university of groningen

Keyword Spotting Using Pretrained Audio Embeddings

Authors:
Alessandro Castoldi (s6133800)
Sebastian Jitaru (s6019595)
Georgios Skoulidis (s6050786)

Abstract

The task of identifying a keyword within a continuous stream of raw audio has become an important feature in modern voice-activated systems. Applications range from the healthcare sector, where human-aid robots continuously listen for distress calls to assist individuals in need, to widely used consumer devices such as Amazon Alexa, Apple Siri, and Google Assistant, which allow users to interact with the devices without using physical contact.

In this project, we build upon the existing keyword spotting pipeline from OpenWakeWord [10], which reimplements PyTorch functions for generating Mel spectrograms from raw audio. These spectrograms are then used to generate audio embeddings through an implementation based on [6], followed by a neural network classifier for final prediction.

Our work tests whether an alternative simpler model can be an effective classifier of the audio embeddings for keyword detection. Upon exploring the dataset, we observed that the negative and positive embeddings were well segregated, suggesting the potential for simpler classification methods. We use ridge logistic regression on PCA-reduced features. Both the deep neural network and the linear model were evaluated using 5-fold stratified cross-validation, with binary cross-entropy loss optimized via Adam (learning rate 1e-3). Notably, PCA confirmed the linear separability of the embeddings. The linear model using 50 principal components achieved an accuracy of 99.60% and an F1-score of 98.88%, slightly outperforming the fully connected network. These findings suggest that a lightweight, interpretable classifier may be sufficient for this task.

Contents

1 Introduction

Detecting keywords in continuous streams of raw audio speech has become an important piece for the correct functioning of modern voice-activated systems. Keyword spotting (KWS) is used in applications such as Amazon Alexa, Google Assistant, and Apple Siri, where a keyword triggers the execution of further processes, for instance, the beginning of a conversation. Moreover, KWS is also used in domains such as healthcare, where human-aid robots are listening for distress signals, allowing for real-time assistance [2].

Traditional approaches to keyword spotting used handcrafted features such as Mel Frequency Cepstral Coefficients (MFCCs) [5]. However, these methods required a higher amount of manual work or were not able to adapt to issues such as noisy environments. In this project, we build upon the OpenWakeWord pipeline [10], which divides the process of keyword spotting into 3 main blocks, first, in the preprocessing phase, the melspectogram is extracted from the audio using a reimplementation of PyTorch function *torchaudio.transforms.MelSpectrogram* which is a composition of *torchaudio.transforms.Spectrogram()* and *torchaudio.transforms.MelScale()*, second, uses a private implementation of the audio embedding model from Google [6] to extract the audio features, see section 3 for an explanation of how these procedures are done. These embeddings are then classified by a neural network 1 on whether a specific keyword is or is not within the inputted audio.

Our goal is to test an alternative model for keyword classification, specifically attempting a simple logistic regression as a potential low cost and more interpretable alternative to the deep neural networks proposed by the authors of [10]. When analyzing the structure of the "Turn on the lights" dataset, we observed that the positive and negative embeddings show a clear separability, making more linear and simpler methods a viable option for classification.

2 Related Work

As previously mentioned, early keyword spotting systems were based on manually crafted features such as Mel Frequency Cepstral Coefficients (MFCCs), for instance, see [5] where MFCCs were used for Language Identification. These approaches, while effective in controlled environments, failed to generalize in real-world conditions where noise and low quality audio is present. Given the recent advances in the Deep learning field, new approaches, have been developed, providing systems that automatically process raw audio or spectrograms.

Audio2Vec [1] and SincNet [9] are two examples of such approaches making important contributions to the field of audio representation learning. Audio2Vec approaches the task of learning vector representations for audio signals by placing the embeddings of similar audio segments closer to each other while the different ones are mapped to a larger distance from the similar ones in a high dimensional space. This approach is based on word embedding techniques like Word2Vec [7] but is adapted for audio tasks, such as keyword detection. SincNet, on the other hand, is a neural network architecture specifically designed for processing raw audio waveforms. By using parametrized sinc functions, SincNet learns features directly from raw waveforms, which is useful for speaker and speech recognition tasks.

Pretrained audio models, such as those described in [6], allow the extraction of embeddings that capture features from audio signals. These embeddings mask out noise and other specified irrelevant variations, focusing on phonetic patterns for tasks like keyword spotting and wake-word detection. OpenWakeWord uses such pretrained embeddings, which require as input the MelSpectogram of the raw preprocessed audio.

