TurboCRNN: A Robust and Efficient Architecture for Wake Word Detection

Nitin Nataraj · Tejeswar A Ramachandran · Dhairya Lalwani · Mohanaprasad Kothandaraman

Abstract

Voice assistants are an integral part of our daily lives, and it is predominant where a significant problem persists, i.e., false triggering. Voice assistants often get triggered by phrases similar to their wake words. Prior research focuses on training CRNN (Convolutional Recurrent Neural Networks) models with Mel Frequency Cepstral Coefficients (MFCCs) or Log Filter Bank Energies (LFBEs) to improve accuracy and reduce either false accepts or false rejects. The existing research addresses only one aspect of the three significant problems, i.e., one among wrong predictions, model complexity, and latency. It does not provide an optimized solution to resolve all three issues. This paper proposes a novel wake-word detection model using Turbo-CRNN with Linear Frequency Cepstral Coefficients (LFCCs) to minimize the above problems' adverse effects. This paper experimented with multi-head attention, regularizers, and optimizers to determine the best possible model for this application to minimize false positives and negatives. This proposed custom CRNN model produced an increase in accuracy, low model complexity (87k parameters), significantly lower false predictions, and outputs very low latency (around 80ms) compared to the CNN, DNN, and TinyCRNN baselines.

Keywords Wake Word Detection · Linear Frequency Cepstral Coefficients · Convolutional Recurrent Neural Networks · False positives · False negatives · Multi-head Attention · Latency.

Nitin Nataraj
nitin.nataraj2019@vitstudent.ac.in

Tejeswar A Ramachandran
tejeswara.ramachandran2019@vitstudent.ac.in

Dhairya Lalwani
dhairya.lalwani2019@vitstudent.ac.in

Mohanaprasad Kothandaraman
kmohanaprasad@vit.ac.in

School of Electronics Engineering, Vellore Institute of Technology, Chennai, Kelambakkam-Vandalur Road, Tamil Nadu, India

1 Introduction

Voice assistants like Google Assistant, Amazon Alexa, Siri, etc., have become our go-to for news updates, cooking recipes, and other daily activities. Despite their wide applications, the problem of false triggering is still prevalent in the industry. Words similar to the wake word activate the voice assistant, called False positives or False accepts. Also, in some cases, the right wake word uttered by the user goes unrecognized by the voice assistant. These are called False negatives or False rejects. A wide range of research has been conducted in automatic speech recognition, keyword spotting, and wake word detection. Starting from the rudimentary models, we have arrived at highly efficient models with low latencies.

During the early stages of research in this domain, Hidden Markov Models were used in a primitive implementation of keyword spotting [1]. The HMM baseline model was also discussed in the training of acoustic models and techniques for handling non-keyword speech and linear channel effects. After the advent of Neural Networks, the possibilities of their use in Wake Word Detection was explored in subsequent research. Deep Neural Network Acoustic Models were trained on Log-mel Filter bank Energy features [2]. A multiclass classification model with low miss rates was developed successfully. The problems and inconsistencies of Automatic Speech Recognition were addressed with a new architecture that integrates Data Augmentation and ensemble learning based on DNN and HMM models, into a single framework [3]. This incorporated a novel keyword rescoring method that uses similarity scores for decision-making. After introducing Convolutional Neural Networks, an approach for detecting the start and end points of the wake word was proposed [4]. A CNN model trained on the Log-mel Filterbank energy features is used to predict the posterior probabilities of detecting the wake word in a given audio fragment. Recurrent Neural Networks are the most common choice for time series analysis. The modified LSTM architecture uses an RNN as a forget gate, which improves model performance by facilitating efficient usage of model parameters [5]. The model was trained on the English Digit dataset, and The model was trained on the English Digit dataset, and the accuracies are relatively higher when compared to the proposed baseline models. Relatively higher accuracy was obtained compared to the proposed baseline models on training the model on the English Digit dataset.

The introduction of attention mechanisms allowed models to focus on specific audio sections that are more relevant to the application at hand. Scaled dot product and Multi-head attentions were the two significant techniques proposed for calculating attention scores using the query, key, and value vectors obtained from audio samples [6]. Self-attention was introduced, where attention scores were calculated for different sections of the same audio sample [7]. The input audio samples were converted into spectrogram images and fed into a custom VGG model based on CNN architecture. Post-down sampling by the VGG model feeds the output into the multi-head attention model, followed by the softmax layer. A series of innovations, starting from simple HMM models, followed by DNNs, CNNs, and RNNs, have paved the way for constant enhancements in model performance for wake word detection. Following this, the CRNN architecture is now being adopted for performing wake word detection. In the CRA architecture (Convolutional-Recurrent-Attention architecture), the convolutional front end directs its outputs toward the Recurrent layers with an attention model [8]. The False Alarm rates show significant reductions on deploying the CRA architecture.

Additionally, research has been conducted on data preprocessing, optimizers, and feature extraction techniques for wake word detection. Since wake word detection involves real-time audio inputs, denoising is essential. Various methods have been employed for denoising audio data. Wavelet decomposition is a technique used in signal processing to break down a signal into its frequency components, also known as wavelets. It entails evaluating a signal or data at multiple resolutions by employing a sequence of wavelet functions. Every wavelet function links to a distinct frequency band, and the decomposition process generates coefficients that signify the magnitude of the signal in each frequency band. Wavelet Decomposition is used to analyze and compress audio, image, and video signals for tasks like signal denoising, data compression, and time-frequency signal analysis [9]. Deploying a combination of wavelet decomposition and deep learning techniques enhances the process of noise removal from speech recordings [10]. The frequency components of noisy and clean speech signals are analyzed. A deep learning algorithm is trained on the amplitude spectra of wavelet-decomposition vectors of respective signals as the predictor and target to remove the noise and improve speech quality. This method is effective across all frequency ranges and can significantly enhance the quality of speech recordings.

