Ryad BENAISSA

Phelma SEOC
M1 / 2A

Hanoi University of Science and Technology
(HUST)
No. 1 Dai Co Viet, Hai Ba Trung, Hanoi, Vietnam

Keyword Spotting System for Consumer Equipment Control

from 13/05/2024 to 06/09/2024

Under the supervision of:

- Company supervisor: Cuong, NGUYEN QUOC, cuong.nguyenquoc@hust.edu.vn
- Phelma Tutor: Michele, PORTOLAN, michele.portolan@univ-grenoble-alpes.fr

Confidentiality: no

Ecole nationale supérieure de physique, électronique, matériaux
Phelma Bât. Grenoble INP - Minatec 3 Parvis Louis Néel
CS 50257 F-38016 Grenoble Cedex 01
Tél +33 (0)4 56 52 91 00 Fax +33 (0)4 56 52 91 03
http://phelma.grenoble-inp.fr

Contents

List of Figures

Glossary

CNN Convolutional Neural Network, a type of deep learning model particularly effective for pattern recognition in images and audio spectrograms.. 2, 12–14, 23

DCT Discrete Cosine Transform, a technique used to convert a signal into elementary frequency components. 11

KWS Keyword Spotting, technology used to detect specific keywords in audio streams, commonly used in voice-activated systems.. 8, 12

Librosa A Python package for music and audio analysis. 23

MFCC Mel Frequency Cepstral Coefficient, a feature extraction method in speech processing that models how humans perceive sound.. 8, 11, 12, 23

PyTorch A Python-based machine learning library used for building and training neural networks, widely used in deep learning.. 23

Raspberry Pi A low-cost, credit-card sized computer that plugs into a computer monitor or TV and uses a standard keyboard and mouse. 23

ReLU Rectified Linear Unit, a widely used activation function in neural networks. 14

SNR Signal-to-Noise Ratio, a measure used in science and engineering to quantify how much a signal has been corrupted by noise. 18

STFT Short-Time Fourier Transform, a method to analyze the frequency content of local sections of a signal as it changes over time. 9

Acknowledgements

I am deeply thankful to the teams at both the Machine Learning and Internet of Things (MLIOT) Lab and the SmartSensor Lab for their incredible support and warm hospitality during my internship. My time spent at these labs has been instrumental in shaping my understanding of Machine Learning and has greatly enriched my appreciation of Vietnamese culture.

My internship offered a unique chance to delve into the complexities of Machine Learning, an area that was relatively new to me. The guidance and mentorship from the dedicated staff at MLIOT and SmartSensor Lab were invaluable, providing me with the tools and knowledge to navigate the challenges of this field. The collaborative spirit in these labs enabled me to build not only technical expertise but also confidence as an apprentice researcher.

In addition to the professional development I experienced, I am particularly grateful for the cultural exposure that accompanied my stay in Vietnam. The efforts made by both teams to immerse me in local customs, cuisine, and traditions made my experience in the country both enlightening and memorable. Their kindness in sharing the rich Vietnamese culture with me has left a lasting impact.

I would like to extend my thanks to Hoang Ngoc Chau for his support and advices throughout my internship this improved my learning experience a lot.

I am also very grateful to my internship supervisor, Professor Nguyen Quoc Cuong for choosing me and believing in my potential for this project. His thoughtful guidance and encouragement throughout this project were crucial to my development in the field of Machine Learning. Working under his supervision has been a significant milestone in my academic and professional journey.

As I conclude my time at MLIOT and SmartSensor Lab, I do so with immense gratitude for the knowledge and cultural experiences I have gained. This internship has been a journey of significant learning and personal growth, and I am sincerely thankful to everyone who made it such a remarkable and impactful experience.

Introduction

In today's rapidly evolving technological landscape, the interaction between humans and machines is increasingly becoming seamless and intuitive. Among the various interfaces available, voice-activated systems have gained considerable traction due to their natural and user-friendly nature. These systems enable users to control a wide array of devices through simple voice commands, thus enhancing the overall user experience. Keyword Spotting (KWS) technology, which forms the backbone of such voice-activated systems, has become an essential component in consumer electronics, facilitating hands-free control of devices.

The importance of KWS lies in its ability to accurately detect specific keywords, which are then mapped to predefined actions or commands, making it an indispensable tool in the domain of consumer electronics.
The objective of this project, titled *"Keyword Spotting System for Consumer Equipment Control,"* is to design and implement a highly efficient KWS system that can be seamlessly integrated into various consumer electronic devices. The primary goal is to develop a system that not only exhibits high accuracy in detecting specific keywords but also operates with minimal latency on embedded platforms.

This is particularly important for ensuring that the system remains responsive and user-friendly, even when deployed on devices with limited computational resources.
To achieve these objectives, this project will leverage advanced deep learning techniques, such as Convolutional Neural Networks (CNNs). These models have proven to be highly effective in handling complex audio signals, making them well-suited for the task of keyword spotting.

Additionally, the project will explore robust acoustic feature extraction methods, including Mel-Frequency Cepstral Coefficients (MFCCs) and spectrograms. These methods are crucial for capturing the unique characteristics of the spoken keywords, which in turn enhances the system's ability to accurately recognize them.
Beyond the implementation of deep learning models, the project will incorporate advanced feature engineering techniques, such as data augmentation and signal processing. Data augmentation, for instance, will help the system generalize better by artificially expanding the training dataset with variations of the original audio samples.

A key aspect of this project is the optimization of the deep learning models and the overall speech processing pipeline. By fine-tuning the model architecture, hyperparameters, and training procedures, the project aims to maximize the accuracy of keyword detection while minimizing the computational load.

This is particularly important for ensuring that the system can be deployed on embedded platforms without compromising performance.
The final objective of the project is to design a user-friendly voice command mapping system. This system will allow users to control various features and functions of a device through intuitive voice commands.

The implementation of this system on an embedded platform will involve careful consideration of both software and hardware aspects, ensuring that the final product is not only functional but also practical for everyday use.

As a result, this report will be structured into several main sections. The first section will provide an overview of the entire system, detailing its various components and principles. The subsequent sections will delve into specific aspects, including the dataset used and its augmentation techniques, the architecture of the neural network model, and the implementation of the system in both use case scenarios and on an embedded platform like the Raspberry Pi 4. Finally, the report will conclude with a demonstration of the system's capabilities through a practical application.

Chapter 1

System's Principles

In the field of voice-activated systems, a Keyword Spotting System (KWS) is an essential component that enables devices to recognize and respond to specific spoken commands. The accuracy and efficiency of such systems rely on a well-structured processing pipeline, which transforms raw audio signals into meaningful data that can be interpreted by machine learning models. This chapter provides a detailed examination of the fundamental principles underlying the KWS, focusing on the crucial stages of the processing pipeline, such as speech feature extraction, acoustic modeling, and decision-making processes.

The processing pipeline of a KWS is designed to efficiently handle the continuous stream of audio data, extracting relevant features and making real-time predictions about the presence of specific keywords. Speech feature extraction, for instance, plays a pivotal role in converting raw audio signals into a format that is both compact and informative, allowing the deep learning models to perform accurate keyword detection. Each stage in this pipeline is meticulously engineered to ensure that the system operates effectively even in challenging environments, such as those with background noise or varying speech patterns.

In the following sections, we will explore the pipeline's structure in greater detail, starting with an overview of the pipeline used in a KWS and then moving on to the different steps inside. Understanding these stages is crucial for grasping how the system achieves its goal of reliable keyword recognition.

1.1 Pipeline of a Keyword Spotting System (KWS)

In this section, we delve into the core principles underlying the keyword spotting system, which plays a crucial role in enabling voice control for consumer devices. A keyword spotting system typically processes audio input to detect specific, predefined keywords that trigger certain actions. The efficiency and accuracy of such systems depend heavily on the architecture of the processing pipeline.

The pipeline of a Keyword Spotting System (KWS) involves several sequential stages, each contributing to the accurate recognition of keywords within a speech signal. These stages include feature extraction, acoustic modeling, and posterior handling, which together ensure that the system can robustly identify keywords even in the presence of noise and other variations in the audio signal.

We will now see the general structure of the pipeline used in a KWS, followed by an example of its implementation in a real-world scenario.

