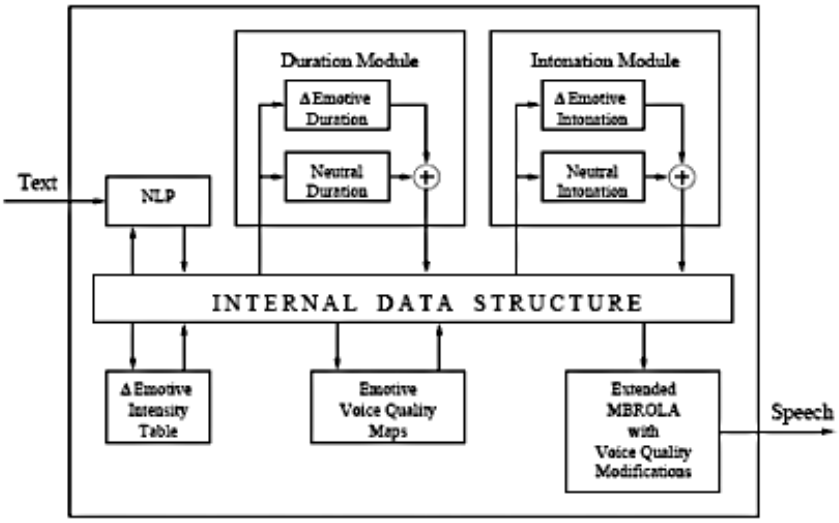

Figure 3.5 Festival TTS Architecture

- **<u>Advantages of Festival TTS:</u>**

1. Multilingual Support: Festival TTS supports multiple languages, allowing developers to generate speech in different languages and accents. It provides language-specific components, such as linguistic models, phonetic rules, and prosody patterns.
2. Modular Architecture: Festival TTS follows a modular architecture, allowing users to customize and extend its functionality. It provides a set of modules that can be combined and configured to control various aspects of the speech synthesis process, such as text analysis, prosody generation, and waveform synthesis.
3. Voice Creation: Festival TTS provides tools and utilities to create new synthetic voices. Users can record voice data, annotate it with linguistic information, and train voice models to produce high-quality and natural-sounding speech.

4. Rich Markup Language: Festival TTS supports a markup language called SSML (Speech Synthesis Markup Language). SSML allows users to add expressive annotations to the input text, such as emphasis, pauses, pitch, and volume changes, enabling fine-grained control over the synthesized speech.
5. Research and Development: Festival TTS is widely used in research and development projects related to speech synthesis, natural language processing, and human-computer interaction. Its flexible architecture and extensive documentation support experimentation, algorithm development, and linguistic analysis.

- **Limitations of Festival TTS:**

1. Complexity and Learning Curve: Festival TTS has a relatively steep learning curve compared to some other text-to-speech systems. Its modular architecture and extensive customization options require a deeper understanding of linguistic, acoustic, and signal processing concepts. This complexity may pose challenges for novice users or those seeking a simpler implementation.
2. Resource Requirements: Festival TTS can be computationally intensive and requires significant system resources to generate speech in real-time. This can limit its performance on lower-powered devices or in resource-constrained environments.
3. Voice Quality: While Festival TTS can produce intelligible speech, the voice quality may not match that of commercial or premium text-to-speech systems. The synthesized speech may sound somewhat robotic or lack the naturalness and expressiveness found in more advanced systems.
4. Limited Language Support: Although Festival TTS supports multiple languages, the availability and quality of language-specific components (e.g., linguistic models, phonetic rules) may vary. Some languages may have more limited support or require additional development effort to achieve satisfactory results.
5. Pronunciation Accuracy: Achieving accurate pronunciation in Festival TTS can be challenging, particularly for complex or irregular pronunciations, non-standard vocabulary, or domain-specific terms. The quality of pronunciation relies on the accuracy and coverage of the linguistic models and rules.
6. Development and Maintenance: While Festival TTS has a strong foundation and community support, the development pace has slowed down in recent years. Updates and bug fixes may not be as frequent compared to more actively

developed text-to-speech systems, potentially leading to a slower response to issues or lack of certain modern features.

7. Integration Complexity: Integrating Festival TTS into larger applications or frameworks can require additional development effort and customization. The modular nature of Festival TTS may necessitate understanding and configuring different components for specific use cases.

3.1.4 Pico Text-to-Speech (Pico TTS) [8]

Pico is a text-to-speech (TTS) engine developed by the Speech Group at the Cambridge University Engineering Department. It is designed to be lightweight and efficient, making it suitable for embedded systems and resource-constrained devices.

Pico TTS provides a way to convert written text into spoken words using a small set of voices. It supports several languages, including English, Spanish, French, German, Italian, and more.

Pico TTS is often used in conjunction with other speech synthesis libraries or applications to provide basic text-to-speech capabilities. It can be integrated into various projects, such as voice assistants, accessibility tools, and voice-enabled applications.

It's worth noting that Pico TTS is one of several available text-to-speech engines, and its usage and availability may depend on the specific software or platform you are working with. It's always a good idea to consult the official documentation or resources specific to the implementation of Pico TTS in your chosen context to ensure accurate information and proper integration.

- **Advantages of Pico TTS:**

1. Lightweight and Efficient: Pico TTS is designed to be lightweight and resource-efficient, making it suitable for embedded systems, mobile devices, and other resource-constrained platforms. It requires minimal processing power and memory, allowing it to run efficiently on devices with limited resources.
2. High Portability: Pico TTS is highly portable and can be easily integrated into various platforms and systems. It has been successfully used in applications for Android, Linux, and other operating systems.
3. Offline Text-to-Speech: Pico TTS performs text-to-speech conversion offline, without requiring an internet connection or external APIs. This is particularly beneficial in situations where internet access is limited or not available.

4. Customization Options: Pico TTS provides some customization options, such as voice selection and pronunciation adjustments. Developers can tailor the speech output to match their application's requirements or user preferences.
5. Open Source: Pico TTS is released as open-source software, allowing developers to study, modify, and distribute the code under the terms of the Apache License. This provides flexibility and encourages community contributions and improvements.
6. Integration Flexibility: Pico TTS can be integrated with other software and applications using APIs or libraries, making it adaptable to a wide range of projects and use cases.

- **Limitations of Pico TTS:**

1. Limited Language Support: Pico TTS has limited language support compared to more comprehensive text-to-speech engines. It primarily focuses on English and a few other languages, which means it may not be suitable for applications or projects requiring support for a wide range of languages.
2. Voice Quality: The voice quality of Pico TTS is often considered less natural and more robotic compared to advanced commercial text-to-speech engines. The synthesized speech may lack expressiveness, natural intonation, and the nuances found in more sophisticated systems.
3. Customization Options: Pico TTS provides limited options for customization. Users have minimal control over adjusting speech parameters like pitch, speed, or voice characteristics. This lack of customization may limit its suitability for applications requiring highly tailored speech output.
4. Pronunciation Accuracy: Pico TTS may struggle with accurately pronouncing certain words or handling complex or irregular pronunciations. Its rule-based approach might not cover all linguistic variations, especially for domain-specific or non-standard vocabulary.
5. Limited Development and Updates: Pico TTS is no longer actively developed or maintained by the original team. The last official release was in 2009, which means it may not benefit from ongoing updates, bug fixes, or performance enhancements. As a result, it may lack some of the improvements seen in more modern text-to-speech systems.
6. Compatibility and Integration: Pico TTS may have limitations when it comes to compatibility with different operating systems, platforms, or programming languages. Integration with certain frameworks or applications may require additional effort or workarounds.

3.1.5 Google Text-to-Speech (gTTS) [9]

gTTS stands for "Google Text-to-Speech." It is a Python library and a command-line interface for converting text into speech using Google's text-to-speech API. With gTTS, you can generate speech from text strings in various languages. The library allows you to control parameters like the language, speech speed, and audio format.

gTTS provides a simple and convenient way to convert written text into spoken words. It can be useful in applications such as creating audio books, adding voice-overs to videos, generating voice prompts, and more. The generated speech can be saved as an audio file or played directly.

It's important to note that gTTS relies on an internet connection as it sends the text to Google's servers for speech synthesis. Therefore, an active internet connection is required to use gTTS effectively.

gTTS (Google Text-to-Speech) supports a wide range of languages. While the exact list of supported languages including Arabic and English.

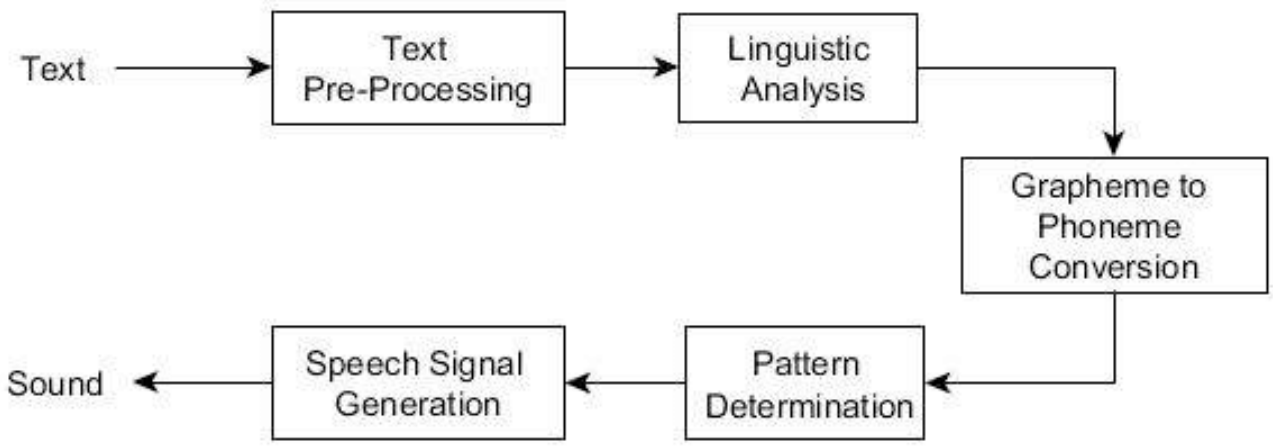

Figure 3.6 gTTS Architecture

- **Advantages of gTTS:**

1. Ease of Use: gTTS provides a straightforward and user-friendly interface for converting text to speech. It has a simple API that allows you to generate speech with just a few lines of code.
2. Multilingual Support: gTTS supports a wide range of languages, allowing you to generate speech in various languages depending on your requirements.

3. Natural-sounding Speech: The text-to-speech engine used by gTTS is powered by Google's speech synthesis technology, which generally produces high-quality and natural-sounding speech output.
4. Customization Options: gTTS allows you to customize parameters such as speech speed, language, and audio format. This flexibility enables you to tailor the generated speech to suit your specific needs.
5. Offline Playback: Although gTTS requires an internet connection to convert text to speech, once the speech is generated, it can be saved as an audio file for offline playback. This can be useful in scenarios where an internet connection is not readily available.
6. Free and Open Source: gTTS is an open-source library, meaning it is freely available for use and can be modified to suit individual requirements. This makes it accessible to a wide range of developers and enthusiasts.
7. Wide Range of Applications: gTTS can be used in various applications, such as creating voice prompts for interactive systems, generating audio versions of text-based content, developing language learning tools, and adding voice-overs to videos or presentations.

- **Limitations of gTTS:**

1. Internet Dependency: gTTS requires an internet connection to function because it relies on Google's cloud-based speech synthesis service. Without internet access, gTTS cannot convert text to speech.
2. Voice Options: gTTS provides a limited set of pre-defined voices. Although these voices cover different languages and accents, there might be constraints in terms of voice variety and customization compared to other text-to-speech solutions.
3. Pronunciation Accuracy: gTTS may encounter challenges with accurately pronouncing certain words or proper nouns, especially from non-standard or domain-specific vocabularies. Since gTTS uses a rule-based system, it may not handle complex or irregular pronunciations as effectively as other specialized speech synthesis systems.
4. TTS Quality: While gTTS provides intelligible speech, the overall quality of synthesized voices may not be on par with more advanced or premium text-to-speech engines. The generated speech might lack naturalness or exhibit some robotic artifacts.
5. API Limitations: gTTS is primarily intended for individual and non-commercial use. It has usage limits and restrictions imposed by Google, such

as restrictions on the number of requests per minute or per day. These limits may impact applications or services that require high-volume or continuous speech synthesis.