Data augmentation is a method commonly utilized in machine learning to enhance the dataset size by creating altered versions of the pre-existing data. This is achieved by executing a set of transformations on the initial data, like introducing noise or changing the pitch of an audio recording. The primary objective of data augmentation is to increase the amount and diversity of data, thus diminishing the possibility of overfitting, a common problem encountered when a model undergoes training on an inadequate amount of data. Different augmentation sets have varying effects on each sound class [11]. This suggests that employing class-specific data augmentation could further improve model performance.

MFCCs are the most commonly used feature extraction technique in speech processing and analysis due to their effectiveness in capturing important spectral information in speech signals while being relatively robust to noise and other distortions. Including Frequency Band Energy (FBE) information in the MFCC calculation, referred to as FBE-MFCC, can be beneficial in both high-quality and adverse speech environments [12]. Putting frame energy (FE) and FBE information into the MFCC should be treated differently for different applications, as the performance may vary. The CNN architecture was applied to the problem of tonal speech signal recognition of Gurbani hymns [13]. The CNN model was combined with the PRAAT technique for speech segmentation and used the MFCC features as input that returned text as output. The proposed method performed better than conventional approaches, with increased accuracy and a minimal Word Error Rate (WER) for continuous and extensive vocabulary sentences of speech signals with different tones.

LFCCs (Linear Frequency Cepstral Coefficients) are obtained by applying a linear filter bank instead of a mel filter bank to the power spectrum of a speech signal. The first few coefficients tend to represent the overall spectral shape of the signal, while the latter captures the temporal dynamics of the signal. A study explores various filterbank spacings and phonetic regions to extract cepstral features from telephone speech while filtering out speaker discriminative frequency regions. The results show that LFCCs outperform MFCCs in speaker recognition systems that utilize nasal and non-nasal consonant broad phonetic regions. The findings are consistent with the filterbank energy f-ratio plots within the telephone bandwidth. Additionally, Antimel Frequency Cepstral Coefficients are recognized to help enhance the overall system performance and contribute to a combined system that improves the equal error rate (EER) by 17.3% relative to the baseline system [14]. The performance of MFCC and LFCC is compared in speaker recognition systems. It is based on two state-of-art back-end systems in the NIST SRE 2010 extended-core task. LFCC consistently outperforms MFCC, particularly in female trials, due to its better ability to capture spectral characteristics in the high-frequency region, which directly relates to the length of the vocal tract [15].

Optimizers are algorithms that adjust the parameters of machine learning models during training to minimize the loss function, resulting in the best performance of the model on a given task. A simple and efficient algorithm called Adaptive Moment Estimation (Adam) has been presented for optimizing stochastic objective functions using gradient-based methods in machine learning problems with large datasets and/or high-dimensional parameter spaces. The technique combines the benefits of two optimization methods: AdaGrad for dealing with sparse gradients and RMSProp for dealing with non-stationary objectives. The algorithm is easy to implement and requires minimal memory. Adam optimization method is robust and suitable for various non-convex optimization problems in machine learning [16]. Adding Nesterov momentum to Adam (Nadam) significantly improved optimizing stochastic objective functions [17]. While it is not always the best algorithm, using Nesterov momentum with RMSProp allows for optimal performance in training machine learning models. The fixed-sized window of past gradients used to scale the gradient updates leads to suboptimal convergence in Adagrad. The AMSGrad algorithm was created by modifying Adagrad and giving it a long-term memory of past gradients. [18].

Apart from the innovations in data preprocessing, feature extraction techniques, and optimizers, various improvements introduced in the model architectures for wake word detection reduce false triggering, latency, and the overall model size. These were done to create a high-performing model deployable on an embedded device with moderate computational abilities. Edge CRNN, a lightweight model deployable on an edge device like Raspberry Pi, was designed for keyword spotting [19]. A depth-wise separable convolutional and residual structure has been proposed [19] to reduce the number of Floating Point Operations per second (FLOPs) and the number of parameters while retaining high accuracy. A depth-wise separable CNN-based classifier was designed and trained on Mel Frequency Spectral Coefficients (MFSC) [20]. The model produced a high hit rate with low false alarm rates. The model produced a high hit rate with low false alarm rates. The effect of model quantization has also been explored. It is observed that the 8-bit quantized model consumed very low memory with a minimal reduction in accuracy on deployment in an ARM Cortex M4 processor. The application of Densely connected grouped Convolutional Networks with the presence of squeeze-and-excitation networks in keyword spotting was explored. [21] The first model achieved an accuracy of 96.3% with a parameter size of 122.63K trainable parameters. A comparison of the performance of an Ultra-Low-Power custom CNN model trained on raw audio and MFCCs was presented in terms of energy consumption and latency. The model was deployed on a MAX78000 NN accelerator and an ARM Cortex M4 processor. More energy is consumed while training the model on MFCCs when compared to training them directly on audio data [22]. The performances of transfer learning and knowledge distillation techniques are compared using a low-power low footprint wake word detection model [23]. Knowledge distillation displays better performance for wake word detection, and on performing phone-aligned training, the model latencies show a significant reduction. A TinyCRNN model architecture was proposed to reduce the number of false accepts and model parameters while exhibiting high model performance. The scaled dot product Attention model was also introduced in the TinyCRNN model [24], which further reduced the false accept count concerning the CNN and DNN baseline models. Several Automatic Speech Recognition (ASR) models and wake word detection models were proposed in earlier research works.

Our research focuses on creating a wake-word detection model that combines previous research works' merits. A collective performance measure of false positives and negatives is essential to evaluate a model's performance. For edge applications, the model speed is crucial. Therefore, latency is an equally important metric. The proposed TurboCRNN architecture aims to solve the critical problem of reducing false predictions and prediction latency while retaining high accuracy relative to the TinyCRNN [24].