Data augmentation techniques have also been important for the given task, particularly due to the challenges associated with acquiring large datasets that provide sufficient samples per parameter. Small datasets often increase the risk of overfitting, as models will have it easier to model the exact distribution of given data. Methods such as jitter (random positional offsets), noise addition, and volume scaling simulate diverse acoustic environments, making models better at generalizing to noise and variability and less prone to overfitting.

For instance [4] provides a process for synthetic data generation in training models for low-resource scenarios, which is directly applicable to wake-word detection, and the authors of OpenWakeWord take inspiration from it.

3 Methodology

3.1 Dataset

The dataset used in this project consists of positive and negative audio samples. For the positive class 3023 synthetic recordings of the phrase *"turn on the lights"* were obtained from the OpenWakeWord repository which was generated using the techniques detailed in [4] and [8]. A mixture of 3 different datasets was used for the negative samples to capture a wide range of non keyword conditions. In the first place, a subset of 200 samples of the *fma-large* dataset containing music samples to simulate real-world scenarios better, was mixed with a portion of 897 samples from *FSD50K* dataset which contained different types of background noise such

as environmental noises (e.g. street, cars, ...) and the last one were 4999 clips from *Common Voice 11 corpus* containing demonstrations of general speech. All of these samples were mixed to ensure coverage of different environments and speech patterns that do not contain the specific keyword. The distribution of the samples within the negative class can be seen in Figure 1.

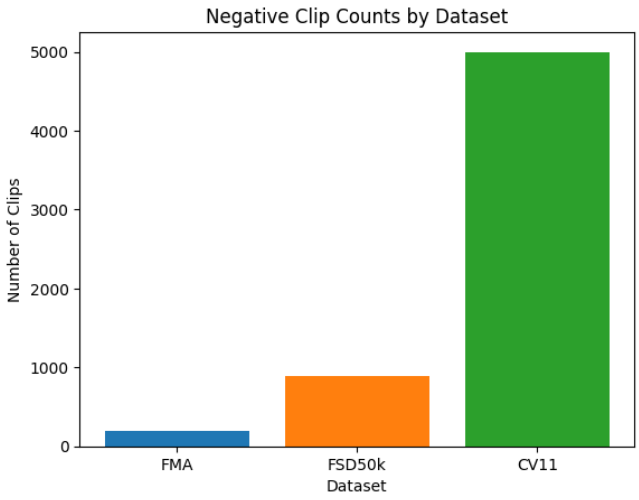


Figure 1: Distribution of the negative samples

3.2 Dataset Preprocessing

The first step is for all recordings to be resampled to a frequency of 16 kHz (16,000 samples per second). In voice processing, a 16 kHz sample rate is frequently employed because other than being computationally economical, it spans the fundamental frequency range of human speech, which is roughly 60 Hz to 8 kHz. The samples are then normalized to a 16-bit PCM (Pulse Code Modulation) format in order to standardize the audio data for processing. PCM samples, which are the output of an Analogue to Digital converter, are binary representations of the immediate amplitude of the analogue signal in a wav file. The sampled audio has a broad amplitude range since they are recorded as arrays of 16-bit integers, spanning the range $[-32767, 32767]$.

The samples after being standardized to 16-bit PCM will be used to extract the *log mel spectogram*. The spectrogram will then be used by the embedding model.

3.2.1 Spectrogram Extraction

OpenWakeWord uses a self ONNX implementation of Torch's melspectrogram function [12] with fixed parameters as to enable efficient performance across devices. ONNX is an open format built to represent machine learning models used to guarantee interoperability between frameworks. The function returns a log mel feature vector which is a 32-dimensional vector that spans the frequency range of 60Hz to 3800Hz and is quantised to 8 bits every 10 ms.

To begin a **pre-emphasis filter** is applied on the 16-bit PCM samples amplifying the high frequencies. The filter can be applied to a signal x via

$$y(t) = x(t) - \alpha x(t-1) \text{ with } \alpha = \{0.95, 0.97\}$$

This filter helps to balance the frequency spectrum and to avoid numerical issues during the Fourier transform operation used later. The signal is then split into short-time frames of 25ms. Given that the signal changes over time, by taking a small enough step, the frequencies will appear stationary. This allows to apply a Fourier transform on the step and, by concatenating adjacent steps (with a 10 ms stride), a good approximation of the frequency contours can be obtained.

To maintain the continuity among all the steps in the signal a **Hamming window function** is applied

$$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right) \text{ with } 0 \leq n \leq N-1$$

where N is the window length. The continuity is vital to counteract the assumptions regarding the infinity of the data made by the Fast Fourier Transform. That is necessary as an N-point FFT is then applied to each frame to obtain the frequency spectrum. The process of splitting the signal into short steps and applying FFT is also known as **Short-Time Fourier-Transform** [11].