6. Customization Options: gTTS does not offer extensive customization options for altering voice characteristics, such as pitch, speed, or intonation. Users have limited control over modifying the speech output to suit specific preferences or application requirements.

3.2 Speech-to-Text Models

3.2.2 CNN Based Models

3.2.2.1 Model One

Dataset: Google's speech commands dataset:

An audio dataset of spoken words designed to help train and evaluate keyword spotting systems. Its primary goal is to provide a way to build and test small models that detect when a single word is spoken, from a set of ten target words, with as few false positives as possible from background noise or unrelated speech. [10]

The dataset is a collection of 35 words, of which we only took the digits and the noise examples.

Feature Extraction Technique: Mel-frequency cepstral coefficients (**MFCC**):

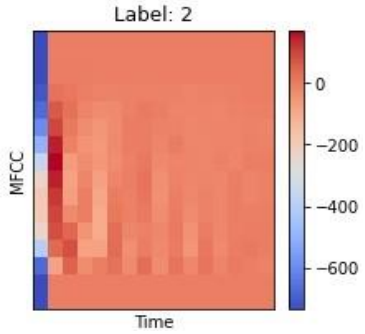

Figure 3.7 MFCCs for an audio file

MFCCs are coefficients that collectively make up an MFC. [11]

They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum").

The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal spectrum.

This frequency warping can allow for better representation of sound, for example, in audio compression that might potentially reduce the transmission bandwidth and the storage requirements of audio signals.

MFCCs are commonly derived as follows: [12]

1) Take the Fourier transform of (a windowed excerpt of) a signal.
2) Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows or alternatively, cosine overlapping windows.
3) Take the logs of the powers at each of the mel frequencies.
4) Take the discrete cosine transform (**DCT**) of the list of mel log powers, as if it were a signal.
5) The MFCCs are the amplitudes of the resulting spectrum.

Model1 Architecture:

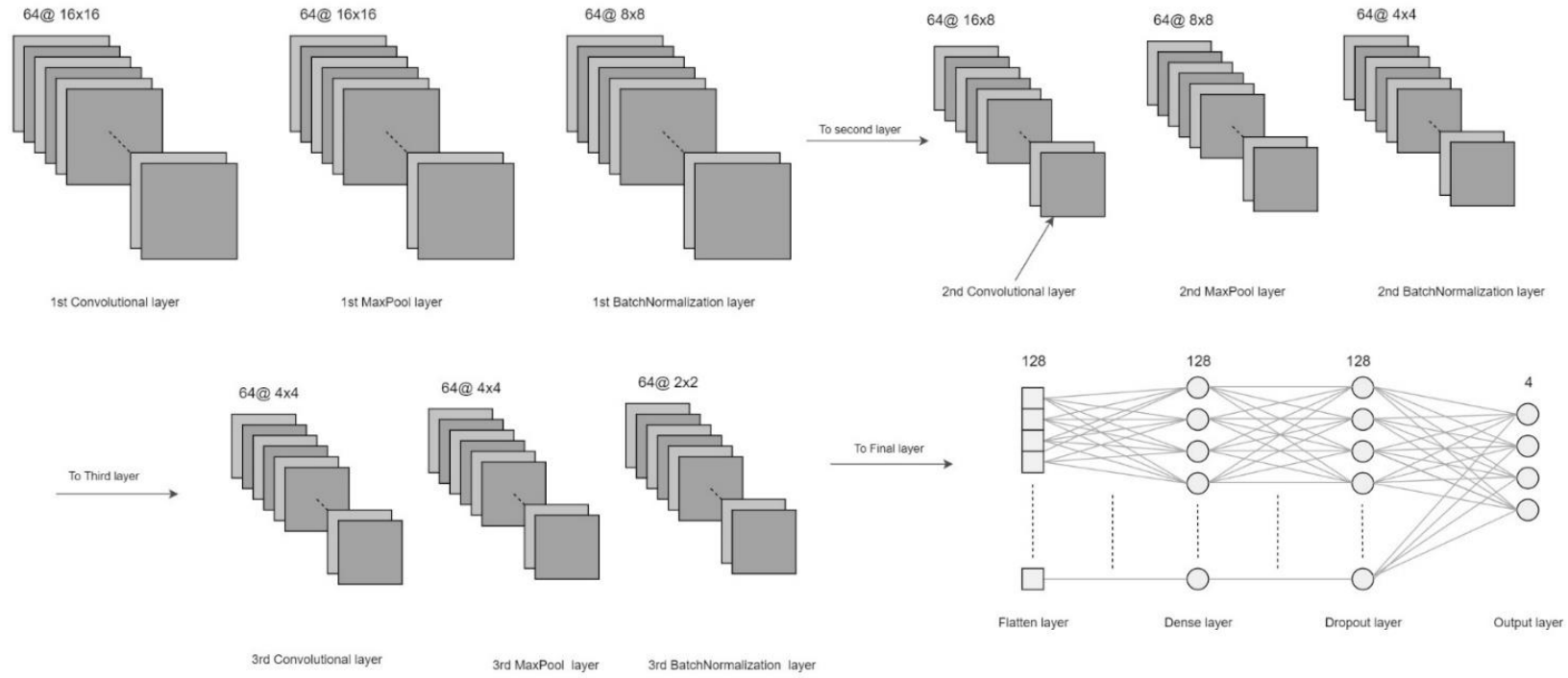

Figure 3.8 Model1 Architecture

Training Compilation Parameters:

- **Epochs:** 25
- **Loss function:** Sparse Categorical Cross entropy.
- Optimizer: Adam optimizer.
- **Metrics:** Accuracy.
- **Learning rate:** 0.001
- **Activation for convolution layers:** ReLU.
- **Activation for output layer:** Softmax.

Training Curve:

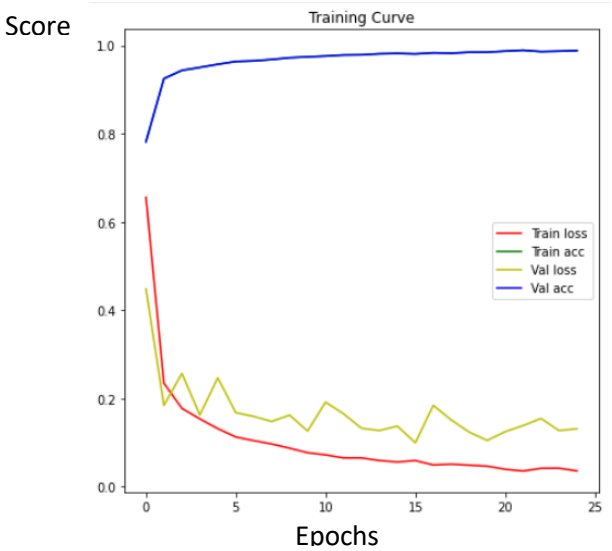

Figure 3.9 Training Curve of Model1

Evaluation:

- **Test accuracy:** 98%
- **Test loss:** 7.3%
- **Total model size:** 822KB.

3.2.2.2 Model 2

Dataset: Free spoken Digits Dataset:

The dataset mainly consists of 3000 records which belong to 11 different classes. The records in the dataset are recorded by 6 different persons, George, Jackman, Lucas, Nicolas, Theo and Yweweler.

Feature Extraction Technique: Spectrogram:

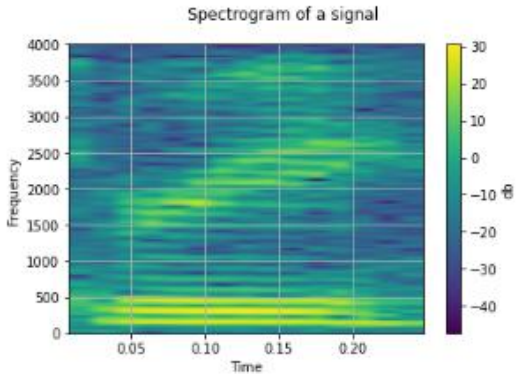

Figure 3.10 Spectrogram of an Audio File

Simply, we are going to convert the audios to spectrograms and save them in this form. Definitely each record has a different spectrogram and this will help us in the classification process.

Model 2 Architecture:

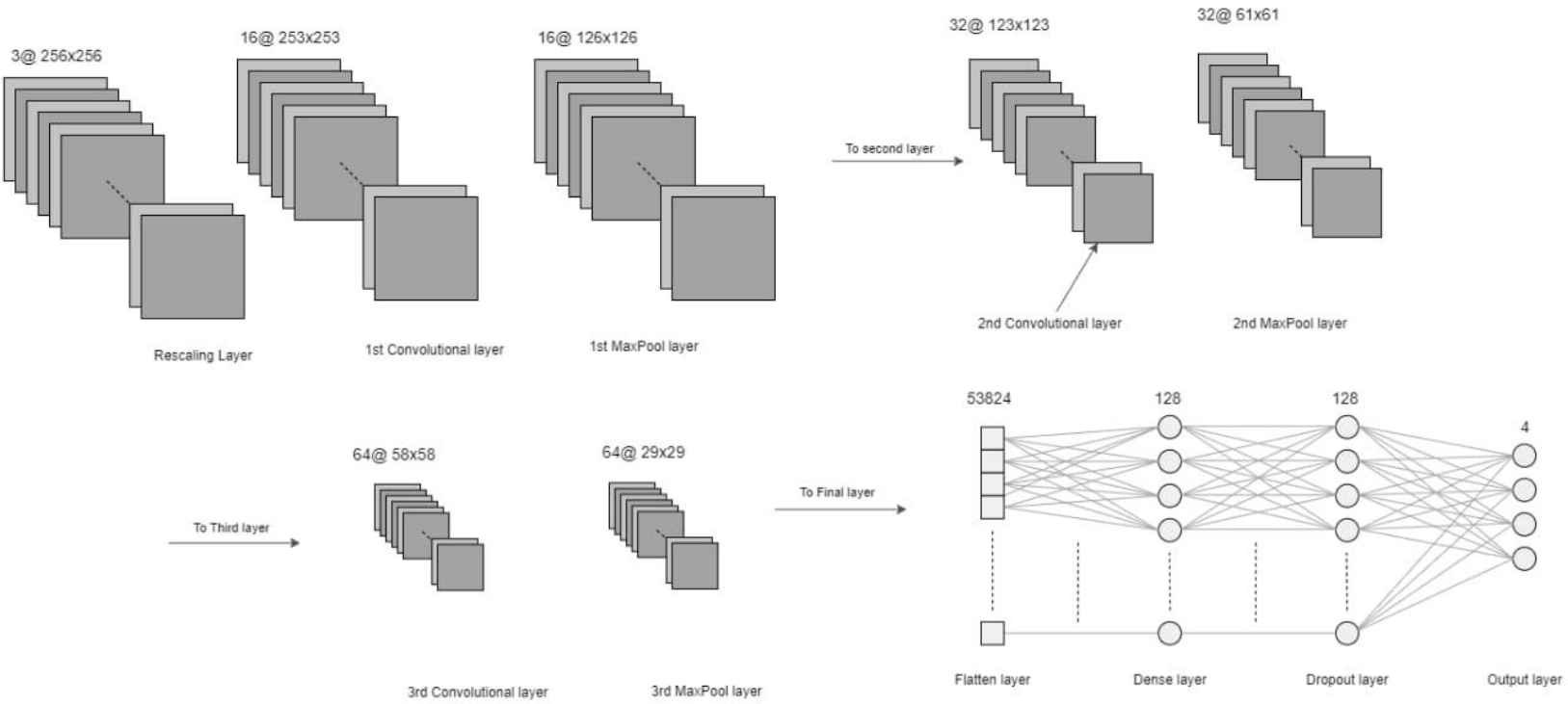

Figure 3.11 Model 2 Architecture

Training and Compilation Parameters:

- **Epochs:** 15
- **Loss function:** Sparse Categorical Cross Entropy.
- **Optimizer:** Adam optimizer.
- **Metrics:** Accuracy.
- **Learning rate:** 0.001
- **Activation for convolution layers:** ReLU.
- **Activation for output layer**: Softmax.