The Google speech commands dataset has been used in this research [25], and the word 'Marvin' has been chosen as the wake word. The structure of the paper is as follows. In section 2, we explore the various neural network architectures, feature extraction techniques, etc., used in previous research on wake word detection. Section 3 explains the different methods implemented in our proposed methodology. Section 4 presents the various experiments performed using the techniques described in Section 3 for applying wake word detection. Following this, Section 5 provides a thorough analysis of the results obtained from the experiments performed. Based on these results, the best-performing models were chosen, and a trade-off has been presented regarding the optimal solution for wake word detection.

2 Wake Word Detection Techniques

Wake word detection identifies a specific word or phrase that acts as a trigger to activate a voice assistant or other automated system. The general process involves preprocessing, extracting features from audio samples, and training Machine Learning Models (usually Neural Networks) using the same to detect the presence of the Wake Word. These algorithms analyze audio signals, looking for specific patterns and features that indicate the presence of the wake word.

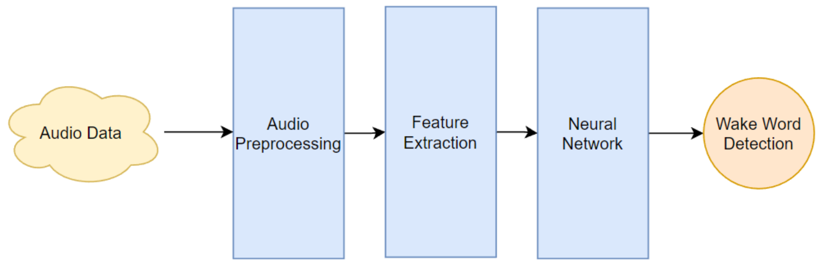

Fig. 1 General Process Flow of Wake Word Detection

In this section, we elaborate on the existing research involving techniques used for wake word detection. We discuss the Mel Frequency Cepstral Coefficients, which are the most commonly used features in the domain of speech recognition. Then dive into various Neural Network architectures used in Wake Word Detection, such as Deep Neural Networks, Convolutional Neural Networks, and different Recurrent Neural Network architectures, such as the LSTM and GRU. We then move to Loss Functions and how Optimizers optimize these functions. And finally, end with the importance and impact of Attention and Regularization Techniques.

2.1 Feature Extraction Techniques

Audio data can be directly fed into neural network models as *.wav* files for training [2]. Raw Audio files contain much information that isn't directly relevant to wake word detection. The neural network model can focus on the signal's most essential features/aspects now related to the wake word of interest by extracting relevant features from the audio signal.

Mel Frequency Cepstral Coefficients (MFCCs) are the most common choice for a variety of speech recognition tasks [15] [13] [12]. MFCCs are coefficients obtained by mapping the signal's spectrum to the frequency bands in the mel-scale. The mel-scale is a pitch scale that approximates how humans perceive sound—performing Direct Cosine Transform (DCT) on the mel-frequency spectrum results in Cepstral coefficients. Finally, the entire spectral envelope of the signal is represented with reduced dimensionality, portraying only the critical information and dominant features [12]. A drawback with MFCCs is that they are prone to distortions in a noisy environment due to their dependency on a logarithmic frequency scale.

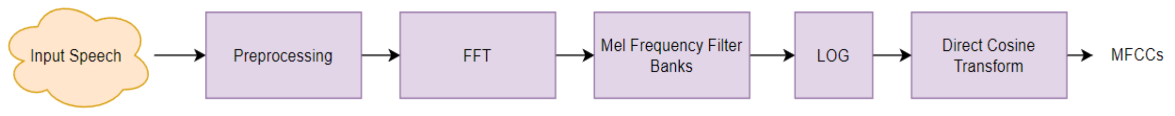

Fig. 2 Generation of MFCCs from Audio samples

Generally, the audio samples provided as input to neural networks are recorded at a specific sampling rate in varied environments. These samples are prone to external noises and disturbances. To make wake word recognition models more robust, it is imperative to eliminate noise and ensure that the audio samples are of comparable quality [10]. The goal is to retain helpful information and remove or reduce the noise components. Numerous methods, such as spectral subtraction, wavelet-based denoising, noise reduction algorithms, and deep learning-based denoising, may denoise audio signals. The ideal denoising technique is selected based on the nature and characteristics of the noise, and the specific needs of the application.

2.2 Neural Network Architecture

Neural networks are machine-learning models that mimic the structure and behavior of the human brain. They consist of layers of interconnected nodes, known as neurons, which process information by receiving input from other neurons and applying an activation function to generate an output that is passed to the next layer. Neural Networks find use in a myriad of applications. These include image and speech recognition, natural language processing, autonomous driving, etc. Several types of neural networks exist with their unique architectures and applications

2.2.1 Deep Neural Network

A Deep Neural Network (DNN) is a variant of an Artificial Neural Network (ANN) with multiple layers between the input and output layers. These layers usually consist of densely connected neurons and have demonstrated their effectiveness in tasks that involve significant amounts of data and intricate decision-making processes. DNNs can be trained on audio features or direct audio samples for performing audio classification [2].