The power spectrum (periodogram) of the obtained signal is passed through a bank of triangular filters spaced on the mel scale, which approximates human auditory perception of sound frequencies. The mel scale is a scale of pitches judged by listeners to be in equal distance between one and the other. It is expressed by pitch (Mels) over frequency (Hz) on a logarithmic scale as to to mimic the non-linear human ear perception of sound. That is achieved by being more discriminative at lower frequencies and less discriminative at higher frequencies.

$$Mel(f) = 2595 \log_{10}(1 + \frac{f}{700})$$

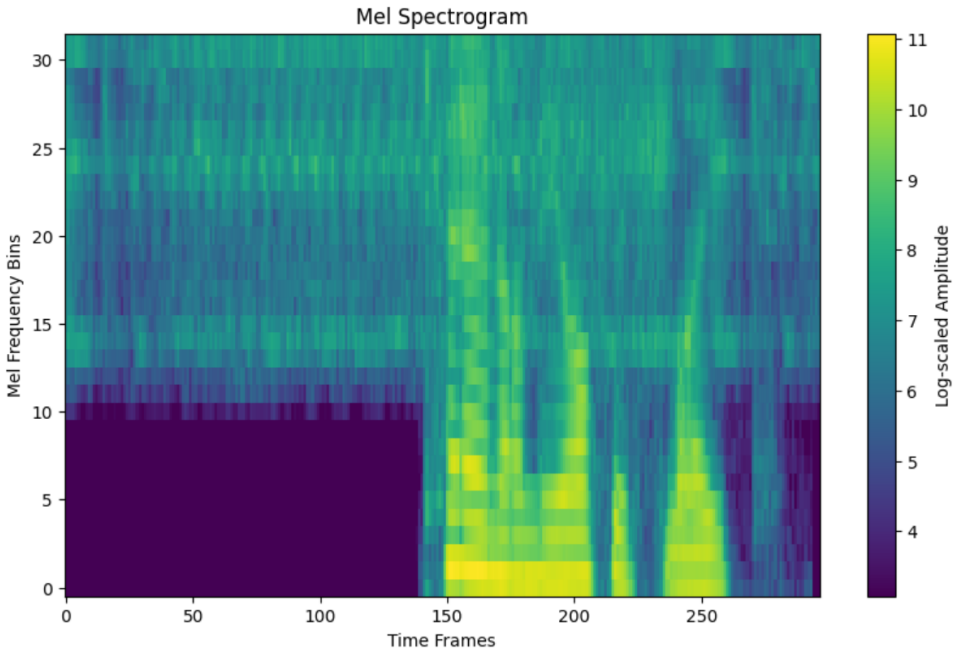


Figure 2: Shows an example of an extracted Mel spectrogram from a given audio stream

3.3 Feature Extraction - Audio embedding extraction

The obtained log mel features are then fed to the embedding model. The embedding model used by OpenWakeWord has been proposed by [6]. It contains 5 convolutional blocks (approximately 330k weights) and a head model that contains a single convolutional block (approximately 55k weights)along with a classification one.
Speech embedding model

1. a 1x3 convolution with 24 channels
2. a 3x1 convolution with 48 channels
3. a maxpool layer
4. a 1x3 convolution with 72 channels
5. a 3x1 convolution with 96 channels

Head model:

1. a 3x1 convolution
2. classification block

A 1×3 convolution has a kernel that is one frequency bin high and three-time steps wide, which allows the model to learn temporal patterns within a single frequency band. On the other hand, a 3×1 convolution is three frequency bins high and one time step wide, capturing spectral variations at a fixed time step. By stacking these two types of convolutions, the architecture captures both time and frequency based patterns. In the last convolutional block, only 3×1 convolutions are used because the frequency dimension has been reduced to 1 by the preceding layers. The 32-dimensional log mel feature vectors are then turn into 96-dimensional feature vector. This operation is repeated every 80 ms.

3.4 Classifier Model Architecture

3.4.1 Provided Neural Network

The architecture of the proposed Fully Connected Network, as available in the repository, is shown in Table 1.