Training Curve:

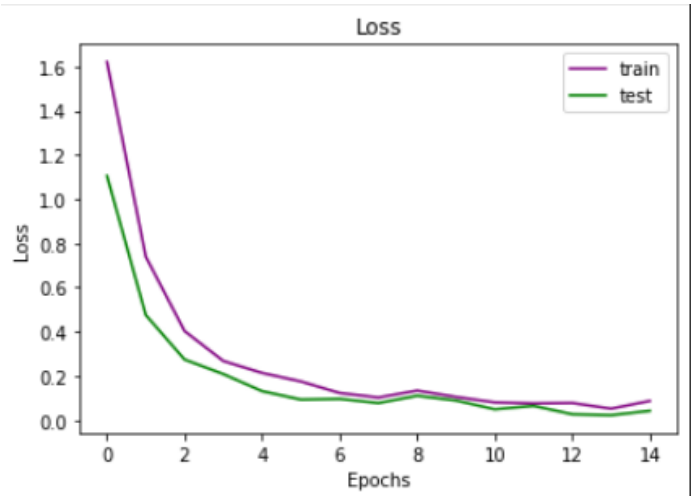

Figure 3.12 Training Curve of Model 2

Evaluation

- **Test accuracy:** 99.59%
- **Test loss:** 2.49%
- **Total model size:** 27.75MB

3.2.2.3 Model 3

Dataset: Arabic Speech Commands Dataset:

The dataset is a list of pairs (x, y), where x is the input speech signal, and y is the corresponding keyword. The final dataset consists of 12000 such pairs, comprising 40 keywords. Each audio file is one-second in length sampled at 16 kHz. We have 30 participants, each of them recorded 10 utterances for each keyword. Therefore, we have 300 audio files for each keyword in total (30 * 10 * 40 = 12000), and the total size of all the recorded keywords is ~384 MB. [13]

Of the 40 keywords, we are only concerned with the digits 0-9, so, the rest is discarded.

Feature Extraction Technique:

Mel-frequency cepstral coefficients (MFCC)

Model 3 Architecture

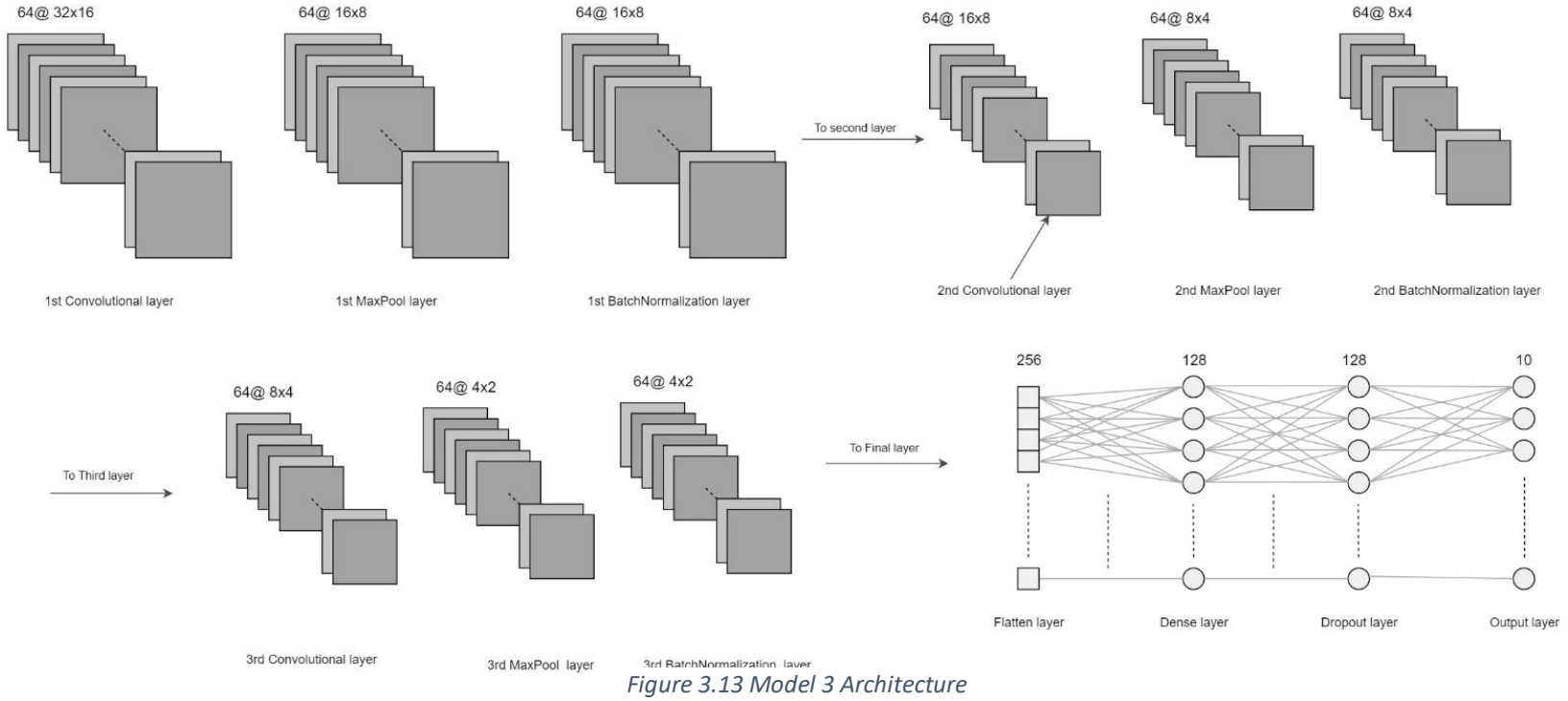

Figure 3.13 Model 3 Architecture

Training and Compilation Parameters:

- **Epochs:** 25
- **Loss function:** Sparse Categorical Cross entropy.
- **Optimizer:** Adam optimizer.
- **Metrics:** Accuracy.
- **Learning rate:** 0.001
- **Activation for convolution layers:** ReLU.
- **Activation for output layer:** Softmax.

Training Curve:

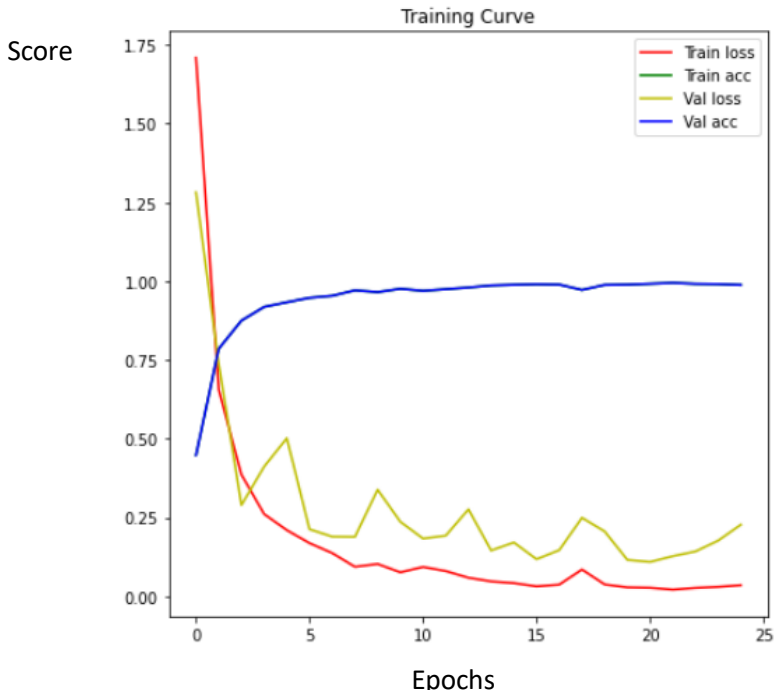

Figure 3.14 Training Curve of Model 3

Evaluation:

- **Test accuracy:** 98.1%
- **Test loss:** 5.3%
- **Total model size:** 1402KB

3.2.3 LSTM Model

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. This is a behavior required in complex problem domains like machine translation, speech recognition, and more. [14]

Recurrent networks have an internal state that can represent context information. They keep information about past inputs for an amount of time that is not fixed a priori, but rather depends on its weights and on the input data. [15]

The success of LSTMs is in their claim to be one of the first implements to overcome the technical problems and deliver on the promise of recurrent neural networks:

> *"Hence standard RNNs fail to learn in the presence of time lags greater than 5 – 10 discrete time steps between relevant input events and target signals. The vanishing error problem casts doubt on whether standard RNNs can indeed exhibit significant practical advantages over time window-based feedforward networks. A recent model, "Long Short-Term Memory" (LSTM), is not affected by this problem. LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing constant error flow through "constant error carrousels" (CECs) within special units, called cells"* [16]

Dataset: Free Spoken Digits Datasets (FSDD)

LSTM Model Architecture:

- LSTM (units=64): An LSTM layer with 64 units.
- Dropout (0.3): A dropout layer with 30% probability.
- LSTM (units=128)
- Dropout (0.3)
- LSTM (units=256)
- Dropout (0.2)
- A final dense layer with 10 neurons (Number of classes which are digits 0-9)

Training and Compilation Parameters:

- **Epochs:** 50
- **Loss function:** Categorical Cross entropy.
- **Optimizer:** Adam optimizer.
- **Metrics:** Accuracy.
- **Learning rate:** 0.001
- **Activation for convolution layers:** ReLU.
- **Activation for output layer:** Softmax.

Training Curve:

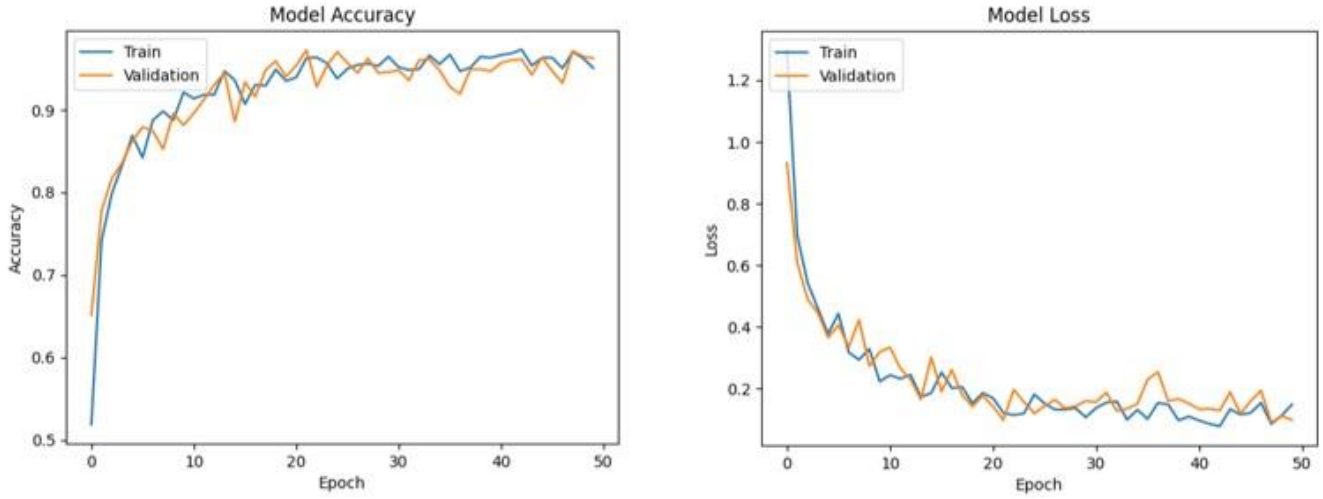

Figure 3.15 LSTM Training Curve

Evaluation:

- **Test accuracy:** 96%
- **Test loss:** 0.09%
- **Total model size:** 6.29MB

3.2.4 Transformers

A transformer is a deep learning model. It is distinguished by its adoption of self-attention, differentially weighting the significance of each part of the input (which includes the recursive output) data. It is used primarily in the fields of natural language processing (NLP), computer vision (CV) and audio processing. Recurrent neural networks, long short-term memory and gated recurrent neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures. [17]

The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU, ByteNet and ConvS2S, all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention.