1.1.1 General Pipeline

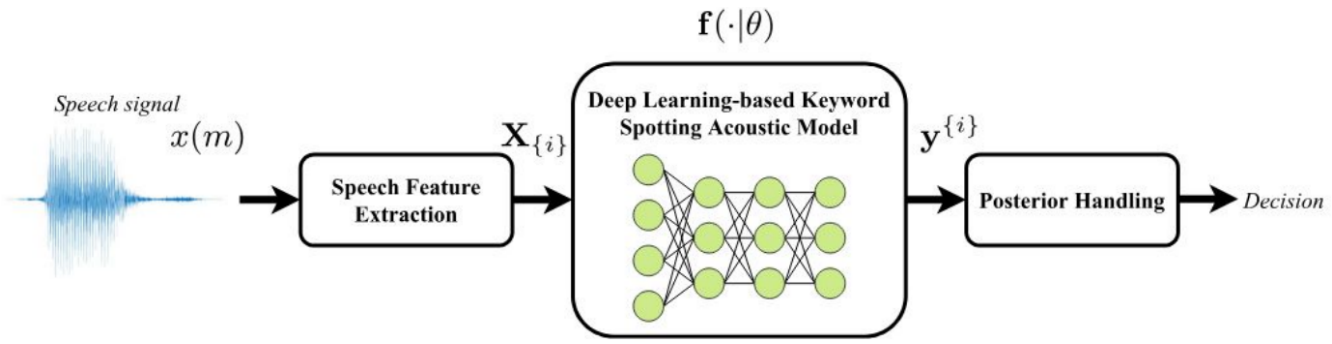


Figure 1.1: General Pipeline for a KWS [1]

The figure 1.1 illustrates the pipeline of a deep learning-based keyword spotting system. This pipeline comprises several key stages. Firstly, the speech signal $x(m)$ is captured and goes through a speech feature extraction phase, where relevant features X_i are extracted to effectively represent the speech signal. These extracted features are then fed into a deep learning-based acoustic model specifically designed for keyword spotting. The model produces outputs y_i, which are processed by a posterior handling stage to make final decisions on keyword detection.

1.1.2 Example of a Keyword Spotting System

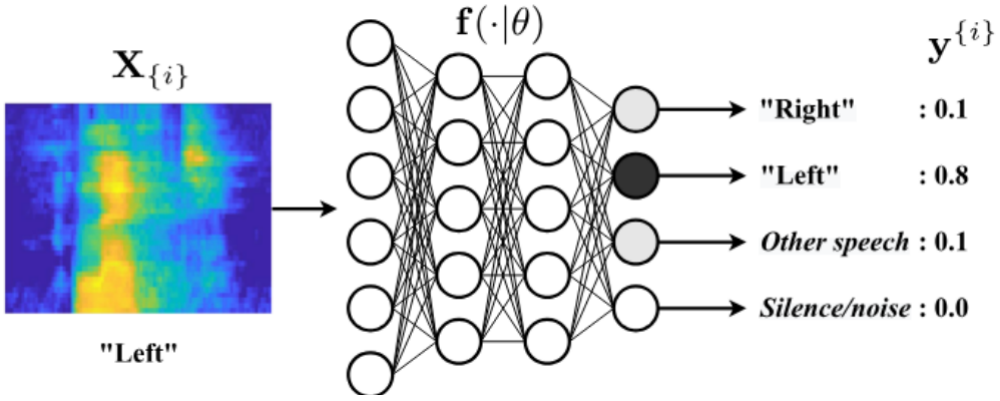


Figure 1.2: Example Pipeline for a KWS [1]

The figure 1.2 illustrates the example of a deep learning-based keyword spotting system. Here the spectrogram of the input speech signal x_i corresponding to the word "Left" is extracted (speech feature extraction phase). The spectrogram features are then fed into a DNN (Deep Neural Network). The model produces outputs y_i, which are probabilities of the keyword detected from the input x_i between 4 keywords and filler (non-keyword) classes : ["Right";"Left;"Silence/noise";"Other speech"]. Keyword "left" has the highest posterior probability = 0.8

1.2 Speech Feature Extraction

Speech feature extraction is a crucial step in the pipeline of a KWS. This process involves transforming raw audio signals into a set of features that capture the essential characteristics of speech, making it easier for machine learning models to detect keywords accurately. In this section, we will explore the methods used for speech feature extraction, focusing on the techniques such as Mel Spectrograms and MFCCs.

1.2.1 Introduction to Speech Feature Extraction

The goal of speech feature extraction is to reduce the dimensionality of the audio data while retaining the information necessary to differentiate between different speech sounds. This process begins with the raw audio signal, which is typically a one-dimensional time-series data representing the air pressure variations captured by a microphone. However, raw audio signals are often too complex for direct use in KWS models. Therefore, we employ feature extraction techniques that transform these signals into a more manageable and informative form.

1.2.2 Pipeline for Speech Feature Extraction

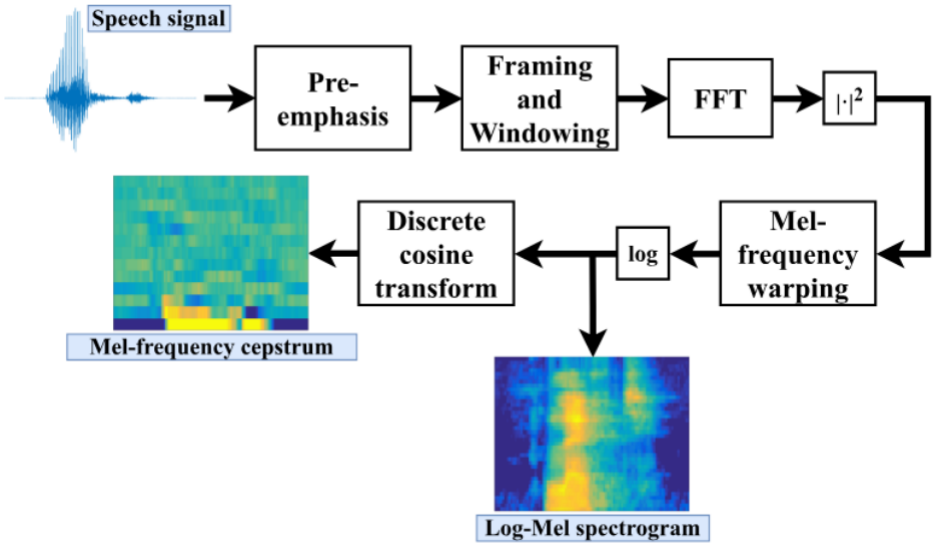


Figure 1.3: Classical pipeline for extracting log-Mel spectral and Mel-frequency cepstral speech features using the fast Fourier transform (FFT) [1]

Here you can see 2 different speech features extracted, and in the next subsections I will try to explain the process to get them (FFT is the digital version of STFT).

1.2.3 Mel Spectrograms

One of the most commonly used methods in speech feature extraction is the creation of Mel Spectrograms. A Mel Spectrogram is a time-frequency representation of the audio signal, where the frequency axis is mapped onto the Mel scale. The Mel scale is a perceptual scale of pitches judged by listeners to be equal in distance from one another, making it more aligned with human auditory perception.

Short-Time Fourier Transform (STFT)

The first step in generating a Mel Spectrogram involves applying the STFT to the audio signal. The STFT divides the audio signal into short, overlapping segments called frames. Each frame is then subjected to a Discrete Fourier Transform (DFT) to convert it from the time domain to the frequency domain. The result is a spectrogram, which is a 2D representation of the signal with time on the x-axis, frequency on the y-axis, and the magnitude of the frequencies represented by the intensity of the colors or shades.

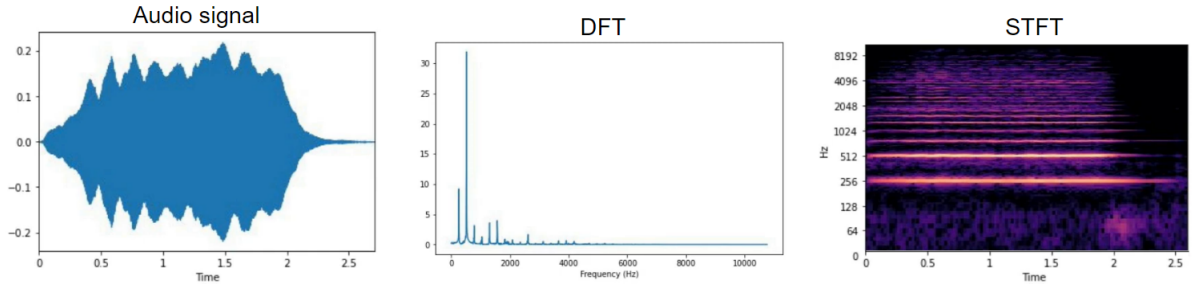


Figure 1.4: Steps of the STFT transform (from audio data to STFT)

Mel Filter Banks

Once the spectrogram is obtained, it is passed through a series of Mel filter banks. These filters are triangular in shape and are spaced according to the Mel scale. The Mel filter banks effectively compress the frequency axis by focusing more on the lower frequencies, which are crucial for understanding speech, while reducing the emphasis on higher frequencies. The output of this filtering process is the Mel Spectrogram, where each pixel represents the

energy in a particular Mel frequency band at a specific time.