Table 1: Fully Connected Network Architecture

Layer	Type	Details
1	Flatten	Converts input to a 1D vector
2	Linear	Fully connected, $X_{\text{timesteps}} \times X_{\text{features}} \rightarrow 32$
3	LayerNorm	Normalization with 32 features
4	ReLU	Activation function
5	Linear	Fully connected, $32 \rightarrow 32$
6	LayerNorm	Normalization with 32 features
7	ReLU	Activation function
8	Linear	Fully connected, $32 \rightarrow 1$
9	Sigmoid	Outputs probability for binary classification

3.4.2 Ridge Logistic Regression

After analyzing the embeddings obtained from the model, see Figure 4(a) it can be estimated that the positive and negative embeddings happen to be well separated, for this reason, *Ridge Logistic Regression (RLR)* seemed like a reasonable low-cost alternative to the proposed Neural Network. *RLR* was chosen instead of standard Logistic Regression because it introduces L_2 regularization which helps with overfitting when dealing with high dimensional data.

Ridge Logistic Regression introduces an L_2-penalty term, which penalizes large coefficients by adding a squared magnitude constraint.

The benefits of using Ridge Logistic Regression over standard logistic regression include, as previously mentioned, reducing the risk of overfitting by preventing large coefficient values and helping with numerical stability by controlling the variance of estimated parameters, especially when the number of features is large compared to the number of samples as detailed in [3].

4 Experiments

4.1 Setup

The model is trained on both negative and positive features and final metrics are averaged over a 5-fold Stratified Cross-Validation. Then we minimize the binary cross-entropy loss, optimize using Adam, and set the learning rate to 1e-3.

Principal Component Analysis (PCA) was also applied to the training audio embeddings in a later step in order to visualize and assess the linearity of the embeddings. We then trained the proposed deep neural network alongside a linear ridge logistic regressor, following the same cross-validation process.

All training was performed on an Intel Core i7-7500U CPU.

4.2 Results

The proposed model achieves an average validation F1-score of 98.31% over 100 epochs. Figure 3 shows the loss and F1-score trends for the training and validation sets over 100 epochs. The model converges extremely fast.

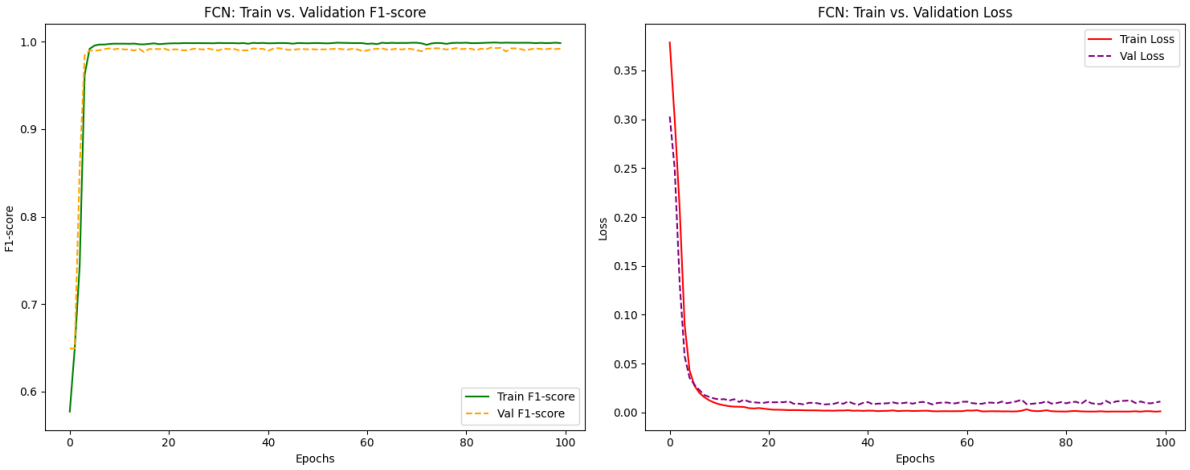
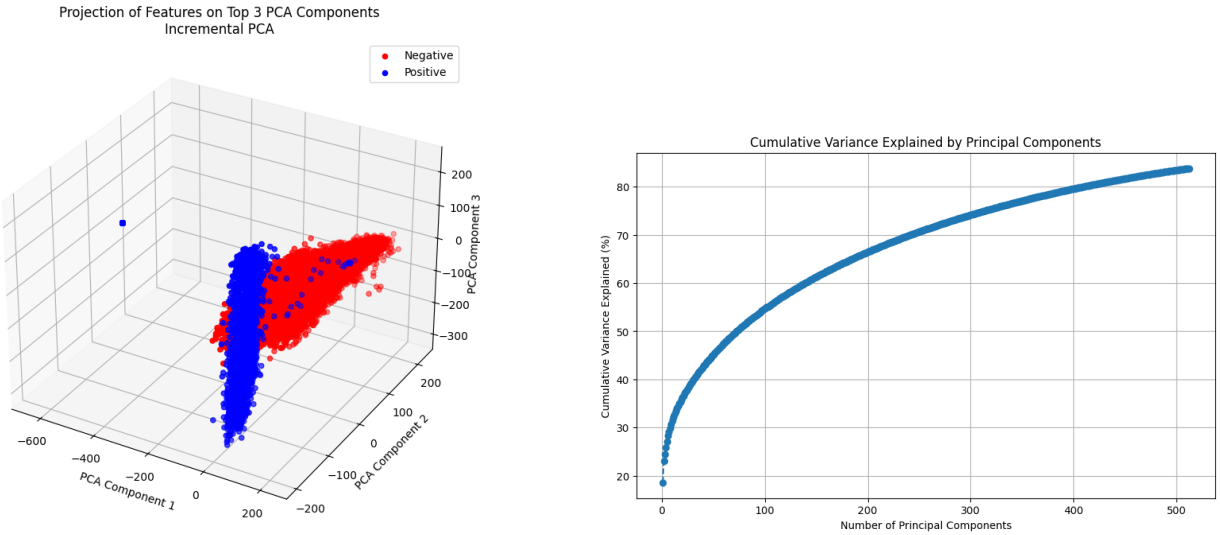