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations. [18]

End-to-end memory networks are based on a recurrent attention mechanism instead of sequence aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks. [19]

To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. [20]

Transformers Architecture:

- Encoder
- Decoder
- Attention
  - Scaled Dot-Product Attention
  - Multi-Head Attention
- Position-wise Feed-Forward Networks
- Embedding and Softmax
- Positional Encoding

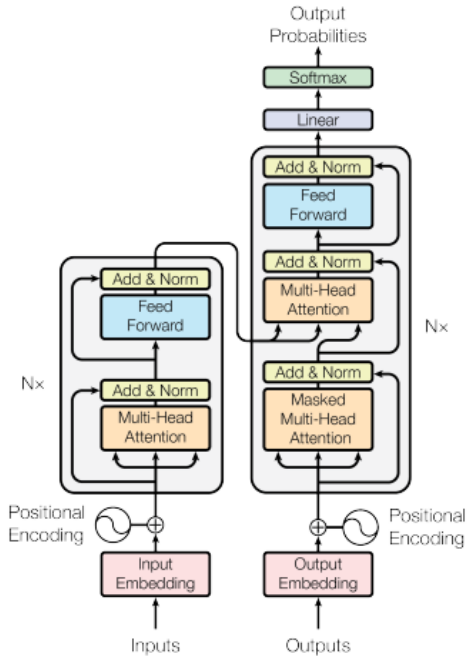

Figure 3.16 Transformer

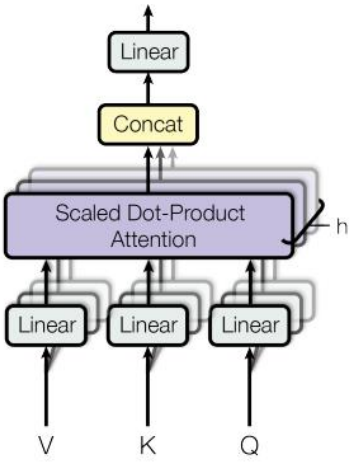

Figure 3.17 Multi-Head Attention

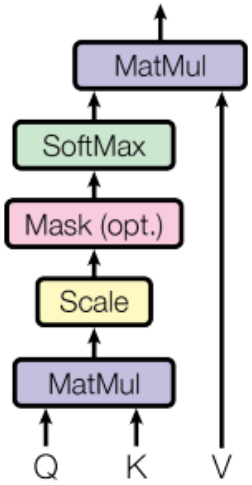

Figure 3.18 Scaled-Dot Product Attention

3.2.4.1 Wav2Vec2 Transformer

Wav2Vec is a framework for self-supervised learning of representations from raw audio data. Basically it learns to efficiently represent the raw audio data as a vector space encoding.

A major advantage of this approach is that we end up training a generic audio model that could be used for multiple downstream tasks! And because of the self-supervised learning, we don't need access to huge amount of labeled data. In the paper, after pre-training on unlabeled speech, the model is fine-tuned on small labeled data with a Connectionist Temporal Classification (CTC) loss for speech recognition task. [21]

Wav2Vec2 Architecture: [21]

- **Feature Encoder:** This is the encoder part of the model. It takes the raw audio data as input and outputs feature vectors. Input size is limited to 400 samples which is 20ms for 16kHz sample rate. The raw audio is first standardized to have zero mean and unit variance. Then it is passed to 1D convolutional neural network (temporal convolution) followed by layer normalization and GELU activation function. There could be 7 such convolution blocks with constant channel size (512), decreasing kernel width (10, 3x4, 2x2) and stride (5, 2x6). The output is list of feature vectors each with 512 dimensions.
- **Transformers:** The output of the feature encoder is passed on to a transformer layer. One differentiator is use of relative positional embedding by using convolution layers, rather than using fixed positional encoding as done in original Transformers paper. The block size differs, as 12 transformers block with model dimension of 768 is used in BASE model but 24 blocks with 1024 dimension in LARGE version.
- **Quantization Module:** For self-supervised learning, we need to work with discrete outputs. For this, there is a quantization module that converts the continuous vector output to discrete representations, and on top of it, it automatically learns the discrete speech units. This is done by maintaining multiple codebooks/groups (320 in size) and the units are sampled from each codebook are later concatenated (320x320=102400 possible speech units). The sampling is done using Gumbel-Softmax which is like argmax but differentiable.

Problems we faced with Wav2Vec2: In order for Wav2Vec to be able to classify spoken digits with high accuracy it must be fine-tuned on a dataset specific for this task. This process of fine-tuning is a complex task. Here are a few reasons why this task can be challenging:

- **Data Variability:** The spoken digits dataset may contain recordings from speakers with different accents, dialects, and speech patterns. This variability can make it challenging to train a model that can generalize well to new speakers.
- **Feature Extraction and Pre-processing:** The wav2vec2 model is designed to extract high-level representations of speech signals but it may not always capture the specific features that are relevant for spoken digits recognition. Pre-processing the data and designing appropriate feature extraction methods can be time-consuming and require expert knowledge.
- **Model Architecture and Hyperparameters**: Choosing an appropriate model architecture and hyperparameters can be challenging and require extensive experimentation. The wav2vec2 model is a complex deep learning architecture with many hyperparameters that can affect model performance.

3.2.5 VOSK Offline Speech Kit (VOSK)

VOSK is an offline open source speech recognition toolkit. It enables speech recognition for 20+ languages and dialects - English, Indian English, German, French, Spanish, Portuguese, Chinese, Russian, Turkish, Vietnamese, Italian, Dutch, Catalan, Arabic, Greek, Farsi, Filipino, Ukrainian, Kazakh, Swedish, Japanese, Esperanto, Hindi, Czech, and Polish. More to come. VOSK models are small (50 Mb) but provide continuous large vocabulary transcription, zero-latency response with streaming API, reconfigurable vocabulary and speaker identification. [22]

VOSK supplies speech recognition for Chatbots, smart home appliances, and virtual assistants. It can also create subtitles for movies, transcription for lectures and interviews. VOSK scales from small devices like Raspberry Pi or Android smartphone to big clusters.

We have two types of models - big and small, small models are ideal for some limited task on mobile applications. They can run on smartphones, Raspberry Pi's. They are also recommended for desktop applications. Small model typically is around 50Mb in size and requires about 300Mb of memory in runtime. Big models are for the high-accuracy transcription on the server. Big models require up to 16Gb in memory since they apply advanced AI algorithms. Ideally you run them on some high-end servers like i7 or latest AMD Ryzen. On AWS you can take a look on c5a machines and similar machines in other clouds. [23]

VOSK Installation: The easiest way to install VOSK API is with pip. We do not have to compile anything.

1. Making sure that we have an up-to-date pip and Python versions
   - Python version: 3.5-3.9
   - pip version: 20.3 and newer
2. Upgrading Python and pip if needed and then installing VOSK on Linux/MAC using pip

pip3 install vosk

Currently Support Platforms:

- Linux on x86 and 64
- Raspbian on Raspberry Pi ¾
- Linux on ARM64
- OSX (Both x86 and M1)
- Window x86 and 64

Platforms that does not support it:

- ARMv6 (Rpi zero is too slow)
- Windows ARM64

3.2.6 DeepSpeech

DeepSpeech is an open source Speech-To-Text engine, using a model trained by machine learning techniques based on Baidu's Deep Speech research paper. Project DeepSpeech uses Google's TensorFlow to make the implementation easier.

Top speech recognition systems rely on sophisticated pipelines composed of multiple algorithms and hand-engineered processing stages. "DeepSpeech" is an end-to-end speech system, where deep learning supersedes these processing stages. Combined with a language model, this approach achieves higher performance than traditional methods on hard speech recognition tasks while also being much simpler. These results are made possible by training a large recurrent neural network (RNN) using multiple GPUs and thousands of hours of data. Because this system learns directly from data, we do not require specialized components for speaker adaptation or noise filtering. In fact, in settings where robustness to speaker variation and noise are critical, our system excels: Deep Speech outperforms previously published methods on the Switchboard Hub5'00 corpus, achieving 16.0% error, and performs better than commercial systems in noisy speech recognition tests. [24]

3.2.6.1 Language Model

When trained from large quantities of labeled speech data, the RNN model can learn to produce readable character-level transcriptions. Indeed for many of the transcriptions, the most likely character sequence predicted by the RNN is exactly correct without external language constraints. The errors made by the RNN in this case tend to be phonetically plausible renderings of English words—**Figure 3.19** shows some examples. Many of the errors occur on words that rarely or never appear in our training set. In practice, this is hard to avoid: training from enough speech data to hear all of the words or language constructions we might need to know is impractical. Therefore, we integrate our system with an N-gram language model since these models are easily trained from huge unlabeled text corpora. For comparison, while our speech datasets typically include up to 3 million utterances, the N-gram language model used for the experiments is trained from a corpus of 220 million phrases, supporting a vocabulary of 495,000 words.

RNN output	Decoded Transcription
what is the weather like in bostin right now	what is the weather like in boston right now
prime miniter nerenr modi	prime minister narendra modi
arther n tickets for the game	are there any tickets for the game

Figure 3.19: Examples of transcriptions directly from the RNN (left) with errors that are fixed by addition of a language model (right)

3.2.6.2 Training Data

Large-scale deep learning systems require an abundance of labeled data. For this system we need many recorded utterances and corresponding English transcriptions, but there are few public datasets of sufficient scale. To train our largest models an extensive dataset consisting of 5000 hours of read speech from 9600 speakers was collected.

3.2.6 Selecting a Speech-to-Text Model for the Project

We have selecting a model based on the performance of each model.

- **CNN Models Performance:** All models provided transcription results with high accuracy on recorded test data. The percentage of correctly transcribed words varied slightly between models. Unfortunately, when testing in real-time, only model 3 (the model trained on Arabic Speech Commands) could predict certain words correctly, but fails to predict other words and gives wrong answers. All other CNN models performed poorly in real-time testing. In addition, as the models provide the output as a percentage of how it is sure that the audio it was fed with is a certain class (digit), when selecting the threshold of this percentage above which the model will output the predicted class, it did not provide output for most uttered digits, which means the confidence of the prediction is very low. So, according to the previously mentioned drawbacks, we had to find a better alternative for our project other than CNN-based models.
- **LSTM Model:** Same as CNN models, LSTM model provided high accuracy output with recorded data, but failed to match this level of accuracy for real-time testing.
- **Wav2Vec2 (Transformer Model):** The model showed an unacceptable weakness in predicting digits, which is our main purpose of using Speech recognition models. To solve this problem we had to fine-tune the model on spoken digits, which we discovered -by search and trial- to be complex and requires an experience in fine-tuning models and demands a lot of effort and time.
- **VOSK Model Performance:** VOSK model is selected for the project as it provided a splendid performance in real-time testing on both PC and Raspberry Pi and it is the best model that fits our project needs.

3.3 Object Detection Models

Object Detection is one of the most famous and vastly researched topics in the field of Computer Vision and Machine Learning. It has attracted many researchers working in different areas such as computer vision, robotics, medical imaging, mechanical engineering, and telecommunications. Object Detection is a methodology in Machine Learning focused on localizing and recognizing distinct objects in images and videos. Every object has discrete features that help in distinguishing it in a photo or a video frame. Object Detection methodologies use those features in determining objects, identifying, and labeling them. Methods for object detection can fall under machine learning-based approaches and deep learning-based approaches. Deep Learning methodologies are able to do end-to-end object detection using Convolutional Neural Networks. The algorithms designed to do object detection are based on two approaches - one-stage object detection and two-stage object detection. [25]

One-stage detectors have high inference speeds and two-stage detectors have high localization and recognition accuracy. The two stages of a two-stage detector can be divided by a RoI (Region of Interest) Pooling layer. One of the prominent two-stage object detectors is Faster R-CNN. It has the first stage called RPN, a Region Proposal Network to predict candidate bounding boxes. In the second stage, features are by RoI pooling operation from each candidate box for the following classification and bounding box regression tasks [26]. In contrast, a one-stage detector predicts bounding boxes in a single-step without using region proposals. It leverages the help of a grid box and anchors to localize the region of detection in the image and constraint the shape of the object.