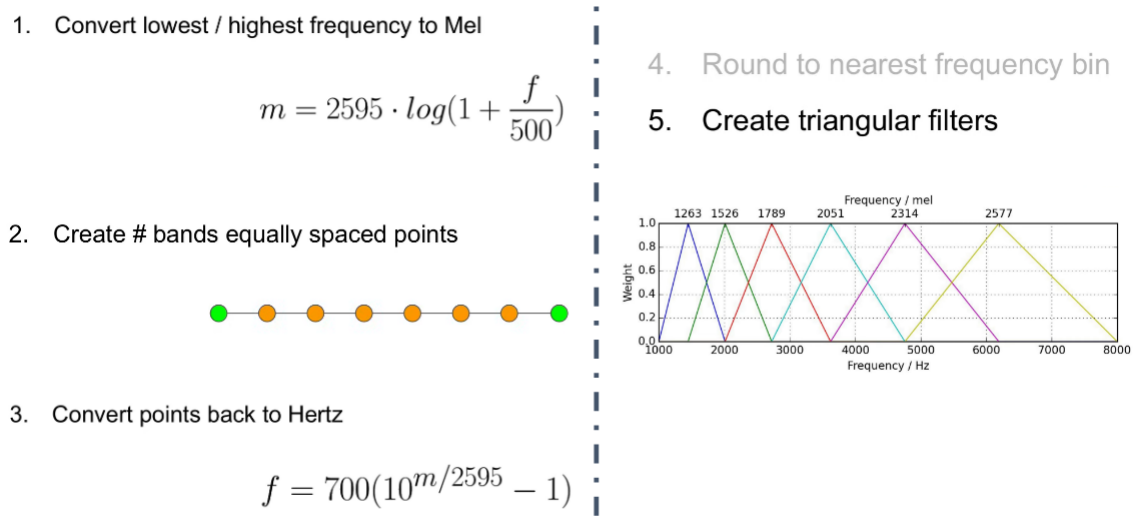


Figure 1.5: Steps to create the Mel Filter Banks

Spectrogram Visualization

The Mel Spectrogram can be visualized as a heatmap, where the x-axis represents time, the y-axis represents the Mel frequency bands, and the color intensity indicates the amplitude of the signal at that time and frequency. This visualization allows us to observe the temporal evolution of the spectral content, which is vital for recognizing patterns corresponding to different speech sounds.

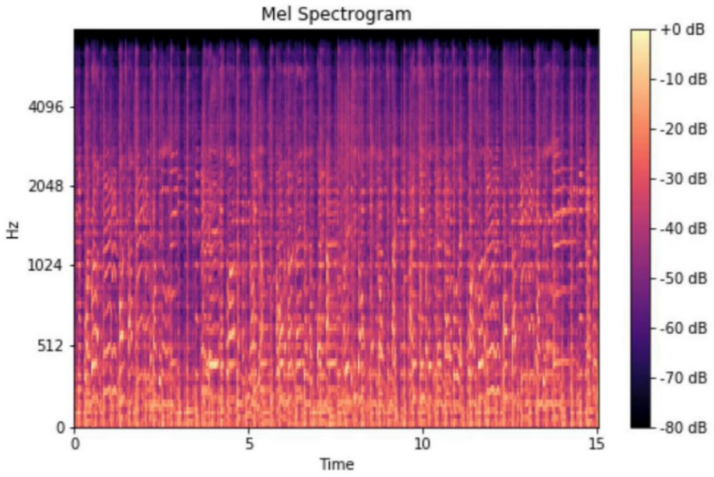


Figure 1.6: Mel Spectogram example

1.2.4 Mel-Frequency Cepstral Coefficients (MFCCs)

Mel-Frequency Cepstral Coefficients (MFCCs) are another powerful feature extraction technique widely used in speech processing. MFCCs are derived from the Mel Spectrogram and are designed to represent the short-term power spectrum of a sound.
The term *cepstrum* is inusual, in simple terms, the *cepstrum* can be thought of as the "spectrum of a spectrum." It is obtained by taking the logarithm of the power spectrum of a signal, followed by an inverse Fourier transform. This process allows for the separation of various components of the signal, making the cepstrum a valuable tool in speech processing.

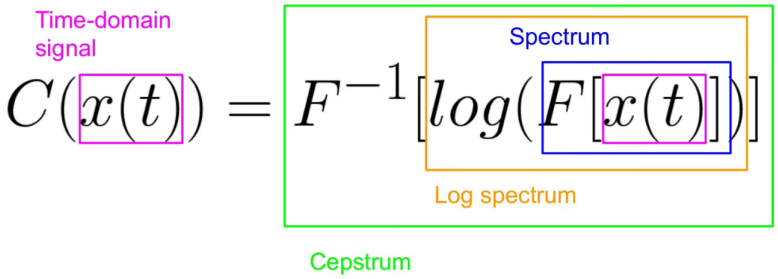


Figure 1.7: Formula to get the cepstrum of the audio signal

Logarithmic Scaling

The process of computing MFCCs starts with the Mel Spectrogram. The amplitude of each frequency component is transformed into the logarithmic scale. This logarithmic scaling mimics the human ear's response to sound, where perceived loudness increases logarithmically with actual sound intensity. This step is crucial for compressing the dynamic range of the speech signal and emphasizing the more perceptually important features.

Discrete Cosine Transform (DCT)

After applying the logarithmic scale, the next step involves performing the DCT on the log Mel Spectrogram. The DCT is used to decorrelate the filter bank coefficients, resulting in a set of coefficients that are more compact and informative. The output of the DCT is the MFCCs, which represent the spectral envelope of the audio signal.

MFCCs result

A Mel-Frequency Cepstral Coefficients (MFCC) representation typically consists of several coefficients that capture the essential characteristics of the speech signal. In a standard MFCC representation, the first 12 to 13 coefficients, known as static coefficients, represent the basic spectral envelope of the sound. These coefficients provide a compact representation of the short-term power spectrum of the speech signal.
Additionally, two types of derivative features, known as delta and delta-delta coefficients, are often computed to capture the temporal dynamics of the signal. The delta coefficients represent the rate of change of the MFCCs over time, providing information about the speech signal's dynamics. The delta-delta coefficients, which are the second derivatives, capture the acceleration or curvature of these changes. Together, the static MFCCs, delta, and delta-delta coefficients provide a comprehensive description of the speech signal, making the MFCC representation highly effective for speech recognition tasks.

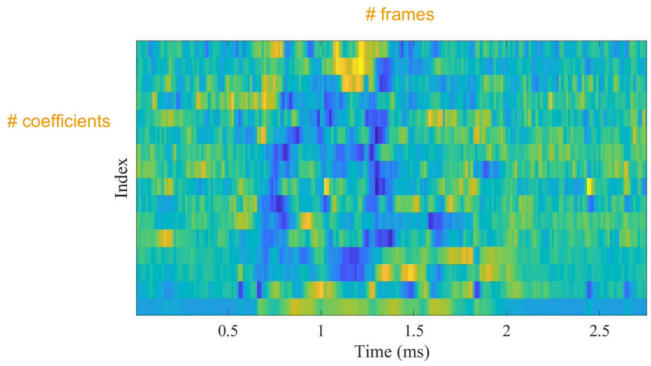


Figure 1.8: Visualisation of MFFCs

Importance of MFCCs

MFCCs are highly effective for speech recognition tasks because they capture the timbral texture of the audio signal, which is crucial for distinguishing between different phonemes. Unlike raw spectrogram features, MFCCs are relatively low-dimensional, making them suitable for training machine learning models without requiring excessive computational resources.

1.3 Comparison of Feature Extraction Methods

While both Mel Spectrograms and MFCCs are derived from the same fundamental principles, they serve different purposes in the context of speech recognition. Mel Spectrograms are particularly useful for visualizing the temporal dynamics of speech, while MFCCs provide a more compact representation that is ideal for feeding into a neural network.

1.3.1 Speech Features Extraction Summary

In summary, speech feature extraction is a critical component of the Keyword Spotting System pipeline. Techniques like Mel Spectrograms and MFCCs help distill the raw audio signal into features that are both informative and computationally manageable, enabling the subsequent stages of the system to perform accurate keyword detection. Understanding these methods is essential for designing systems that can operate effectively in real-world environments, where speech signals are often corrupted by noise and other distortions.

1.4 Deep Learning Model

In this section, we will explore the deep learning model chosen for the KWS system. The selection of the appropriate model architecture is crucial for ensuring accurate and efficient keyword detection. Based on a review of relevant studies and practical considerations, we have chosen to implement a CNN for this task.

1.4.1 Justification for Choosing CNN

Convolutional Neural Networks (CNNs) have emerged as one of the most effective architectures for processing time-series and image-like data, which makes them particularly well-suited for tasks like keyword spotting. Several studies have demonstrated that CNNs offer an excellent balance between accuracy and computational complexity, especially when compared to other architectures like Recurrent Neural Networks (RNNs) or Convolutional Recurrent Neural Networks (CRNNs).