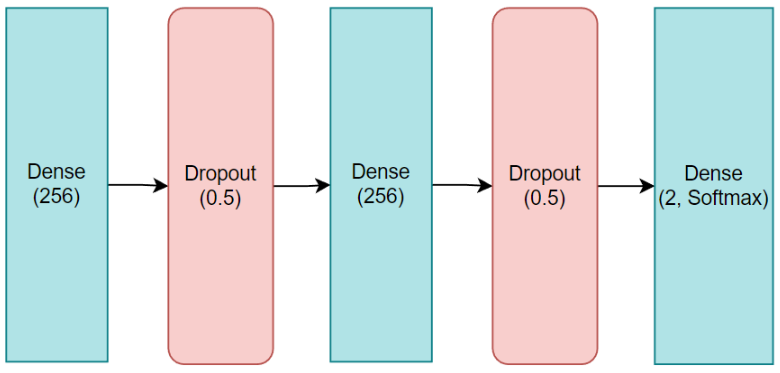

Fig. 3 Deep Neural Network

2.2.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep neural network that uses convolutional layers that apply filters to the input image, extracting features like edges or textures and creating feature maps highlighting these features. These feature maps are passed to fully connected layers, which classify the input image. CNNs in speech recognition are also used to detect the audio alignment inside the input window and the start and end points of the wake word [4].

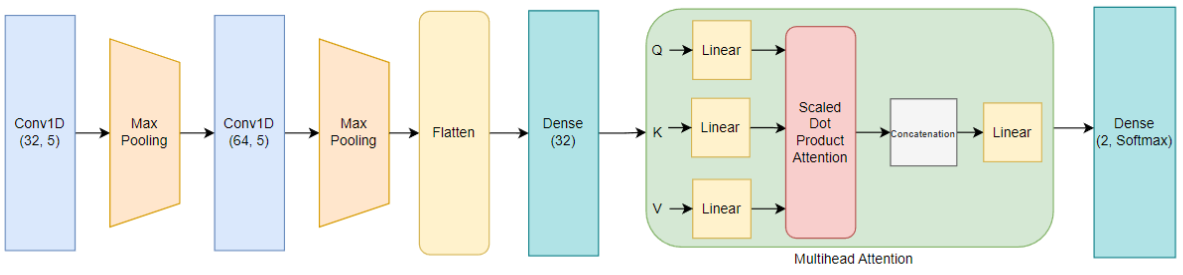

Fig. 4 Convolutional Neural Network

It is imperative to use an architecture that can capture temporal dependencies. A Recurrent Neural Network (RNN) is a deep neural network that captures the temporal dependencies between sequence sections using feedback connections. These feedback connections enable the network to maintain an internal state summarizing the information it has seen.

2.2.3 Recurrent Neural Networks

In traditional Recurrent Neural Networks (RNNs), during backpropagation, the gradient of the error function for the weights can become very small when multiplied repeatedly across many time steps. This results in the vanishing gradient problem. It causes the weights to be updated very slowly or not at all, making it difficult for the network to learn long-term dependencies in sequential data. As a result, the network struggles to accurately predict future values or make sense of complex patterns in the input data. Specific RNN architectures such as LSTM and GRU are used to solve the vanishing gradient problem. LSTM stands for Long Short-Term Memory and GRU stands for Gated Recurrent Unit.

The LSTM architecture has memory cells and gates that regulate the information flow in and out of the cells. An LSTM architecture with an additional recurrent layer acting as the forget gate [5] allows the network to efficiently process continuous input streams and effectively utilize the model parameters. Each memory cell in an LSTM network comprises three key elements: an input gate, a forget gate, and an output gate. The input gate determines the amount of new input that should enter the memory cell, while the forget gate decides how much of the last memory should be forgotten. Finally, the output gate determines how much memory should be output to the next network layer. An LSTM unit mainly consists of the sigmoid activation function (σ) and the tanh activation function, as shown in Figure 5.

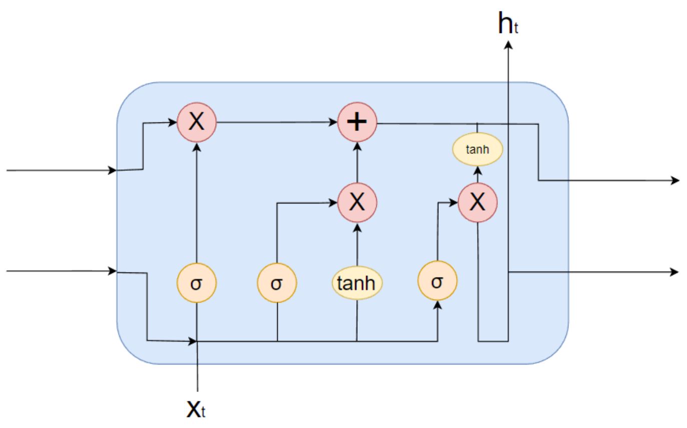

Fig. 5 Long Short-Term Memory Network (LSTM)

GRU is similar to LSTM, but it has fewer computations and parameters. It involves a network of sigmoid (σ) and tanh activation functions as shown in Figure 6. It has two gates: the reset gate and the update gate. The reset gate controls how much past information is forgotten, while the update gate determines how much current input is added to the current state. Therefore, GRU can learn short-term and long-term dependencies in sequential data making it an apt choice for analyzing sequential data for applications like Music Generation.

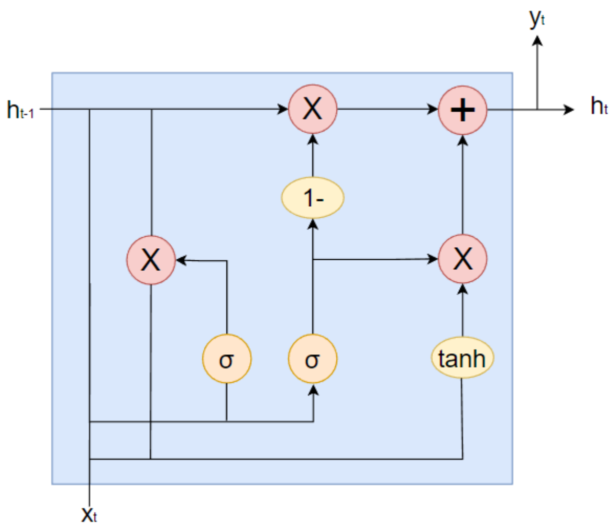

Fig. 6 Gated Recurrent Unit (GRU)

As already stated, CNNs are good at processing local spatial features in input, such as edges or textures, but they cannot capture temporal dependencies or context information. On the other hand, RNNs are designed to handle sequential data and capture long-term dependencies or context information, but they may not be as effective in processing spatial features.