YOLO (You Only Look Once), a popular object detection and image segmentation model, was developed by Joseph Redmon and Ali Farhadi at the University of Washington. Launched in 2015, and has since undergone several iterations, the latest being YOLOv8 and YOLO-NAS. YOLO quickly gained popularity for its high speed and accuracy. YOLOv8 was launched on January 10th, 2023.

Introducing Ultralytics YOLOv8, the latest version of the acclaimed real-time object detection and image segmentation model. YOLOv8 is built on cutting-edge advancements in deep learning and computer vision, offering unparalleled performance in terms of speed and accuracy. Its streamlined design makes it suitable for various applications and easily adaptable to different hardware platforms, from edge devices to cloud APIs.

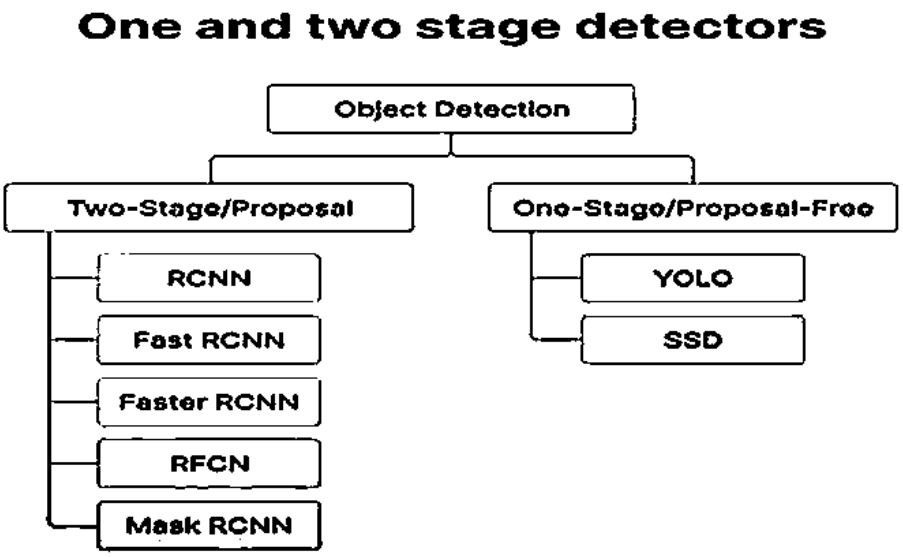

Figure 3.20 Examples of one-stage and two-stage detectors

3.3.1 YOLOv8

YOLOv8 Architecture

YOLOv8 does not yet have a published paper, so we lack direct insight into the direct research methodology and ablation studies done during its creation.The following image made by GitHub user RangeKing shows a detailed visualization of the network's architecture. [27]

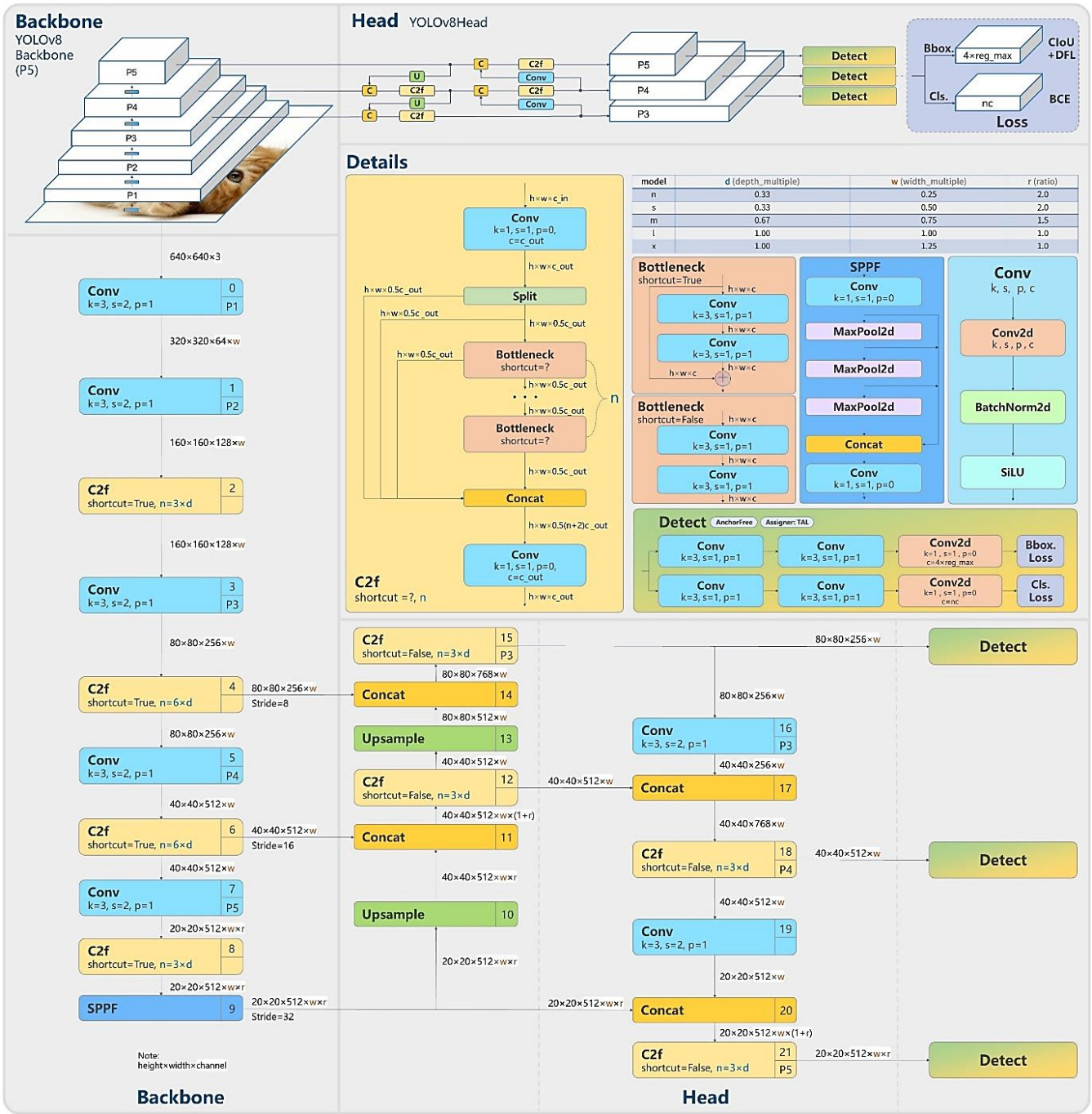

Figure 3.21 YOLOv8 Architecture and Visualization

YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box. Anchor boxes were a notoriously tricky part of earlier YOLO models, since they may represent the distribution of the target benchmark's boxes but not the distribution of the custom dataset. Anchor free detection reduces the number of box predictions, which speeds up Non-Maximum Suppression (NMS), a complicated post processing step that sifts through candidate detections after inference.

Deep learning research tends to focus on model architecture, but the training routine in YOLOv5 and YOLOv8 is an essential part of their success. YOLOv8 augments images during training online. At each epoch, the model sees a slightly different variation of the images it has been provided.

One of those augmentations is called mosaic augmentation. This involves stitching four images together, forcing the model to learn objects in new locations, in partial occlusion, and against different surrounding pixels.

However, this augmentation is empirically shown to degrade performance if performed through the whole training routine. It is advantageous to turn it off for the last ten training epochs. This sort of change is exemplary of the careful attention YOLO modeling has been given in overtime in the YOLOv5 repo and in the YOLOv8 research.

YOLOv8 COCO Accuracy [28]

COCO (Common Objects in Context) is the industry standard benchmark for evaluating object detection models. When comparing models on COCO, we look at the mAP value and FPS measurement for inference speed. Models should be compared at similar inference speeds.

The image below shows the accuracy of YOLOv8 on COCO, using data collected by the Ultralytics team.

▼ Detection

Model	size (pixels)	mAP^{val} 50-95	Speed CPU (ms)	Speed T4 GPU (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	-	-	3.2	8.7
YOLOv8s	640	44.9	-	-	11.2	28.6
YOLOv8m	640	50.2	-	-	25.9	78.9
YOLOv8l	640	52.9	-	-	43.7	165.2
YOLOv8x	640	53.9	-	-	68.2	257.8

- **mAP^{val}** values are for single-model single-scale on COCO val2017 dataset.
  Reproduce by `yolo mode=val task=detect data=coco.yaml device=0`
- **Speed** averaged over COCO val images using an Amazon EC2 P4d instance.
  Reproduce by `yolo mode=val task=detect data=coco128.yaml batch=1 device=0/cpu`

Figure 3.22 YOLOv8 COCO Evaluation

YOLOv8 COCO accuracy is state of the art for models at comparable inference latencies as of now.

3.3.2 OpenCV

OpenCV was started at Intel in 1999 by **Gary Bradsky**, and the first release came out in 2000. **Vadim Pisarevsky** joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day. [29]

OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. Interfaces for high-speed GPU operations based on CUDA and OpenCL are also under active development.

OpenCV-Python is the Python API for OpenCV, combining the best qualities of the OpenCV C++ API and the Python language.

OpenCV has a modular structure, which means that the package includes several shared or static libraries. The following modules are available: [30]

- **Core functionality (core)** - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.
- **Image Processing (imgproc)** - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.
- **Video Analysis (video)** - a video analysis module that includes motion estimation, background subtraction, and object tracking algorithms.
- **Camera Calibration and 3D Reconstruction (calib3d)** - basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence algorithms, and elements of 3D reconstruction.
- **2D Features Framework (features2d)** - salient feature detectors, descriptors, and descriptor matchers.

- **Object Detection (objdetect)** - detection of objects and instances of the predefined classes (for example, faces, eyes, mugs, people, cars, and so on).
- **High-level GUI (highgui)** - an easy-to-use interface to simple UI capabilities.
- **Video I/O (videoio)** - an easy-to-use interface to video capturing and video codecs.
- Some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others.

3.3.2.1 Machine Learning with OpenCV [31]

a) Training Data

In machine learning algorithms there is notion of training data. Training data includes several components:

- A set of training samples. Each training sample is a vector of values (in Computer Vision it's sometimes referred to as feature vector). Usually all the vectors have the same number of components (features); OpenCV ml module assumes that. Each feature can be ordered (i.e. its values are floating-point numbers that can be compared with each other and strictly ordered, i.e. sorted) or categorical (i.e. its value belongs to a fixed set of values that can be integers, strings etc.).
- Optional set of responses corresponding to the samples. Training data with no responses is used in unsupervised learning algorithms that learn structure of the supplied data based on distances between different samples. Training data with responses is used in supervised learning algorithms, which learn the function mapping samples to responses. Usually the responses are scalar values, ordered (when we deal with regression problem) or categorical (when we deal with classification problem; in this case the responses are often called "labels"). Some algorithms, most noticeably Neural networks, can handle not only scalar, but also multi-dimensional or vector responses.
- Another optional component is the mask of missing measurements. Most algorithms require all the components in all the training samples be valid, but some other algorithms, such as decision tress, can handle the cases of missing measurements.