ID	Description	Year	Accuracy (%)		Computational complexity	
			GSCD *v1*	GSCD *v2*	No. of params.	No. of mults.
1	Standard FFNN with a pooling layer [32]	2020	91.2	90.6	447k	–
2	DenseNet with trainable window function and mixup data augmentation [67]	2018	92.8	–	–	–
3	Two-stage TDNN [58]	2018	94.3	–	251k	25.1M
4	CNN with striding [32]	2018	95.4	95.6	529k	–
5	BiLSTM with attention [133]	2018	95.6	96.9	202k	–
6	Residual CNN `res15` [30]	2018	95.8 ± 0.484	–	238k	894M
7	TDNN with shared weight self-attention [16]	2019	95.81 ± 0.191	–	12k	403k
8	DenseNet+BiLSTM with attention [48]	2019	96.2	97.3	223k	–
9	Residual CNN with temporal convolutions `TC-ResNet14` [50]	2019	96.2	–	137k	–
10	SVDF [32]	2019	96.3	96.9	354k	–
11	SincConv+(Grouped DS-CNN) [70]	2020	96.4	97.3	62k	–
12	Graph convolutional network `CENet-40` [49]	2019	96.4	–	61k	16.18M
13	GRU [32]	2020	96.6	97.2	593k	–
14	SincConv+(DS-CNN) [70]	2020	96.6	97.4	122k	–
15	Temporal CNN with depthwise convolutions `TENet12` [52]	2020	96.6	–	100k	2.90M
16	Residual DS-CNN with squeeze-and-excitation `DS-ResNet18` [51]	2020	96.71 ± 0.195	–	72k	285M
17	`TC-ResNet14` with neural architecture search `NoisyDARTS-TC14` [146]	2021	96.79 ± 0.30	97.18 ± 0.26	108k	6.3M
18	LSTM [32]	2020	96.9	97.5	–	–
19	DS-CNN with striding [32]	2018	97.0	97.1	485k	–
20	CRNN [32]	2020	97.0	97.5	467k	–
21	BiGRU with multi-head attention [32]	2020	97.2	98.0	743k	–
22	CNN with neural architecture search `NAS2_6_36` [125]	2020	97.22	–	886k	–
23	Keyword Transformer `KWT-3` [90]	2021	97.49 ± 0.15	98.56 ± 0.07	5.3M	–
24	Variant of TC-ResNet with self-attention `LG-Net6` [91]	2021	97.67	96.79	313k	–
25	Broadcasted residual CNN `BC-ResNet-8` [100]	2021	98.0	98.7	321k	89.1M

Figure 1.9: Performance comparison among some of the latest deep KWS systems in terms of both accuracy parameters and multiplications) of the acoustic model. Accuracy, provided with confidence intervals for some systems, is on the Google Speech Commands Dataset (GSCD) v1 and v2. Results from Deep Spoken article [1]

CNNs excel in recognizing local patterns in data, which is a key requirement for keyword spotting where the model needs to identify specific patterns within the speech signal. Furthermore, CNNs are relatively less computationally intensive compared to RNNs, making them more suitable for deployment on resource-constrained devices such as the Raspberry Pi 4.

The choice of CNN for this project is therefore justified by its proven effectiveness in similar tasks, its ability to handle the local feature extraction from the spectrograms and MFCCs, and its compatibility with the hardware constraints of the target embedded system.

1.4.2 Principles of CNNs

CNNs are a class of deep learning models specifically designed to process data with a grid-like topology, such as images or spectrograms. A CNN typically consists of several layers, each performing a specific function to progressively transform the input data into a more abstract and useful representation.

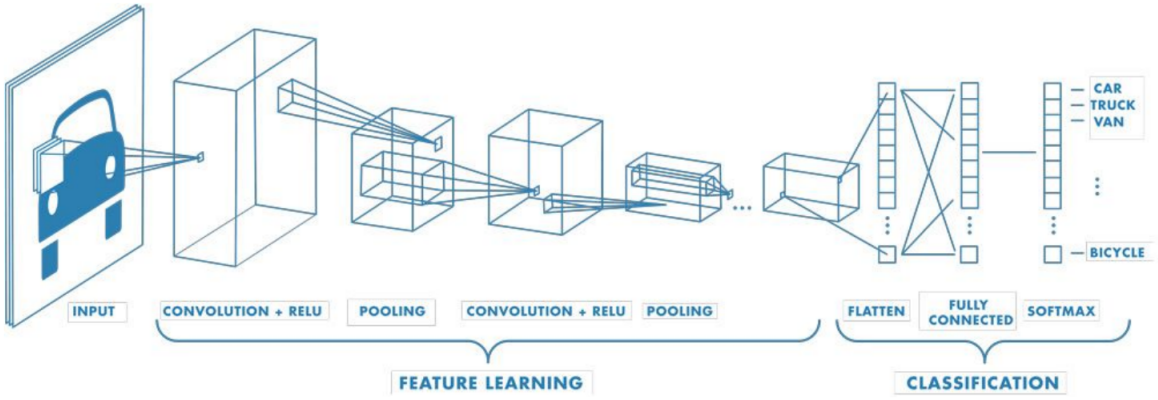


Figure 1.10: Visualisation of a CNN and its components

The main components of a CNN include:
Convolutional Layers: These layers apply a series of convolutional filters (kernels) to the input data. Each filter is designed to detect specific patterns or features within the data, such as edges, textures, or more complex shapes. The output of the convolutional layers is a set of feature maps that highlight the presence of these patterns.
Activation Functions: After each convolutional operation, an activation function is applied to introduce non-linearity into the model. The most commonly used activation function in CNNs is the Rectified Linear Unit (ReLU), which helps the network learn more complex patterns.
Pooling Layers: These layers reduce the spatial dimensions of the feature maps, typically using operations like max pooling. Pooling helps to reduce the computational load and the number of parameters in the network, while also providing some translation invariance.
Fully Connected Layers: At the end of the convolutional and pooling layers, the output is flattened and passed through one or more fully connected layers. These layers combine the extracted features to produce the final output, such as the probability distribution over the possible keywords.

1.4.3 Choice of Parameters

The effectiveness of a CNN depends heavily on the choice of hyperparameters, which include the number of layers, the size of the convolutional filters, the stride, the type and size of the pooling layers, and the learning rate, among others. Each of these parameters plays a critical role in determining the model's ability to accurately detect keywords while maintaining computational efficiency.

Number of Layers: The depth of the CNN, determined by the number of convolutional layers, directly impacts the model's capacity to learn hierarchical features. With more layers, the network can capture increasingly complex patterns in the data, potentially leading to better accuracy. However, deeper networks also come with the risk of overfitting and increased computational demands, which may not always result in significant improvements in performance, especially on smaller datasets or in real-time applications.

Filter Size and Stride: The size of the convolutional filters (or kernels) and the stride (the step size with which the filter moves across the input) are crucial for feature extraction. Smaller filters, such as 3x3, are often preferred because they allow the network to capture fine-grained details in the data, while larger strides might skip over important features, leading to a loss of spatial resolution. The choice of stride must balance thorough data scanning with the need to reduce the dimensionality of the feature maps.

Pooling Strategy: Pooling layers are used to downsample the feature maps, reducing their dimensionality and the number of parameters in the network. Max pooling, in particular, is effective in retaining the most significant features by selecting the maximum value within a pooling window. A typical pooling window size is 2x2, which helps in reducing the computational burden while preserving essential features. However, excessive pooling can lead to a loss of important spatial information, so it's crucial to strike the right balance.

Learning Rate and Optimization: The learning rate controls how quickly the model adjusts its parameters in response to the error calculated during training. A too-high learning rate can cause the model to converge too quickly to a suboptimal solution or even diverge, while a too-low learning rate can make the training process excessively slow. An appropriate learning rate ensures that the model converges effectively to a minimum of the loss function. Optimization algorithms like Adam are widely used for their efficiency in handling noisy gradients and their ability to dynamically adjust the learning rate during training, leading to faster convergence and better

generalization.

The choice and tuning of these parameters significantly influence the performance of the CNN, affecting both its accuracy in detecting keywords and its suitability for deployment in resource-constrained environments. By carefully balancing these factors, it is possible to design a CNN that meets the specific requirements of the keyword spotting system.

1.5 Streaming Mode

In the context of keyword spotting systems, the streaming mode refers to the continuous processing of an audio stream to detect keywords in real-time. Unlike non-streaming modes, which process isolated segments of audio independently, streaming mode is designed to handle a continuous flow of data, making it suitable for applications like voice assistants, smart home devices, and other real-time voice-activated systems. This section delves into the principles, challenges, and implementation strategies associated with streaming mode in keyword spotting.

1.5.1 Principles of Streaming Mode

The primary goal of streaming mode is to enable the keyword spotting system to monitor and analyze incoming audio in real-time, detecting keywords as they occur. This requires the system to continuously process overlapping segments of the audio stream, applying the keyword detection model to each segment and making rapid decisions. To achieve this, the audio stream is divided into short, overlapping frames or windows. Each window is processed sequentially by the feature extraction and deep learning models, producing a series of predictions. These predictions are then analyzed over time to determine the presence of a keyword. The use of overlapping windows is crucial because it ensures that no keyword is missed, even if it occurs at the boundary between two windows.