By combining both architectures, Convolutional Recurrent Neural Networks (CRNNs) can capture local spatial features and temporal dependencies, making them well-suited for tasks requiring both, such as speech recognition [24].

2.3 Loss Functions and Optimizers

A loss function, also known as a cost or objective function, is a mathematical function that measures the difference between the predicted and actual output for a given input. They provide a quantitative measure of performance. Wake word detection is a classification problem. A wake word exists in a given audio or doesn't, so it is a binary classification problem.

Binary cross-entropy, categorical cross-entropy, and sparse categorical cross-entropy are all loss functions commonly used in deep learning for classification tasks. Binary cross entropy is used for binary classification, while absolute cross entropy and sparse categorical cross entropy are used for multiclass classification. When the true labels are one-hot encoded vectors, Categorical cross-entropy is used. On the other hand, when the true labels are integers, Sparse categorical entropy is used.

A neural network adjusts its parameters (such as weights and biases) to minimize the value of this loss function. This process is called optimization, often done using optimizer algorithms. Several algorithms exist for classification problems, such as Stochastic Gradient Descent (SGD), Adam, etc. SGD is a simple and popular optimizer that updates the model parameters based on the gradient of the loss function with respect to the parameters. Adam calculates an exponential moving average of the past gradients and their squares to adjust the learning rate for each parameter [16]. NAdam is an extended version of Adam optimizer, including Nesterov Momentum [17]. The deployment of loss functions and optimizers is done in the model compilation stage.

2.4 Attention Models & Regularizers

Attention models allow neural network models to selectively attend to certain parts of the input data that are most relevant for making predictions rather than treating all parts equally. Scaled dot product attention improves the performance of CRNN models [24]. The scaled dot product attention model is a powerful and efficient mechanism for modeling dependencies between elements in a sequence.

Here, the information the model is trying to retrieve from the audio input is represented by the query vector, and the information used to compute attention scores are represented by the key vector.

$$\text{Attention}\ (Q, K, V) = \text{softmax}(QK^T/\sqrt{d_K})\ V \tag{1}$$

The scaled-dot product attention mechanism computes an attention score for each element in the audio sample. As shown in equation (1), where Q, K, and V stand for Query, Key, and Value vectors, the attention scores are computed by calculating the dot product of the query and key vectors and dividing them by the dimensionality of the key vector. The final score is then passed to the softmax function to obtain the probability distribution [6].

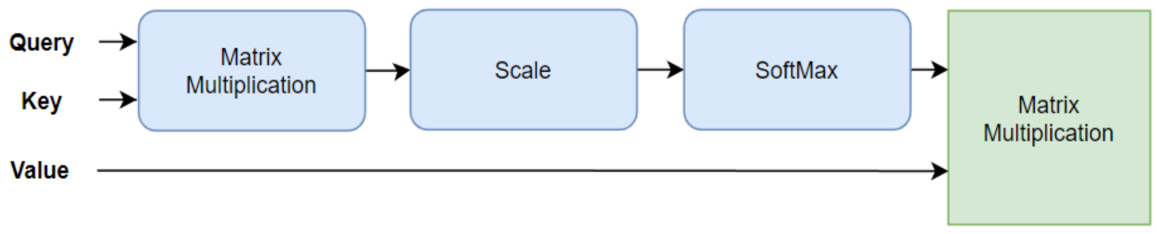

Fig. 7 Scaled Dot Product Attention Model

It is possible that the model becomes too complex and begins to fit to noise in the training data rather than the underlying patterns. To prevent overfitting and generalize the model, we use regularization. A penalty term is added to the loss function, hence allowing the minimization of losses during training. This penalty term discourages the model from overfitting by either constraining the magnitude of the model weights or adding a complexity cost.

The most simple form of regularization is called Early Stopping. This monitors the performance of a model on a validation dataset and stops training when the performance begins to deteriorate. In some cases, model performance may degrade for a few epochs and then gradually climb and cross the initial best accuracy. But the Early Stopping regularizer would stop the model training before it begins to rise again; this is a serious demerit. Another regularizer, called L1 or Lasso Regularization, adds a penalty proportional to the absolute value of the weights, but it leads to sparse solutions. Dropout regularizers prevent over-dependence on a single feature. It randomly drops out (sets to zero) a proportion of the activations in a layer during training.

3 Proposed Methodology

Despite the innovations, some shortcomings persist with the existing techniques. The scope for improving accuracy and reducing false predictions and model latency still exists. Our proposed wake word detection technique aims to improve overall model performance based on the aforementioned parameters by experimenting with data augmentation and preprocessing, feature extraction techniques, different model architectures, and attention models.

3.1 Data Preprocessing and Augmentation

As mentioned earlier, the audio inputs provided to the neural network might be noise-prone; hence, denoising is essential. In this research, we denoise the audio samples using Wavelet Decomposition. Wavelet decomposition is a technique employed in signal processing to break down a signal into its frequency components, also known as wavelets. Wavelet Decomposition is used to analyze and compress audio, image, and video signals for tasks like signal denoising, data compression, and time-frequency signal analysis [9].