- In the case of classification problem user may want to give different weights to different classes. This is useful, for example, when:
  - User wants to shift prediction accuracy towards lower false-alarm rate or higher hit-rate.
  - User wants to compensate for significantly different amounts of training samples from different classes.
- In addition to that, each training sample may be given a weight, if user wants the algorithm to pay special attention to certain training samples and adjust the training model accordingly.
- Also, user may wish not to use the whole training data at once, but rather use parts of it, e.g. to do parameter optimization via cross-validation procedure.

b) Classifiers in OpenCV

i. Normal Bayes Classifier

This simple classification model assumes that feature vectors from each class are normally distributed (though, not necessarily independently distributed). So, the whole data distribution function is assumed to be a Gaussian mixture, one component per class. Using the training data the algorithm estimates mean vectors and covariance matrices for every class, and then it uses them for prediction.

ii. K-Nearest Neighbors

The algorithm caches all training samples and predicts the response for a new sample by analyzing a certain number (**K**) of the nearest neighbors of the sample using voting, calculating weighted sum, and so on. The method is sometimes referred to as "learning by example" because for prediction it looks for the feature vector with a known response that is closest to the given vector.

iii. Support Vector Machines

Originally, support vector machines (SVM) was a technique for building an optimal binary (2-class) classifier. Later the technique was extended to regression and clustering problems. SVM is a partial case of kernel-based methods. It maps feature vectors into a higher-dimensional space using a kernel function and builds an optimal linear discriminating function in this

space or an optimal hyper- plane that fits into the training data. In case of SVM, the kernel is not defined explicitly. Instead, a distance between any 2 points in the hyper-space needs to be defined.
The solution is optimal, which means that the margin between the separating hyper-plane and the nearest feature vectors from both classes (in case of 2-class classifier) is maximal. The feature vectors that are the closest to the hyper-plane are called *support vectors*, which means that the position of other vectors does not affect the hyper-plane (the decision function). [32]

iv. Decision Trees

A decision tree is a binary tree (tree where each non-leaf node has two child nodes). It can be used either for classification or for regression. For classification, each tree leaf is marked with a class label; multiple leaves may have the same label.
For regression, a constant is also assigned to each tree leaf, so the approximation function is piecewise constant.

3.3.2.2 Object Detection

Object Detection is a computer technology related to computer vision, image processing, and deep learning that deals with detecting instances of objects in images and videos. We will do object detection in this article using something known as haar cascades.

Haar Cascades

Haar Cascade classifiers are an effective way for object detection. This method was proposed by Paul Viola and Michael Jones in their paper Rapid Object Detection using a Boosted Cascade of Simple Features. Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier.

- **Positive images** – These images contain the images which we want our classifier to identify.
- **Negative Images** – Images of everything else, which do not contain the object we want to detect.

Steps to download the requirements below:

- Run the following command in the terminal to install opencv.

  pip install opencv-python

- Run the following command to in the terminal install the matplotlib.

  pip install matplotlib

Figure 3.23 Image Used for Object Detection

Recognition

We will use the detectMultiScale() function of OpenCV to recognize big signs as well as small ones:

```
import cv2
from matplotlib import pyplot as plt

# Opening image
img = cv2.imread("image.jpg")

# OpenCV opens images as BRG
# but we want it as RGB We'll
# also need a grayscale version
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


# Use minSize because for not
# bothering with extra-small
# dots that would look like STOP signs
stop_data = cv2.CascadeClassifier('stop_data.xml')

found = stop_data.detectMultiScale(img_gray,
                                   minSize =(20, 20))
```

```
# Don't do anything if there's
# no sign
amount_found = len(found)

if amount_found != 0:

    # There may be more than one
    # sign in the image
    for (x, y, width, height) in found:

        # We draw a green rectangle around
        # every recognized sign
        cv2.rectangle(img_rgb, (x, y),
                      (x + height, y + width),
                      (0, 255, 0), 5)

# Creates the environment of
# the picture and shows it
plt.subplot(1, 1, 1)
plt.imshow(img_rgb)
plt.show()
```

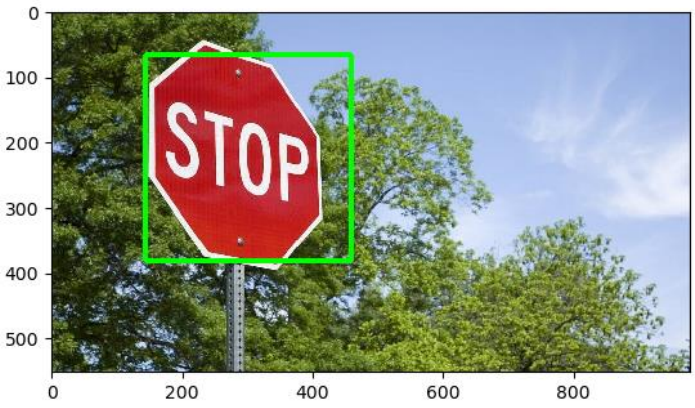

Figure 3.24 Output Image

3.3.2.3 OpenCV People Counter with Python [33]

Understanding object detection vs. object tracking

There is a fundamental difference between object detection and object tracking that you must understand before we proceed.

When we apply object detection we are determining where in an image/frame an object is. An object detector is also typically more computationally expensive, and therefore slower, than an object tracking algorithm. Examples of object detection algorithms include Haar cascades, HOG + Linear SVM, and deep learning-based object detectors such as Faster R-CNNs, YOLO, and Single Shot Detectors (SSDs).

An object tracker, on the other hand, will accept the input (x, y)-coordinates of where an object is in an image and will:

1) Assign a unique ID to that particular object
2) Track the object as it moves around a video stream, predicting the new object location in the next frame based on various attributes of the frame (gradient, optical flow, etc.)

Examples of object tracking algorithms include MedianFlow, MOSSE, GOTURN, kernalized correlation filters, and discriminative correlation filters, to name a few.

Combining both object detection and object tracking

Highly accurate object trackers will combine the concept of object detection and object tracking into a single algorithm, typically divided into two phases:

- **Phase 1 — Detecting:** During the detection phase we are running our computationally more expensive object tracker to (1) detect if new objects have entered our view, and (2) see if we can find objects that were "lost" during the tracking phase. For each detected object we create or update an object tracker with the new bounding box coordinates. Since our object detector is more computationally expensive we only run this phase once every N frames.

- **Phase 2 — Tracking:** When we are not in the "detecting" phase we are in the "tracking" phase. For each of our detected objects, we create an object

tracker to track the object as it moves around the frame. Our object tracker should be faster and more efficient than the object detector. We'll continue tracking until we've reached the N-th frame and then re-run our object detector. The entire process then repeats.

The benefit of this hybrid approach is that we can apply highly accurate object detection methods without as much of the computational burden. We will be implementing such a tracking system to build our people counter.

Combining object tracking algorithms

To implement our people counter we'll be using both OpenCV and dlib. We'll use OpenCV for standard computer vision/image processing functions, along with the deep learning object detector for people counting.

We'll then use dlib for its implementation of correlation filters. We could use OpenCV here as well; however, the dlib object tracking implementation was a bit easier to work with for this project.

Step 1:

We accept a set of bounding boxes and compute their corresponding centroids (i.e., the center of the bounding boxes):

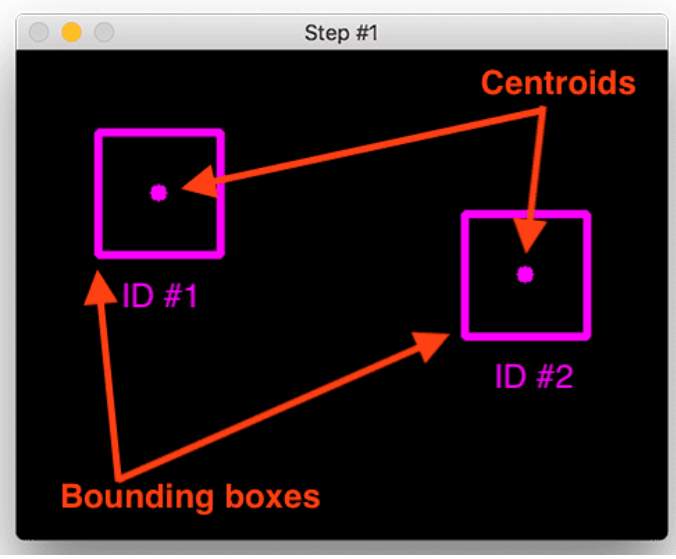

Figure 3.25: To build a simple object tracking via centroids script with Python, the first step is to accept bounding box coordinates and use them to compute centroids.

The bounding boxes themselves can be provided by either:

- An object detector (such as HOG + Linear SVM, Faster R- CNN, SSDs, etc.)
- Or an object tracker (such as correlation filters)

In the above image you can see that we have two objects to track in this initial iteration of the algorithm.

Step 2:

We compute the Euclidean distance between any new centroids (yellow) and existing centroids (purple):

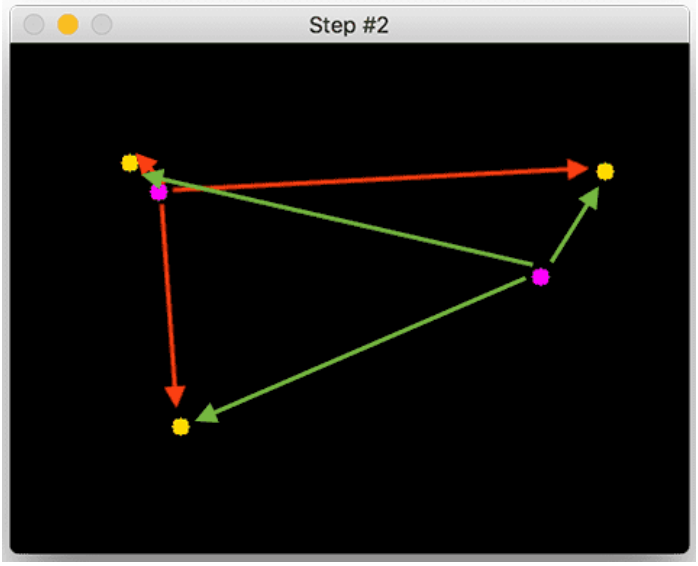

Figure 3.26: Three objects are present in this image. We need to compute the Euclidean distance between each pair of original centroids (red) and new centroids (green)

The centroid tracking algorithm makes the assumption that pairs of centroids with minimum Euclidean distance between them must be the same object ID. In the example image above we have two existing centroids (purple) and three new centroids (yellow), implying that a new object has been detected (since there is one more new centroid vs. old centroid).

The arrows then represent computing the Euclidean distances between all purple centroids and all yellow centroids.

Step 3:

Once we have the Euclidean distances we attempt to associate object IDs:

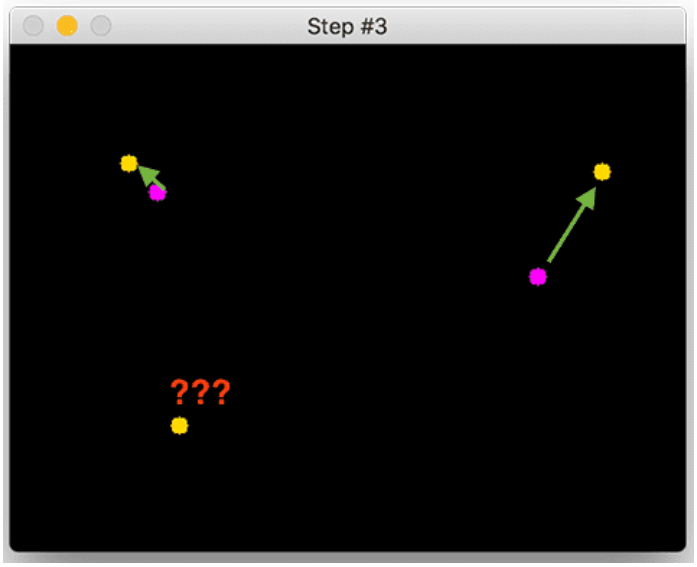

Figure 3.27: Our simple centroid object tracking method has associated objects with minimized object distances

In Figure 3 you can see that our centroid tracker has chosen to associate centroids that minimize their respective Euclidean distances.

But what about the point in the bottom-left? It didn't get associated with anything — what do we do?