Figure 3: F1-score for training and validation

Additionally, we visualize three principal components in Figure 4 for all the training audio embeddings. A significant proportion of the data appears to be easily separable by eye, so we also train 5 logistic regressors for each fold's split and average the results. In fact, a deep neural network may be unnecessary in our case, as logistic regression performs extremely well, likely because the embedding model extracts linearly separable features. It is reasonable to consider that the task may not require deep learning, and a more feasible, lightweight, and interpretable solution could be explored. All the results are displayed in Table 2.



(a) Incremental PCA visualization

(b) Cumulative variance explained

Figure 4: PCA analysis of the training dataset.

Note: No final testing score is reported for any of the models, as this would ideally require a three-way split and therefore a third test set, and we mainly focus on comparing the two methods. Thus, additional data is needed for testing.

Table 2: Final Cross-Validation Results

Method	Configuration	Mean Accuracy ± Std	Mean F1-score ± Std
FCN	-	0.9947 ± 0.0012	0.9831 ± 0.0024
PCA + Ridge	10 PCs	0.9596 ± 0.0069	0.8724 ± 0.0212
PCA + Ridge	50 PCs	0.9960 ± 0.0014	0.9888 ± 0.0036

[0]*After extracting new features by performing PCA in Audio Embeddings.

5 Conclusion and Future Work

Again, it is very possible that what makes the difference in this pipeline is the audio embeddings model. The majority of the extracted features are well separable for the two classes, enabling keyword/keyphrase spotting with remarkably high performance.

Nonetheless, it should be noted that in future work larger datasets must be utilized, not only for the positive class but especially for the negative one. Additionally, introducing noise into the positive dataset could help simulate real-world speech conditions.

Exploring other keywords for training the model could also be beneficial. Several experiments could take place in order to compare different approaches.

We want to specifically focus on the use of the deep fully connected network. It might be valuable for larger and more complicated datasets. However, future research should focus on comparing complex models with more linear ones to determine the best classifier.

References

[1] Yu-An Chung, Chao-Chung Wu, Chia-Hao Shen, Hung-Yi Lee, and Lin-Shan Lee. Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder, 2016.

[2] Lukas Grasse, Sylvain J. Boutros, and Matthew S. Tata. Speech interaction to control a hands-free delivery robot for high-risk health care scenarios. *Frontiers in Robotics and AI*, 8, 2021.

[3] Herbert Jaeger. Machine learning (wmai010-05) lecture notes.

[4] Jaehyeon Kim, Jungil Kong, and Juhee Son. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech, 2021.

[5] Shashidhar G. Koolagudi, Deepika Rastogi, and K. Sreenivasa Rao. Identification of language using mel-frequency cepstral coefficients (mfcc). *Procedia Engineering*, 38:3391–3398, 2012.

[6] James Lin, Kevin Kilgour, Dominik Roblek, and Matthew Sharifi. Training keyword spotters with limited and synthesized speech data, 2020.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[8] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis, 2018.

[9] Mirco Ravanelli and Yoshua Bengio. Speaker recognition from raw waveform with sincnet, 2019.

[10] Daniel Scripka. Openwakeword: Open source wake word detection, 2023.

[11] Ervin Sejdić, Igor Djurović, and Jin Jiang. Time–frequency feature representation using energy concentration: An overview of recent advances. *Digital signal processing*, 19(1):153–183, 2009.

[12] PyTorch Team. torchaudio.transforms.melspectrogram.