1.5.2 Challenges in Streaming Mode

Implementing streaming mode presents several unique challenges compared to non-streaming modes. These challenges arise primarily from the need to process audio in real-time while maintaining high accuracy and low latency.
Latency vs. Responsiveness: One of the key challenges is balancing latency and responsiveness. Latency refers to the delay between the occurrence of a keyword in the audio stream and the system's detection of it. To reduce latency, the system needs to process each window as quickly as possible. However, if the window size or stride is too small, the system may become overly sensitive, leading to an increase in false positives. Conversely, if the windows are too large, the system may miss short keywords or exhibit delayed responses. Therefore, selecting the optimal window size and stride is critical for maintaining a balance between low latency and high detection accuracy.
Handling Overlapping Windows: Another challenge in streaming mode is the handling of overlapping windows. Because the audio stream is divided into overlapping segments, the system needs to ensure that the predictions made for each segment are consistent and accurate. This often involves smoothing techniques or the use of temporal sequence analysis to aggregate predictions over multiple windows. By doing so, the system can reduce the likelihood of false detections and improve the reliability of keyword spotting.
Resource Constraints: Streaming mode also imposes significant computational demands, particularly on resource-constrained devices such as embedded systems or mobile platforms. The need to process audio in real-time requires the system to be highly optimized, both in terms of model complexity and computational efficiency. This may involve trade-offs between model accuracy and processing speed, as well as the implementation of efficient algorithms for feature extraction and prediction.

1.5.3 Implementation of Streaming Mode

The implementation of streaming mode in a keyword spotting system involves several key steps. These steps are designed to ensure that the system can continuously process audio and accurately detect keywords in real-time.

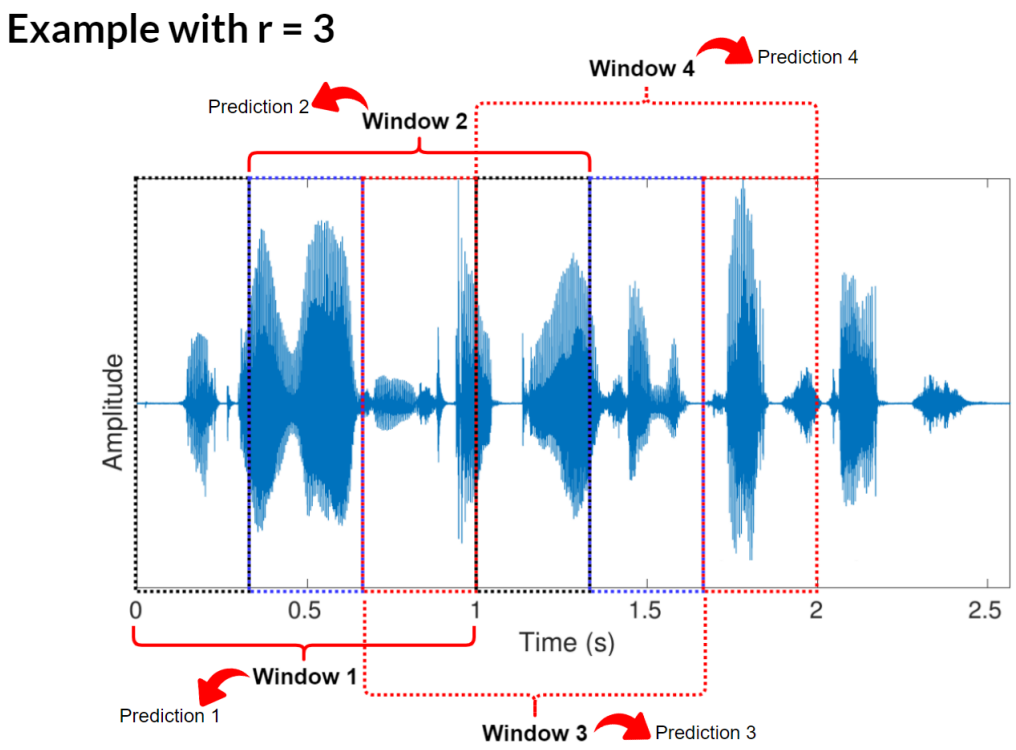


Figure 1.11: Visualisation of the streaming mode implementation

Windowing and Buffering: The first step in the implementation is the division of the audio stream into windows. A common approach is to use a sliding window mechanism, where each window overlaps with the previous one by a certain percentage. The audio data within each window is then buffered and processed sequentially. The choice of window size and overlap percentage is crucial, as it directly impacts the system's ability to detect keywords and its overall responsiveness.
Feature Extraction in Real-Time: Once the audio is segmented into windows, each window is passed through the feature extraction pipeline. In streaming mode, this process must be highly efficient to keep up with the continuous flow of data. Techniques such as MFCC extraction are applied to each window and then fed into the keyword detection model.
Sequential Prediction and Smoothing: As the system processes each window, it generates predictions about the presence of a keyword. These predictions are made in sequence, with each window producing its own set of results.
Handling Edge Cases: Special consideration must be given to edge cases, such as the initial and final windows of the audio stream. During the initial stages, the buffer may not be fully populated. To address this, padding or waiting strategies may be employed to ensure that the buffer is filled before making any predictions.

Streaming mode is a critical component of keyword spotting systems, enabling real-time voice interaction. By carefully designing and optimizing the streaming process, it is possible to create an efficient keyword spotting system capable of operating in real-world environments.

Chapter 2

Creation of an efficient Model

In this chapter, we will explore the specifics of the model implemented for the Keyword Spotting System. Initially, the model was trained and evaluated without any data augmentation to establish a baseline performance. However, after simulating real-world conditions by introducing noise into the test set, the model's performance significantly deteriorated, highlighting the need for more robust training techniques. This led to the implementation of data augmentation methods, specifically noise addition, to enhance the model's robustness and generalization ability.

2.1 Dataset Description

The foundation of any machine learning model lies in the quality and quantity of the data it is trained on. For this project, the SpeechCommands dataset version 0.02 was selected as the primary source of training and testing data, as it is a norm in the KWS field for studies [1],[2],[3]. The dataset contains thousands of labeled one-second long utterances of 35 different keywords (around 3 thousand samples per keyword):
['backward', 'bed', 'bird', 'cat', 'dog', 'down', 'eight', 'five', 'follow', 'forward', 'four', 'go', 'happy', 'house', 'learn', 'left', 'marvin', 'nine', 'no', 'off', 'on', 'one', 'right', 'seven', 'sheila', 'six', 'stop', 'three', 'tree', 'two', 'up', 'visual', 'wow', 'yes', 'zero'].
Each keyword represents a command that the model should be able to recognize. In addition to the keywords, the dataset also includes background noise files.

The dataset was split into training, validation, and test sets to ensure that the model was properly evaluated during development. Approximately 80 percent of the data was used for training, 10 percent for validation, and the remaining 10 percent for testing. This split allowed for rigorous testing and tuning of the model, ensuring it performed well on unseen data.

2.2 Initial Model Performance Without Data Augmentation

The model was first trained on the original dataset without any data augmentation. For preprocessing, the Mel-Frequency Cepstral Coefficients (MFCC) transformation was applied to the audio data. This transformation is crucial for converting raw audio signals into features that capture the essential characteristics of speech, making them more suitable for analysis by the deep learning model.

A Convolutional Neural Network (CNN) with four layers was then employed, similar to the architectures used in related studies. This model was designed to capture the hierarchical features of the MFCC-transformed data, enabling it to recognize patterns corresponding to different keywords. After training, the model achieved satisfactory accuracy on the test set, about 91 percent accuracy after 21 epochs of training, indicating that it could effectively recognize the target keywords in relatively clean, noise-free conditions.

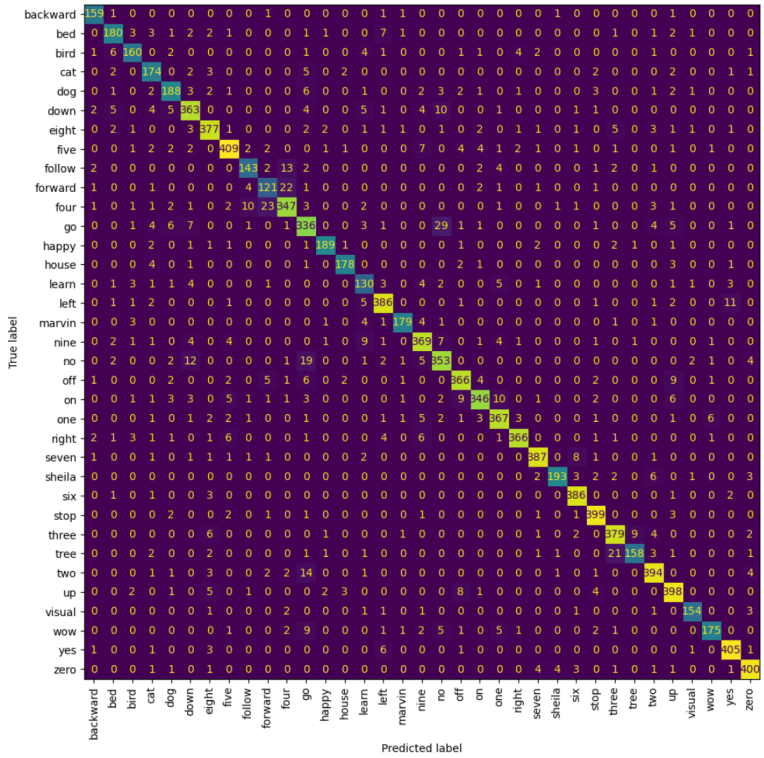


Figure 2.1: Confusion Matrix for Initial Model Performance (91 percent Accuracy) without Data Augmentation

However, while the model performed well in this controlled environment, it was essential to test its robustness in more challenging, real-world scenarios where background noise is present.