Mathematical functions that have a finite duration and oscillate between zero and one are used as wavelet functions for wavelet decomposition. The signal is broken down into approximations and details at varying resolutions to perform wavelet decomposition. The approximations indicate the general trend of the signal, while the particulars represent its high-frequency components [9]. The process can be iterated to get additional levels of detail. Some commonly used wavelet functions are Haar, Daubechies, Coiflet, and Symlet. Each family of wavelets makes varying trade-offs based on the compactness of its basis function, localization in space, and smoothness [9]. Haar wavelets are composed of two coefficients and are considered the simplest wavelets. They are discontinuous and valuable for fundamental signal analysis and noise reduction tasks. On the other hand, Daubechies wavelets are more complex and have a larger number of coefficients. They are available in different sizes and come with varying numbers of vanishing moments. They are considered smooth and beneficial for data compression and denoising. Symlets are comparable to Daubechies wavelets but have additional symmetry properties. They are efficient at capturing both high and low-frequency components of a signal. Lastly, Coiflets are similar to Daubechies wavelets but are more symmetric and have a higher number of vanishing moments. They are advantageous for analyzing signals with sharp changes or singularities and are thus suitable for image processing applications.

In this research, we employ the ‘db4’ wavelet for denoising the audio samples. The 'db4' wavelet, which belongs to the Daubechies family of wavelets, is orthogonal, allowing it to be used for signal reconstruction without introducing distortion. The wavelet function for db4 is defined by four coefficients, making it a four-term wavelet. These coefficients determine the wavelet's shape and its properties, reflecting its ability to capture and represent different signal types. Moreover, it has high smoothness, making it ideal for analyzing signals with gradual and abrupt changes. Additionally, the db4 wavelet can accurately represent polynomial signals up to the third order, as it has four vanishing moments.

Despite the even distribution of data, the available data would not be sufficient to generalize the essential features during the training phase required for proper wake word detection of the proposed models. Hence, performing data augmentation after denoising generates more data for training the model.

3.2 Feature Extraction: Linear Frequency Cepstral Coefficients

For wake word detection, MFCCs are the most common choice of feature extraction technique, and there is very little research on the performance of LFCCs for this application. MFCCs and LFCCs capture important spectral features of the audio signal that can identify different phonetics and classify the signal.

Linear Frequency Cepstral Coefficients (LFCCs) use a linear filter bank on the power spectrum, and cepstral coefficients are calculated using the discrete cosine transform of the power spectrum. Linear Scale Filterbanks emphasize high-frequency components of the spectrogram by employing more filter banks in the high-frequency regions [14]. According to prior research, LFCCs provide some advantages over MFCCs in Speaker Recognition [14], are as robust as MFCCs to babble noise, and are more vital to reverberations [15].

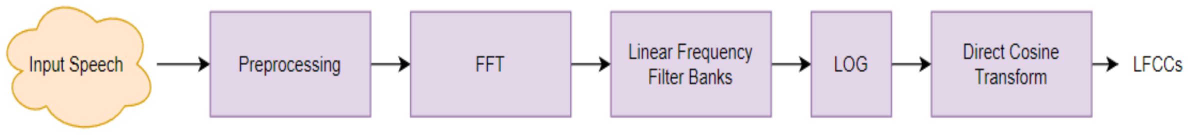

Fig. 8 Generation of LFCCs from Audio samples

In this research, we expand on this idea and check whether LFCCs provide an advantage even in Wake Word Detection. Hence, we present a comparison of model performance based on the feature extraction method (MFCC vs LFCC) in a later section of the paper.

3.3 Custom CRNN Model

Much research has been conducted on the standalone implementations of CNNs and RNNs for wake word detection and speech recognition tasks [4] [13]. A few have used their combination to yield better results. Along those lines, this research focuses on experimenting with Convolutional Recurrent Neural Networks (CRNNs), a variety of Convolutional and Recurrent Neural Networks. Prior research on wake word detection using CRNNs involves the usage of multiple convolutional layers followed by a single LSTM or GRU layer [24] [8] [19]. The convolutional layers in the CRNN architecture are used as feature extractors that capture local patterns and structure in the input data. The receptive field of the output obtained from the convolutional layers contains only the dominant features, and the models are trained on these dominant features obtained from the convolutional front end, whose outputs are then processed by recurrent layers for sequence modeling and prediction [24]. Hence, the training process is expected to be faster in these recurrent layers when compared to their standalone baseline implementations. Two custom variants of the CRNN architecture created for experimentation are shown in Figures 9 and 10.

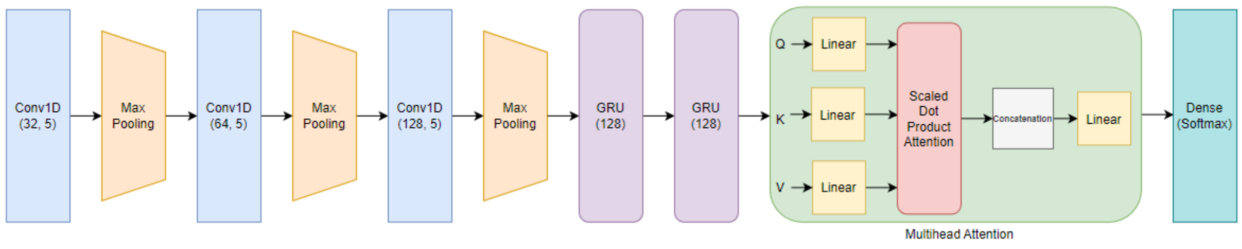

Fig. 9 Proposed Custom CRNN_GRU_GRU Model Architecture

The first custom CRNN model constructed for this research uses a combination of Convolutional Layers with two GRU layers, as shown in Figure 9. Multiple convolutional layers with increasing filter sizes, each followed by max-pooling layers were added to the architecture. The Convolutional layers use a kernel of size five and a stride length of one. The ReLU activation function used with these Convolutional layers controls the activation of neurons and allows for more effective and faster training. The kernel keeps moving to the right based on the fixed stride length until it traverses the entire audio feature array. Following each 1D Convolutional layer, a Max pooling layer performs dimensionality reduction by extracting the dominant features from each portion of the audio features covered by the kernel. This is followed by the recurrent section consisting of two Gated Recurrent Units (GRUs). The recurrent GRU layers are followed by an attention model and a Dense layer with two units and softmax activation for detecting the presence of wake word.