To answer that question we need to perform Step 4

Step 4:

Registering new objects

Registering simply means that we are adding the new object to our list of tracked objects by:

- Assigning it a new object ID
- Storing the centroid of the bounding box coordinates for the new object

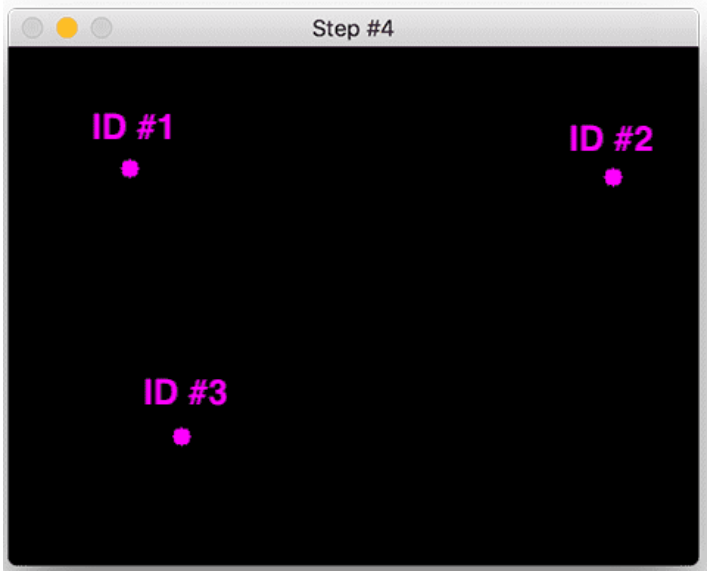

Figure 3.28: In our object tracking example, we have a new object that wasn't matched with an existing object, so it is registered as object ID #3.

In the event that an object has been lost or has left the field of view, we can simply deregister the object (Step 5).

Exactly how you handle when an object is "lost" or is "no longer visible" really depends on your exact application, but for our people counter, we will deregister people IDs when they cannot be matched to any existing person objects for 40 consecutive frames.

Again, this is only a brief overview of the centroid tracking algorithm.

3.4 Wake Word Model

A wake-word model is a tiny algorithm that monitors a stream of audio for a special wake-word and activates your voice assistant upon detecting it. [34]

A wake-word model must run on the edge (not cloud) for multiple reasons:

1) Privacy.
2) Cost: It is simply impractical to have a stream of audio from every toed device to the cloud 24/7.
3) Power efficiency: In order to always run it must be extremely power efficient on mobile/wearable devices.

Dataset:

To create a wake word model with custom wake word, we need to create the dataset ourselves. The wake word model needs a data set consisting of at least 100 records of the wake word we want, and it also needs to record the background sound or the surrounding environment.

In the model we built, we created a dataset of 2,000 records of the word "up" and also more than 1,000 records of the background.

Feature Extraction Technique:

Mel-frequency cepstral coefficients (MFCC).

Model 3 Architecture

We used the same architecture as the speech-to-text Model 3 but with making output layer equal 2 instead of 10.

Training and Compilation Parameters:

- **Epochs:** 30
- **Loss function:** Sparse Categorical Cross entropy.
- **Optimizer:** Adam optimizer.
- **Metrics:** Accuracy.
- **Learning rate:** 0.0001
- **Activation for convolution layers:** ReLU.
- **Activation for output layer:** Softmax.

Training Curve:

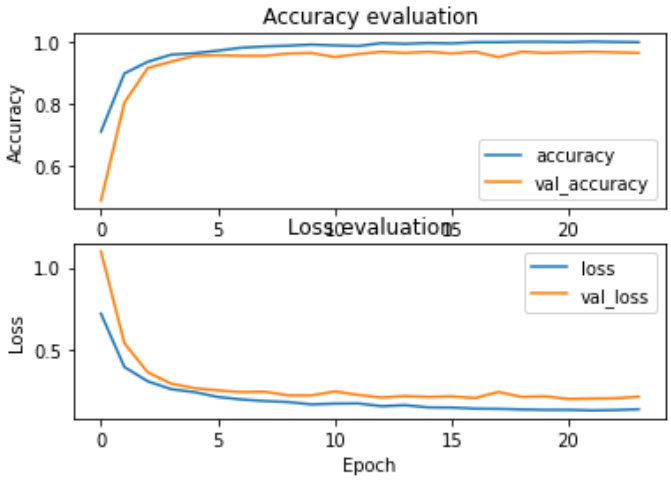

Figure 3.29 Training Curve of the wake word model

Evaluation:

- **Test accuracy:** 96.87%
- **Test loss:** 0.2245
- **Total model size:** 1.65 MB

Chapter 4. Hardware Implementation

4.1 Hardware Components

- Raspberry Pi4 Model B
- Relay Module 8 Channels
- Raspberry Pi Camera Module
- Mic
- Speaker

Figure 4.30 Raspberry Pi4 Model B

Figure 4.31 Relay Module 8 Channels

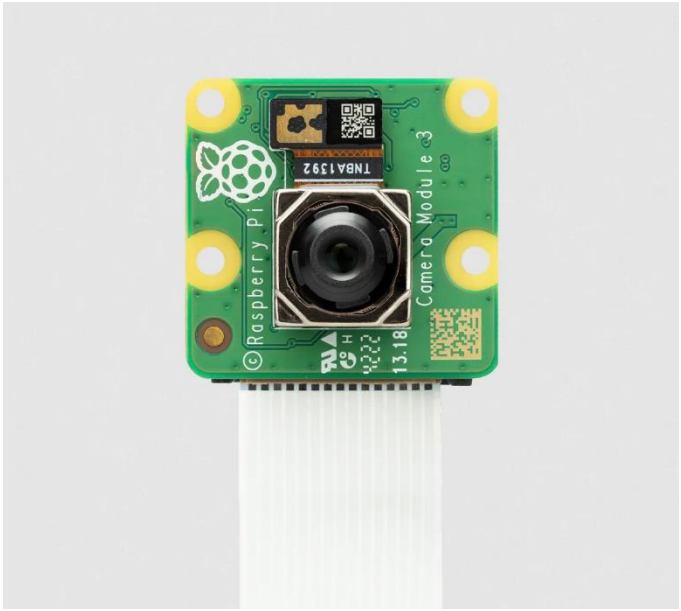

Figure 4.32 Raspberry Pi Camera Module

4.2 Raspberry Pi Environment

The Raspberry Pi environment serves as a versatile and affordable platform for various applications. It is a single-board computer that provides ample processing power and connectivity options. The Raspberry Pi ecosystem offers a range of models, each with unique specifications and capabilities. In the context of your "Elevator" project, we have chosen the Raspberry Pi4 as the hardware foundation.

The Raspberry Pi 4 is a powerful board that features a quad-core ARM Cortex-A72 CPU, clocked at 1.5GHz, providing improved processing capabilities compared to previous models. It supports up to 8GB of LPDDR4 RAM, allowing for efficient multitasking and handling of complex tasks. The board provides numerous connectivity options, including USB ports, HDMI output, Ethernet, and wireless capabilities like Wi-Fi and Bluetooth. These features make the Raspberry Pi 4 an ideal choice for your project, enabling seamless integration with the camera, microphone, and speaker components required for your "Elevator" system.

4.3 Hardware for Person Detection

To enable person detection within the "Elevator" system, we have incorporated a camera module. The chosen 5-megapixel camera provides high-resolution image capture, allowing for accurate identification and tracking of individuals. By utilizing computer vision algorithms, such as Open CV, the Raspberry Pi can analyze the camera feed in real-time, detecting and recognizing persons within its field of view. This functionality forms the basis of the system's ability to identify users and interact with them effectively.

4.4 Hardware for User Interaction

User interaction is a key aspect of the "Elevator" system. To facilitate this, we have integrated a microphone and speaker with the Raspberry Pi. The microphone allows users to provide voice commands and communicate with the system. Through speech recognition algorithms, including wake-up word detection and natural language processing, the Raspberry Pi can accurately capture and interpret user input. The speaker component plays a crucial role in providing informative responses and system feedback, ensuring clear communication with the user. By Using Speech to Text models, the Raspberry Pi can generate clear and intelligible synthesized speech output. It asks for confirmation before using the elevator, provides instructions for

elevator usage, informs the user when their destination floor has been successfully registered and delivers other relevant updates and instructions. The speaker's audio quality and amplification ensure that the system's feedback is easily understandable and audible to the user.

4.5 Required Circuit for Interacting with a Real Elevator

In order to interface with a real elevator and simulate the functionality of the elevator keypad, a specific circuit is needed. The elevator keypad typically consists of several buttons, and each button is associated with a specific floor. In this case, let's assume the keypad has 8 buttons, corresponding to 8 different floors.

The elevator control circuit is designed to receive signals from the keypad and process them accordingly. The keypad is connected to the control circuit through a series of output wires. By default, when none of the buttons on the keypad are pressed, these output wires do not conduct any voltage to the elevator control circuit.

When a button on the keypad is pressed, it completes the circuit between the corresponding output wire and a common wire. This allows the common voltage to be conducted through the selected wire to the elevator control circuit. Each pin or wire in the control circuit is dedicated to a specific floor, such as wire 0 being associated with floor 0.

To simulate the functionality of the elevator keypad, an electronic switch, specifically a relay module with 8 channels, is utilized. Since we assumed the keypad has 8 buttons, each corresponding to a specific floor, the relay module with 8 channels is a suitable choice. The common voltage is connected to the common terminal of the relay module, and the 8 output wires from the elevator keypad are connected to the respective output terminals of the relay module.

In addition, the relay module is connected to the Raspberry Pi 4. The Pi's 8 pins are used as inputs for the relay module, with each pin corresponding to a specific floor. By default, when there are no voice commands being processed, the relay module keeps the normally open contacts open, which means there is no voltage output.

When a user speaks the wake-up word and issues a voice command indicating a specific floor (ranging from 0 to 7), the Raspberry Pi 4 processes the voice command accordingly. Based on the floor number received, the Pi outputs a signal to the corresponding pin in the relay module. This causes the relay module to close the

normally open contacts associated with the floor wire, allowing the common voltage to be conducted through that wire for a certain duration.

The other wires in the relay module remain in their default state, which is "floating" or not conducting any voltage. This way, the functionality of the elevator keypad is effectively simulated through the coordination between the Raspberry Pi, relay module, and elevator control circuit.

By implementing this circuit, you can accurately replicate the behavior of the elevator keypad and control the elevator's movements based on the user's voice commands and floor selections.