2.2.1 Simulating Real-World Conditions: Noise in the Test Set

After testing the precedent model with my own microphone inputs, i noticed a huge difference between my samples and the one in the dataset, because even in a calm environment I could notice noise on the spectrograms of my inputs, that why I decided to test my model in similar conditions. To simulate real-world conditions, background noise was added randomly to the test set at various SNRs (firstly SNR = 10, then SNR = 2). This simulation aimed to evaluate the model's performance in noisy environments, reflecting common use cases where the system might be deployed in homes, offices, or public spaces with varying levels of ambient noise.

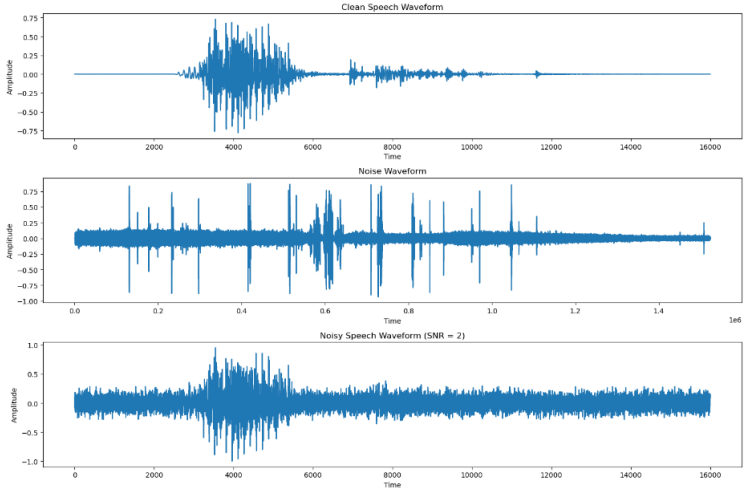


Figure 2.2: Visualisation of the Noise Addition on a sample from the Test Set with an SNR=2

The results were revealing: the model's accuracy dropped significantly when tested on noisy data and achieved only a 47 percent accuracy score. This indicated that the model, while effective in clean conditions, struggled to maintain its performance in the presence of noise. The degradation in performance underscored the need for data augmentation techniques to improve the model's robustness and generalization to noisy environments.

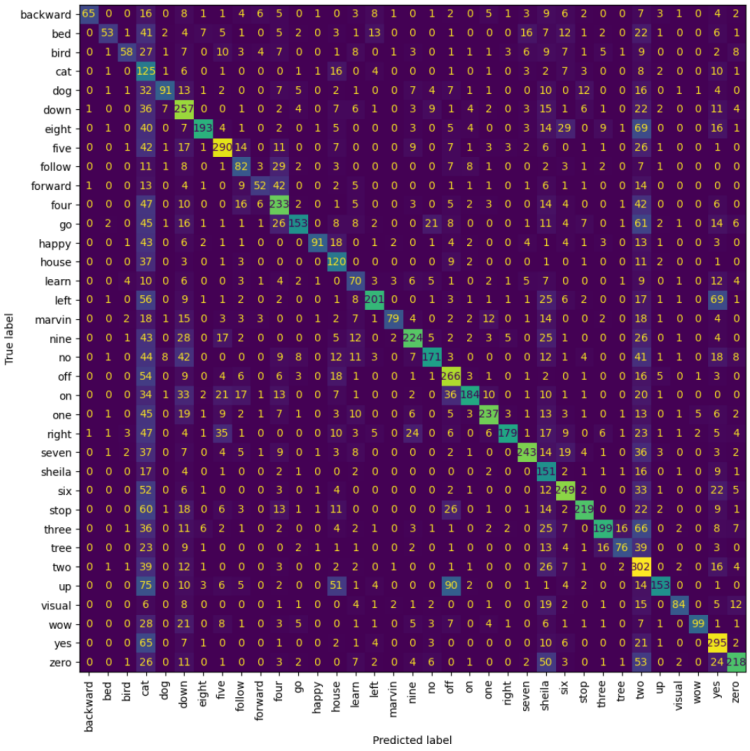


Figure 2.3: Confusion Matrix for Initial Model Performance (47 percent Accuracy) without Data Augmentation on the Noisy Test Set

2.3 Data Augmentation Techniques: Improving Robustness

To address the performance issues observed under noisy conditions, data augmentation techniques were employed to enhance the model's ability to handle such environments. Data augmentation artificially expands the dataset by creating modified versions of the existing data, which helps the model learn to recognize keywords under varying conditions. Among the techniques applied, noise addition was particularly critical.

2.3.1 Noise Addition

Noise addition is a crucial data augmentation technique used to simulate real-world environments where background noise is present. Random noise samples, such as white noise or background chatter, were added to the original audio recordings at varying Signal-to-Noise Ratios (SNR). By training the model on this augmented data, the goal was to make it more resilient to noisy conditions, which are common in real-world applications.

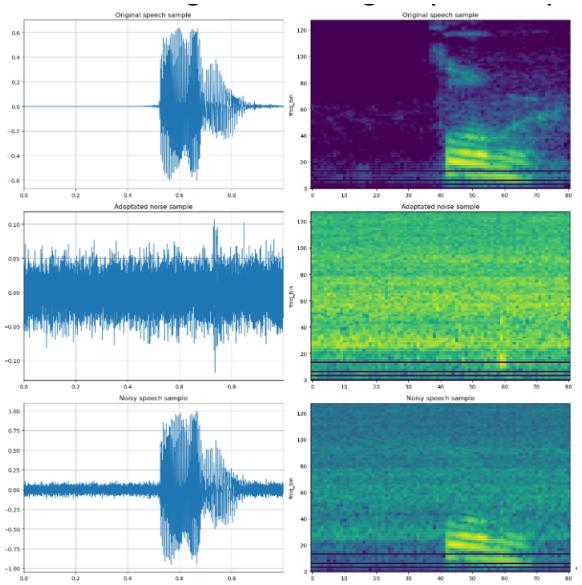


Figure 2.4: Visualisation of the Noise Addition on a sample from the Training Set with an SNR=10

The impact of this augmentation was significant: the model trained with noise-augmented data showed a marked improvement in accuracy when evaluated on the noisy test set, compared to the model trained on the original dataset alone. This demonstrated that the model could now better generalize to real-world conditions, where background noise is inevitable. The final results were more stable between noisy and non-noisy conditions, we achieved to have almost 80 percent accuracy (82 percent and 78 percent) on both test set (noisy and non-noisy)

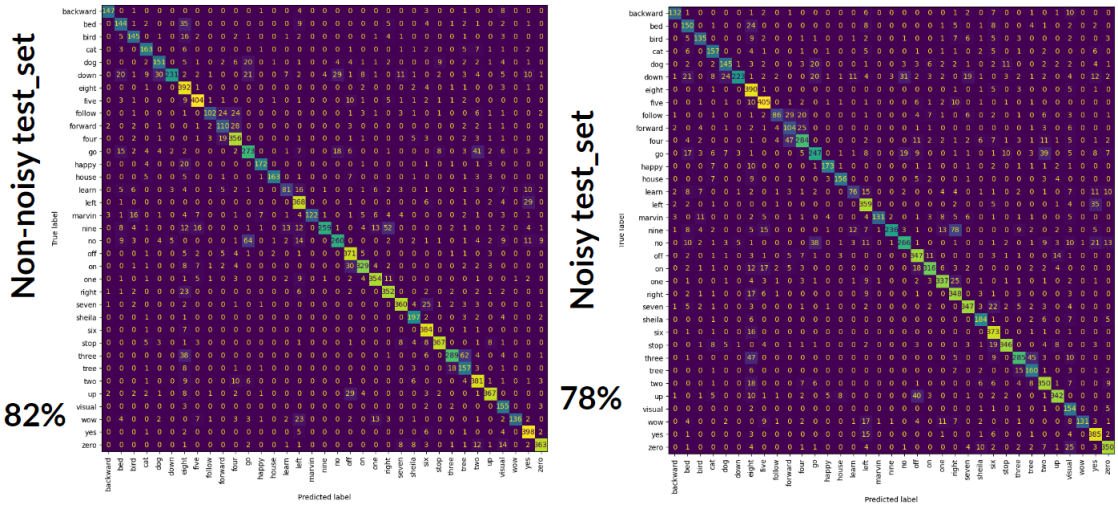


Figure 2.5: Confusion Matrices for Data Augmented Model Performance on both Non-Noisy Test Set and Noisy Test Set

During training, the model's performance was monitored on the validation set, with adjustments made to the learning rate and other hyperparameters as needed. The Adam optimizer was used to minimize the loss function, and training was conducted over multiple epochs to ensure that the model converged to an optimal solution.