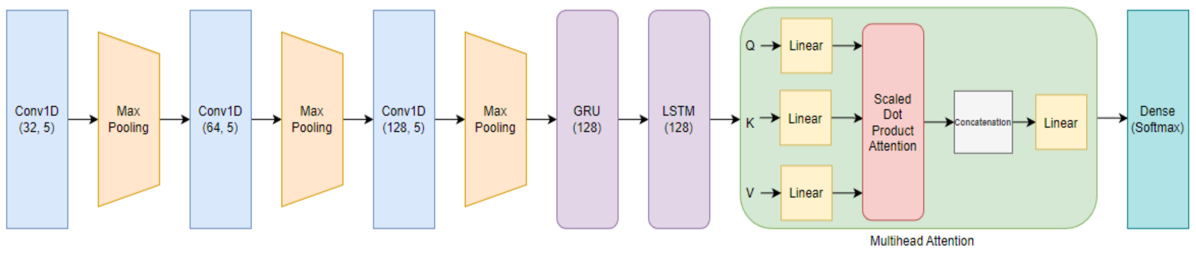

Fig. 10 Proposed Custom CRNN_GRU_LSTM Model Architecture

Another custom CRNN model constructed by replacing the second GRU layer (in the previous model) with a Long-Short term memory (LSTM) recurrent layer, is shown in Figure 10. Otherwise, the overall model architecture remains the same. Experiments were conducted on the model architecture to examine the possibilities for enhancing model performance.

In the case of Model Optimizers, Adam and RMSProp optimizers exhibit slow convergence and poor generalization over non-convex optimization problems with sparse gradients. The AMSGrad optimizer, a modified version of the Adam optimizer [18], prevents overshooting of minimum by using the maximum of past squared gradients for learning rate updation. The AMSGrad optimizer was chosen as the optimizer for the proposed wake word detection system. Following this, experiments were conducted on the proposed custom CRNN model's performances after the introduction of attention models.

3.4 Attention Model

Attention models assign weights to each section of the input, indicating its importance. This enables the model to focus on specific areas of the audio which are more relevant to the application at hand. For this research, multi-head attention was introduced in the designed custom CRNN models with an expectation of performance enhancement.

Multi-head attention is an attention mechanism that analyzes multiple parts of the data in parallel to identify the essential data sections and capture dominant and relevant features [6]. Linear Transformations are performed on the queries, values, and keys obtained from the audio input and projected onto multiple subspaces or 'heads'. These projections are split into various heads with reduced dimensionality. The number of heads is dependent on the model architecture and the application.

$$\text{MultiHead}\,(Q, K, V) = [\text{head}_1, \text{head}_2, \text{head}_3, \ldots, \text{head}_n]\, W_0 \qquad (2)$$

where Q, K, and V are query, key, and value vectors respectively, and $\text{head}_i = \text{Attention}(QW^Q_i, KW^K_i, VW^V_i)$ W is a set of all learnable parameter matrices, n is the number of heads, and head_i is the output of the scaled dot product applied on the ith head [6].

Then, scaled dot product attention is applied to the query, key, and value vectors. The dot product between the query and key vectors is calculated and divided by the square root of the dimensionality of keywords. This is done to obtain the attention scores for each head. The attention weights obtained are embedded onto a single vector with an increased dimensionality, as shown in equation (2). The concatenated output passes through a final linear transformation and the final output is obtained. Self-attention allows the model to simply focus on different parts of its input while processing it, by pooling the short-term features. Deploying Self-multi-head attention on a CNN model allows it to map the features into a long-term speaker embedding [7]. The structure of the multi-head attention model [6] is shown in Figure 11.

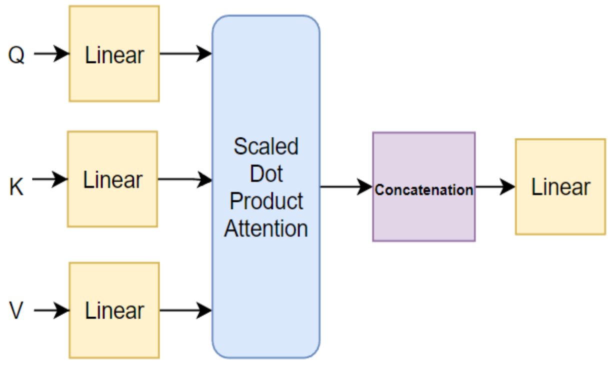

Fig. 11 Multi-head attention model

3.5 Regularization

In this research, we use regularization to prevent overfitting and generalize the model. This is done by adding a penalty term to the loss function that discourages the model from overfitting by either constraining the magnitude of the model weights or adding a complexity cost. The focus is on Dropout and L2 Regularizers. Dropout regularizers prevent over-dependence on a single feature. It randomly drops out (sets to zero) a proportion of the activations in a layer during training. L2 or Ridge Regularizer adds a penalty proportional to the square of the weights. This tends to shrink the weights towards zero but typically does not result in a sparse solution.