4.6 System Design

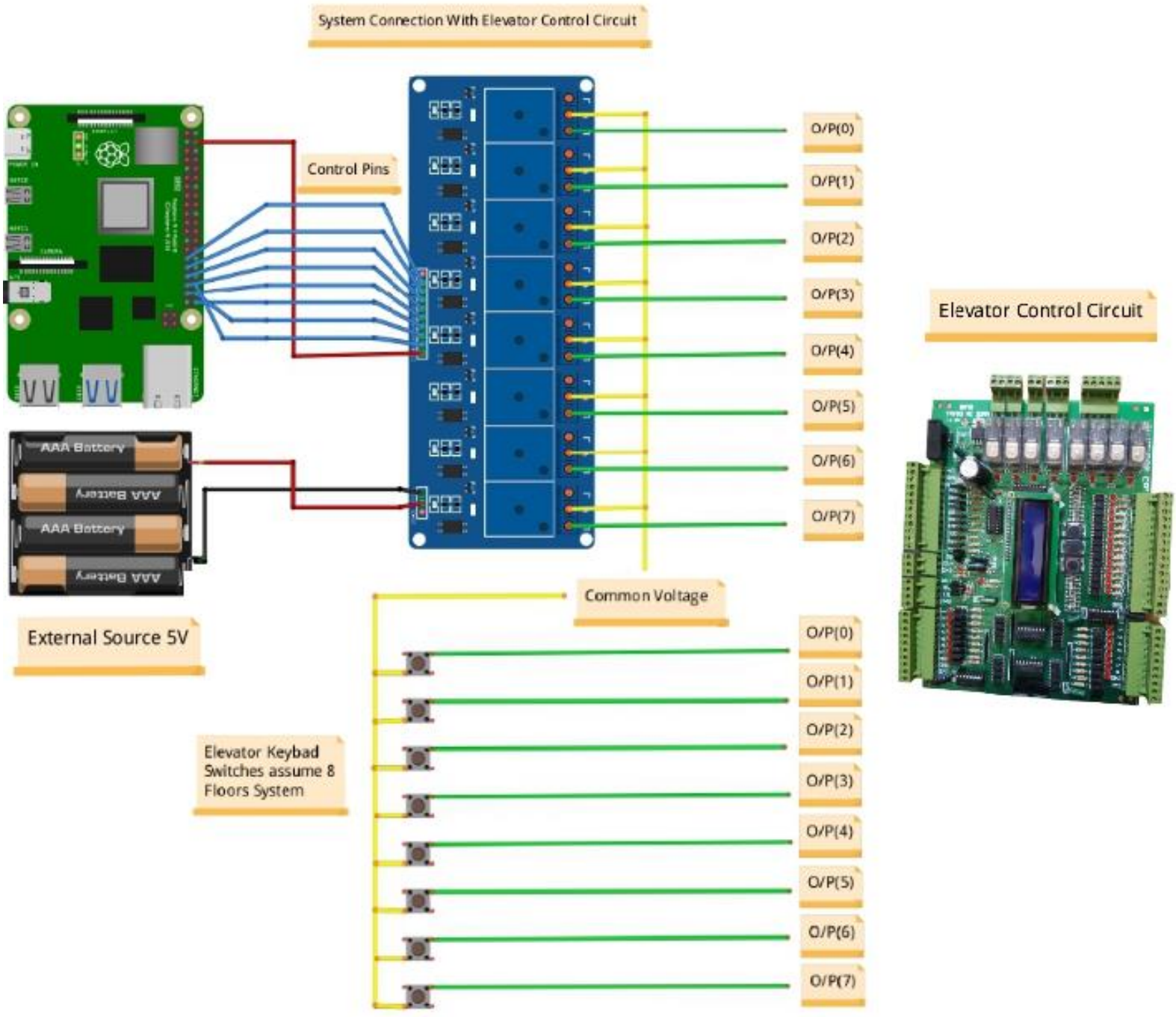

Figure 4.33 Inside System Connection with Elevator Control Circuit

4.7 Hardware Challenges

1) We encountered compatibility issues with some DL frameworks and libraries that require more advanced hardware features.

Example of some libraries we could not install:

- **Librosa**
  Which we needed to install for speech recognition model
- **Pytorch**
  Which we needed to install YOLOv8

So, we changed the YOLOv8 to OpenCV. OpenCV requires the installation of TensorFlow.

2) The mic which we used does not work directly with Raspberry Pi. So, we used the external USB sound card to activate it

3) Low computational power: The Raspberry Pi 4 has a quad-core ARM Cortex-A72 CPU and a Video Core VI GPU, which are not designed for high-performance DL tasks

4) Power consumption and heat dissipation: The Raspberry Pi 4 consumes more power and generates more heat than its predecessors, especially when running intensive DL tasks, so we used a proper power supply and a cooling system (a fan) to prevent overheating or damage.

4.8 AI Models that fit the Hardware

4.8.1 Text-to-Speech Models

In text-to-speech model, we have tested five different models on the Raspberry Pi4 and they are Python Text-to-Speech Version 3 (Pyttsx3), eSpeak Text-to-Speech (eSpeak TTS), Festival Text-to-Speech (Festival TTS), Pico Text-to-Speech (Pico TTS) and Google Text-to-Speech (gTTS).

From this testing we found out that:

1) The best model is gTTS, as the sound produced from it resembles the sound of a real person, and there is also no cracking in the sound produced, but the big problem for us is that the model works in the presence of internet connection.
2) The best offline model is Pico TTS.
3) eSpeak TTS and Festival TTS models were much better than pyttsx3 and slightly worse than pico.
4) The worst model is pyttsx3, as the output audio is choppy and the model can't convert the entire text into audio.

So we used the Pico Text-to-Speech (Pico TTS) model on our Raspberry Pi.

4.8.2 Speech-to-Text Models

Although we made more than one model using CNN, LSTM, Transformers and we also tested more than one model like the DeepSpeech model, the best model we have tried is the VOSK. The VOSK model on the Raspberry Pi runs fast, converts speech to text with high accuracy, and provides good performance in the presence of noise.

So, we used the VOSK model on our Raspberry Pi.

4.8.3 Object Detection Models

In object detection, we used YOLOv8 Model and OpenCV models. Yolov8 model is faster and more accurate than the object detection model using OpenCV, and from our experience with the two models we noticed that the object detection model using OpenCV, predicts a greater number of people than the existing ones and this happens when a person is near from camera or during the movement of the person.

Although the YOLOv8 is better than the object detection model using OpenCV, we used the object detection model using OpenCV because of the difficulty in installing some libraries for using YOLOv8, and because of we do not care about the number of predicted people, but we only care about predicting people.

4.8.4 Wake Word Model

Although we made a wake word model using CNN, but we encountered two problems in trying this model, which is our inability to install librosa library, and also sometimes it predicts the background sound as a wake word when there is a loud noise.

So, we used the VOSK model on our Raspberry Pi as a wake word model.

Chapter 5. Future Work

In this chapter, we discuss the future work and potential directions for enhancing the capabilities of our Smart Elevator. With the aim of creating an inclusive and accessible elevator environment, we are committed to covering the needs of diverse user categories. While our current system addresses the requirements of many users, there are specific areas that require further exploration and development. Due to hardware limitations, the future work outlined in this chapter focuses on two key aspects: the integration of a Sign Language model and the coordination of blind individuals with precision. By delving into these areas, we strive to extend the benefits of our Smart Elevator to a broader range of individuals, ensuring that their needs are met effectively.

5.1 Blind People's Precise Coordination

One of the key areas of future work is to enhance the elevator experience for blind individuals by providing them with reliable coordination and guidance. Currently, blind users can use the wake-up word followed by their desired floor number to initiate elevator movement. However, we envision a system that goes beyond floor selection and actively assists blind users in positioning themselves correctly within the elevator cabin.

To achieve this, we propose the implementation of AI-based algorithms that utilize computer vision and depth sensing technologies. By leveraging these technologies, the system can provide real-time guidance and instructions to blind users, enabling them to position themselves accurately in front of the elevator door. For example, the system can detect the user's position inside the elevator cabin and instruct them to move a certain number of steps to the right or left to align with the door.

To enable this precise coordination, the system would use a combination of audio cues and tactile feedback mechanisms. Through text-to-speech (TTS) capabilities, the system can provide clear and concise instructions to the user, guiding them towards the desired position. Additionally, tactile feedback mechanisms, such as haptic vibration or textured surfaces, can be incorporated to provide physical cues that assist blind users in aligning themselves correctly.

The implementation of such AI-driven coordination and guidance systems would greatly enhance the accessibility and usability of the elevator for blind individuals.

By providing step-by-step instructions and real-time feedback, the system aims to empower blind users to navigate the elevator environment with confidence and ease.

5.2 Sign Language Model

One of the primary challenges faced by individuals with hearing impairments is the difficulty of communication in environments where spoken language dominates. To address this, a potential future enhancement to our Smart Elevator involves integrating a Sign Language model. By leveraging computer vision and machine learning techniques, the system can be trained to recognize and interpret sign language gestures used by individuals with hearing impairments. This integration would enable seamless communication between users who rely on sign language and the Smart Elevator, providing them with an inclusive and accessible experience.

The Sign Language model would incorporate advanced deep learning algorithms to accurately detect and interpret hand movements, gestures, and facial expressions commonly used in sign language. It would also utilize natural language processing techniques to convert the recognized signs into text or spoken language, ensuring effective communication between the user and the system. This feature would empower individuals with hearing impairments to interact with the elevator system confidently and independently, further enhancing their overall accessibility experience.

5.3 Summary

In this chapter, we have discussed the future work and potential advancements for our Smart Elevator. By integrating a Sign Language model, we aim to facilitate effective communication for individuals with hearing impairments who rely on sign language. Additionally, by enhancing blind people's precise coordination within the elevator cabin through tactile feedback mechanisms and audio-based spatial guidance, we strive to improve their overall navigation and accessibility experience. These developments are driven by our commitment to serving the needs of diverse user categories, ensuring that our Smart Elevator remains inclusive, supportive, and accessible to all.

Through these future enhancements, we envision a future where our Smart Elevator caters to a wide range of users, including those with hearing impairments and visual impairments. By providing effective communication for individuals using sign language and improving navigation for blind users, our system will further enhance the accessibility and inclusivity of elevator experiences. These advancements exemplify our dedication to leveraging AI and innovative technologies to overcome traditional limitations and create smart solutions that benefit people in various scenarios, including challenging situations like the COVID-19 pandemic.

6. References

[1] Kloeckl, K., & Schmidinger, C. Intelligent Elevators: A Framework for Artificial Intelligence Integration in Vertical Transportation Systems. Proceedings of the International Conference on Computing, Networking and Communications (ICNC), Maui, HI, USA. (2018)

[2] Zhao, W., Liu, Y., Zhang, Q., & Liu, M. A Smart Elevator Control System Based on Computer Vision. IEEE Access, 8, 69045-69055. (2020).

[3] Gupta, R., Choudhary, S., Gupta, M., & Sahu, R. Accessibility in Elevator Systems for Visually Impaired: A Review. Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Kolkata, India. (2021).

[4] Sousa, J., & Nunes, I. A Review of Indoor Navigation Systems for Blind People. International Journal of Online Engineering, 13(1), 60-64. (2017).

[5] https://gtts.readthedocs.io/en/latest/

[6] https://espeak.sourceforge.io/

[7] http://www.cstr.ed.ac.uk/projects/festival/

[8] https://github.com/Kaljurand/PicoTTS

[9] https://github.com/nateshmbhat/pyttsx3#documentation

[10] Tensor Flow, speech commands
[https://www.tensorflow.org/datasets/catalog/speech_commands]

[11] Min Xu; et al. "HMM-based audio keyword generation" (PDF). In Kiyoharu Aizawa; Yuichi Nakamura; Shin'ichi Satoh (eds.). Advances in Multimedia Information Processing – PCM 2004: 5th Pacific Rim Conference on Multimedia. Springer. (2004).

[12] Sahidullah, Md.; Saha, Goutam "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition". Speech Communication. 54 (4): 543–565. (May 2012).

[13] Kaggle, Arabic Speech Commands Dataset
[https://www.kaggle.com/datasets/murtadhayaseen/arabic-speech-commands-dataset]

[14] A Gentle Introduction to Long Short-Term Memory Networks by the Experts [https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/]

[15] Yoshua Bengio, et al., Learning Long-Term Dependencies with Gradient Descent is Difficult, 1994

[16] Hassim Sak, et al., Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling, 2014

[17] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation

[18] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading

[19] Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pages 2440–2448. Curran Associates, Inc., 2015

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez and Łukasz Kaiser. Attention Is All You Need

[21] http://mohitmayank.com/a_lazy_data_science_guide/audio_intelligence/wav2vec2/

[22]https://pypi.org/project/vosk/#:~:text=Vosk%20supplies%20speech%20recognition%20for,Android%20smartphone%20to%20big%20clusters

[23] https://alphacephei.com/vosk/models

[24] Deep Speech: Scaling up end-to-end speech recognition. Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. Ng. arXiv:1412.5567

[25]Lohia, Aditya; Kadam, Kalyani Dhananjay; Joshi, Rahul Raghvendra; and Bongale, Dr. Anupkumar M., "Bibliometric Analysis of One-stage and Two-stage Object Detection" (2021).

[26]R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” 580–587, June 2014

[27]Brief summary of YOLOv8 model structure · Issue #189 · ultralytics/ultralytics · GitHub

[28]What is YOLOv8? The Ultimate Guide. https://blog.roboflow.com/whats-new-in-yolov8/

[29]Introduction to OpenCV-Python Tutorials. https://docs.opencv.org/3.4/d0/de3/tutorial_py_intro.html

[30]Introduction to OpenCV.

https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/index.html#:~:text=OpenCV%20(Open%20Source%20Computer%20Vision,x%20API

[31]Machine Learning Overview.

https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/page_ml_intro.html

[32]Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3):27, 2011.

[33]OpenCV People Counter. https://pyimagesearch.com/2018/08/13/opencv-people-counter/

[34] https://medium.com/@alirezakenarsarianhari/yet-another-wake-word-detection-engine-a2486d36d8d4