After training with the augmented data, the model was re-evaluated on the noisy test set. The performance metrics included accuracy, precision, recall, and F1-score. The model showed a significant improvement in its ability to detect keywords in the presence of background noise, compared to its performance prior to data augmentation. This validated the effectiveness of noise addition as a data augmentation technique, confirming its role in enhancing the robustness of the keyword spotting system in real-world scenarios.

Chapter 3

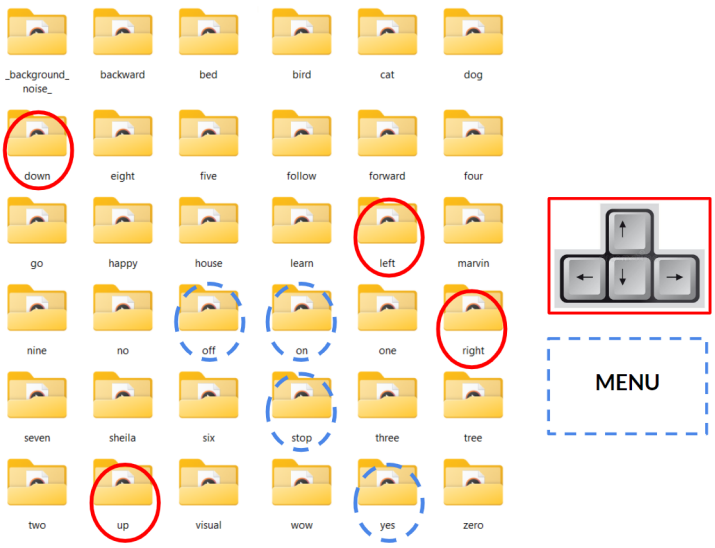
Implementation and Usage

After detailing the underlying concepts and techniques, we are now equipped to build a practical and efficient system capable of being deployed on an embedded platform. However, before proceeding with the implementation, it is essential to define a specific use case and adapt our model to suit this application. Additionally, new concepts such as silence detection must be integrated. In the following sections, we will explore the selection of a use case and the steps involved in implementing the system on an embedded device.

3.1 Use Case Selection

With a wide array of keywords at our disposal, the potential applications are virtually limitless. However, to focus our efforts, we must select a specific use case. I began by reviewing the available keywords and identified several that seemed less practical, such as 'dog', 'house', and 'tree'. Instead, I concentrated on simpler, more functional keywords. The keywords 'left', 'right', 'down', and 'up' caught my attention as they could be used to simulate the arrow keys on a keyboard.

This observation led me to consider a playful application: the Snake Game. By incorporating additional keywords like 'stop', 'yes', 'on', and 'off', we could create a fully functional game and menu system controlled entirely by voice commands.



With the concept in place, the next step is implementation. This requires two key components: first, a new, specialized model tailored to the selected keywords, and second, a program that runs a snake game controlled via speech recognition and keyword detection.

3.1.1 Developing the Snake Model

The new model must be capable of recognizing the specific keywords ['left', 'right', 'down', 'up', 'stop', 'yes', 'on', 'off'], as well as detecting instances where none of these keywords are spoken. In such cases, the model should

determine whether the input is silence (no keyword pronounced) or an unknown word. To achieve this, we adapted the model as follows:

Method:

- Generate silence samples with added noise (ensuring the noise differs from that used in the earlier data augmentation process).
- Convert all other keywords not in the selected list into a single "unknown" label.

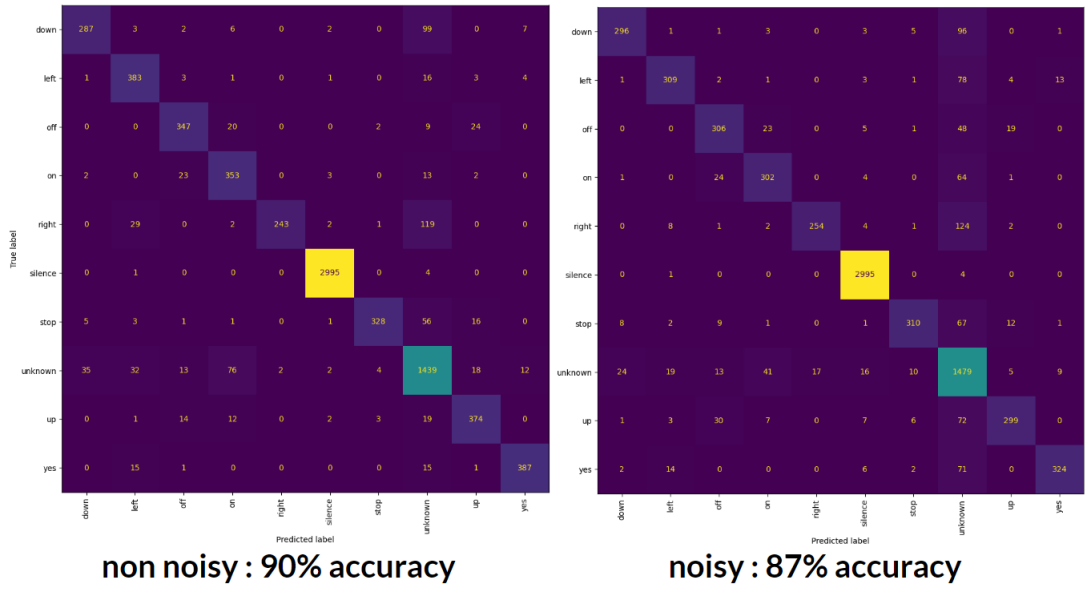


Figure 3.1: Confusion Matrices for Snake Model Performance on both Non-Noisy Test Set and Noisy Test Set

Once this new model is able to accurately predict whether a keyword from the list is spoken or detect silence or an unknown word, and with an efficient Streaming Mode in place, we can proceed to develop the Snake Game controlled by speech recognition. Below is the interface of the game:

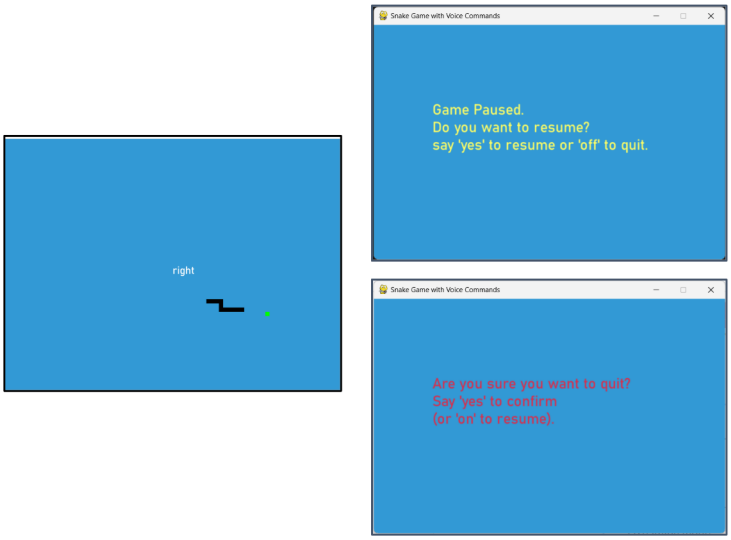


Figure 3.2: Speech Controlled Snake Game Interface

The snake is controlled using voice commands corresponding to the keywords: 'left', 'right', 'down', and 'up'. The game also includes a menu system, which can be accessed by saying 'stop'. From there, the player can say 'yes' to resume, 'off' to quit, or 'on' to resume if they accidentally triggered the menu. To exit the game, the player confirms by saying 'yes', or says 'on' to continue playing if the exit command was given by mistake.
The game is highly responsive, with the option to adjust the number of inferences per second to further enhance responsiveness. However, increasing the inference rate may impact the accuracy of the controls. After testing, it

was determined that 3 inferences per second strikes a good balance, providing a smooth and enjoyable gameplay experience.

3.2 Embedded Implementation

The final stage of this project involves deploying the keyword spotting system on an embedded platform. The choice of hardware, software considerations, and the implementation process are critical to ensuring that the system operates efficiently in a real-world environment, where resources are limited and responsiveness is crucial. In this section, we will discuss the steps taken to successfully implement the system on an embedded device, focusing on the challenges encountered and the solutions developed.

3.2.1 Hardware Selection

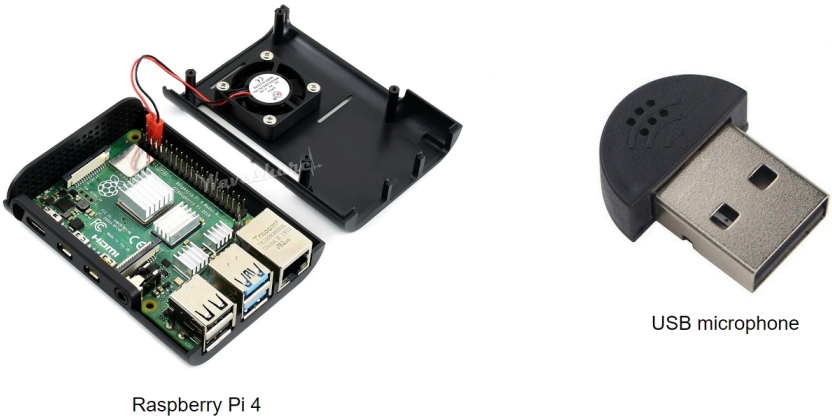


Figure 3.3: Material used

The embedded platform chosen for this implementation is the Raspberry Pi 4, a popular single-board computer that offers a good balance between processing power and energy efficiency. The Raspberry Pi 4 is equipped with a quad-core ARM Cortex-A72 CPU, up to 4GB of RAM, and supports a range of peripheral devices, making it a suitable candidate for running lightweight machine learning models. Additionally, its support for various operating systems, including Linux, and its extensive community support made it an ideal choice for this project.

The Raspberry Pi 4 was selected specifically for its ability to handle real-time audio processing and its compatibility with Python-based machine learning frameworks such as TensorFlow and PyTorch. These factors were crucial in ensuring that the keyword spotting system could be implemented effectively without the need for more expensive or power-hungry hardware.

3.3 Software Setup and Optimization

The first step in the embedded implementation was setting up the software environment. The Raspberry Pi 4 was configured with the Raspbian OS (a Debian-based Linux distribution optimized for the Raspberry Pi). The necessary libraries and dependencies, including Python, PyTorch, and the ‘sounddevice‘ library for real-time audio capture, were installed.
One of the primary challenges during the software setup was optimizing the model to run efficiently on the limited hardware resources of the Raspberry Pi. The original CNN model, which had been trained on a more powerful machine, required significant adjustments to ensure effective inference on the embedded device. This optimization process included the following key steps:

- **CPU-Only Inference:** Given the lack of GPU and CUDA support on the Raspberry Pi, the model was specifically optimized for CPU-only execution. This involved fine-tuning the model to ensure it could perform inference with minimal latency using only the CPU’s processing power.
- **Real-time Audio Processing:** The initial MFCCs transformation provided by PyTorch was incompatible with the arm64 architecture of the Raspberry Pi. To address this, I transitioned to using the Librosa module for the MFCCs transformation, ensuring reliable and efficient audio feature extraction.

3.3.1 Testing and Performance Evaluation

After the successful deployment of the system on the Raspberry Pi 4, a series of tests were conducted to evaluate its performance in real-world scenarios. These tests included both synthetic and live audio inputs to assess the accuracy, latency, and overall responsiveness of the keyword spotting system. The system demonstrated robust performance, effectively recognizing the specified keywords even in the presence of background noise.
To provide a comprehensive overview of the system's performance, I have recorded several test videos showcasing the functionality and responsiveness of the keyword spotting system. These videos can be accessed via the following Google Drive link: Test Videos .
These videos highlight the system's ability to handle various environments and demonstrate its practical applicability for real-time voice control on an embedded platform.

Chapter 4

Conclusion

The development and implementation of a Keyword Spotting System for Consumer Equipment Control have been an extensive journey through the realms of deep learning, speech processing, and embedded systems. This project set out to create an efficient and robust system capable of recognizing specific voice commands, even in challenging real-world environments, and deploying it on an embedded platform.

Initially, the project focused on understanding and implementing the core concepts of speech feature extraction and deep learning. Techniques like Mel-Frequency Cepstral Coefficients (MFCCs) were utilized to transform raw audio signals into informative features, while Convolutional Neural Networks (CNNs) were employed to accurately detect keywords within these features. Through rigorous experimentation, the importance of data augmentation was highlighted, particularly the addition of noise to the dataset, which significantly improved the model's robustness against real-world conditions.

The transition from theoretical development to practical implementation involved deploying the trained model on a Raspberry Pi 4, an embedded platform with limited computational resources. This phase of the project required careful optimization, including model quantization and the adaptation of the audio processing pipeline to operate efficiently without GPU support. The challenges of real-time processing and the constraints of embedded systems were successfully addressed, resulting in a responsive and accurate keyword spotting system.

Moreover, a specific use case was chosen to demonstrate the system's capabilities—a voice-controlled Snake Game. This application not only illustrated the practical utility of the system but also provided an engaging and interactive way to evaluate its performance in a real-world scenario.

In conclusion, the project achieved its goals by developing a functional and efficient keyword spotting system that operates effectively on an embedded platform. The system's ability to recognize voice commands with high accuracy, even in noisy environments, demonstrates the success of the chosen methodologies. The work completed in this project lays a strong foundation for further exploration and enhancement of voice-activated systems in consumer electronics, particularly in the realm of embedded applications. Future work could focus on refining the model's accuracy, expanding its vocabulary, and exploring additional use cases that further integrate voice control into everyday consumer devices.

References

[1] Iván López-Espejo, Zheng-Hua Tan, John Hansen, Jesper Jensen, "Deep Spoken Keyword Spotting: An Overview," arXiv preprint arXiv:2111.10592, 2021.

[2] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello Edge: Keyword Spotting on Microcontrollers," arXiv preprint arXiv:1711.07128, 2017.

[3] S. K. Gouda, S. Kanetkar, D. Harrison, and M. K. Warmuth, "Speech Recognition: Keyword Spotting Through Image Recognition," arXiv preprint arXiv:1803.03759, 2018.

Abstract

In this report, we present the design and implementation of a Keyword Spotting System (KWS) aimed at enabling voice-controlled interaction with consumer electronic devices. The project leverages advanced deep learning techniques, specifically Convolutional Neural Networks (CNNs), combined with robust speech feature extraction methods such as Mel-Frequency Cepstral Coefficients (MFCCs), to accurately recognize predefined voice commands. Initially, the system was developed and evaluated in a controlled environment, achieving high accuracy on clean, noise-free data. However, real-world conditions often introduce background noise, which can significantly degrade the performance of voice recognition systems. To address this challenge, data augmentation techniques, particularly noise addition, were applied to improve the model's robustness and generalization to noisy environments.
The final system was deployed on a Raspberry Pi 4, an embedded platform chosen for its balance of processing power and energy efficiency. The model was optimized for CPU-only execution ensuring efficient real-time performance. As a demonstration of the system's capabilities, a voice-controlled Snake Game was developed, showcasing the practical application of the KWS in a real-world scenario.
The project successfully demonstrates the feasibility of deploying an efficient and robust keyword spotting system on an embedded platform, laying the groundwork for future advancements in voice-controlled consumer electronics.

Keywords: Keyword Spotting, Convolutional Neural Networks, MFCC, Data Augmentation, Embedded Systems, Raspberry Pi, Voice Control.

Résumé (FR)

Dans ce rapport, nous présentons la conception et la mise en œuvre d'un système de détection de mots-clés (Keyword Spotting System, KWS) destiné à permettre l'interaction vocale avec des appareils électroniques grand public. Le projet exploite des techniques avancées d'apprentissage profond, en particulier les réseaux de neurones convolutifs (CNN), combinées à des méthodes robustes d'extraction de caractéristiques vocales telles que les coefficients cepstraux en fréquences Mel (MFCC), pour reconnaître avec précision des commandes vocales prédéfinies.
Dans un premier temps, le système a été développé et évalué dans un environnement contrôlé, obtenant une précision élevée sur des données sans bruit. Cependant, les conditions réelles introduisent souvent des bruits de fond, ce qui peut dégrader significativement les performances des systèmes de reconnaissance vocale. Pour relever ce défi, des techniques d'augmentation de données, notamment l'ajout de bruit, ont été appliquées pour améliorer la robustesse du modèle et sa capacité à se généraliser à des environnements bruyants.
Le système final a été déployé sur un Raspberry Pi 4, une plateforme embarquée choisie pour son équilibre entre puissance de traitement et efficacité énergétique. Le modèle a été optimisé pour une exécution uniquement sur le CPU assurant ainsi des performances en temps réel efficaces. Pour démontrer les capacités du système, un jeu de serpent contrôlé par la voix a été développé, illustrant l'application pratique du KWS dans un scénario réel.
Le projet démontre avec succès la faisabilité du déploiement d'un système de détection de mots-clés efficace et robuste sur une plateforme embarquée, posant ainsi les bases de futurs développements dans le domaine des appareils électroniques grand public contrôlés par la voix.
Mots-clés : Détection de mots-clés, Réseaux de neurones convolutifs, MFCC, Augmentation de données, Systèmes embarqués, Raspberry Pi, Contrôle vocal.