4 Experiments and Simulations

4.1 Dataset Preparation

The audio samples used in this research were procured from the Google Speech Commands Dataset [25]. The Google Speech Commands Dataset contains 65000 utterances of 30 short words each of one-second duration in the *.wav* format. All these audio samples were recorded under real-time scenarios using microphones on smartphones and laptops. From this dataset, we choose the word ‘Marvin’, as the wake word for our research. The dataset contains 1746 audio samples for the word 'Marvin'. Ideally, all the audio samples except those of 'Marvin' would constitute the false label, i.e., the scenario where the wake word goes undetected. But this would cause a huge data distribution bias and may even lead to cases of under-fitting as the true data samples (Marvin audio samples) are way fewer than the False samples. Therefore, 61 audio samples were chosen at random using a Python script, from each of the remaining 29 words to have a balanced distribution of data between the true and false samples. The dataset for false samples contains 1769 audio samples. The newly prepared dataset contains 1746 audio samples for 'Marvin' and 1769 samples for false data.

4.2 Dataset Preprocessing and Augmentation

Since the audio inputs are recorded in real-time scenarios, denoising is necessary. Prior research involves using the Morse Wavelet function for wavelet decomposition [10]. Morse wavelets can capture both local and global features from the audio, but they can be computationally intensive. On the other hand, db4 wavelets can denoise the signal while preserving the essential signal features at low computational complexity. On those lines, denoising was carried out in this research by the Wavelet decomposition technique using the 'db4' wavelet, which belongs to the Daubechies family of wavelets.

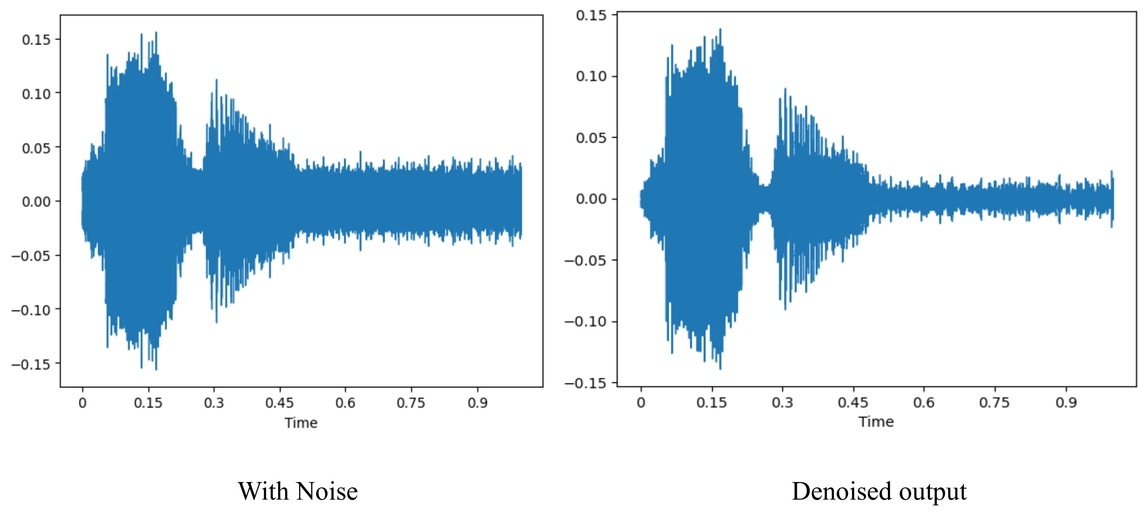

Fig.12 Visualizing an audio waveform with and without noise

The first image represents an audio sample from the generated dataset in its natural state (with noise). The second image represents the audio sample after getting denoised with wavelet decomposition using the db4 wavelet.

Applying data augmentation on the newly prepared dataset allows for the generation of more data in order to generalize the model and enhance its performance. Data augmentation techniques were employed for time stretching, pitch scaling, white noise addition, polarity inversion, and random gain [11]. After data augmentation, the dataset size increased to 10476 true samples and 10614 false samples (21,090 total audio samples), as shown in Table 1.

Table 1 Dataset sizes before and after data augmentation

	Before Data Augmentation	After Data Augmentation
True Samples	1746	10476
False Samples	1769	10614

4.3 Feature Extraction

This experiment uses the LFCCs extracted from the audio files using the *spafe* python library to train the model. The extracted LFCCs are stored in a CSV file which is later used for the generation of training, validation, and testing data samples. After experimenting with different proportions of train, test, and validation ratios, a ratio of 80% train, 10% test and 10% validation has been found to be the most optimized distribution. These LFCCs are saved as NumPy files and are reloaded into the environment later for model training and testing. A similar process was carried out for extracting MFCCs from the audio samples. The models were trained on MFCC and LFCC features separately, and their performances were compared to check whether the use of different features impacts model performance.

Although the Google dataset claims that all audio files are one second in length, on analysis, we found that there were audio files with a slightly smaller duration. This is an inconsistency, as when we compute features, the feature array's length is directly proportional to the audio file's duration. This meant that for those audio files, the feature-length array was lesser than what was specified for the model. To ensure that the feature input supplied to the model has the same dimensions as required by the model, we perform zero padding. In zero padding, we add the value '0' as many times as required until the feature array length for that audio file is equivalent to what would be the ideal feature length assuming duration as one second.

4.4 Custom CRNN Model

As the proposed methodology section states, we create two custom CRNN model architectures. The CRNN_GRU_GRU architecture consists of three 1D convolutional layers with filter sizes 32, 64, and 128, followed by two GRU layers with 128 units each. Similarly, the CRNN_GRU_LSTM consists of three 1D convolutional layers with filter sizes 32, 64, and 128, followed by a GRU layer and an LSTM layer with 128 units each. Finally, a dense layer with 2 units and a softmax activation function is used for the final classification. Further research was conducted within the proposed models' architectures to explore possibilities of enhancing model performance. The model performance was tested in the presence and absence of multi-head attention, L2 regularizer, and an extra dense layer with 32 units. The baseline models chosen for this research are based on Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Long-Short term memory networks (LSTM), and Gated Recurrent Unit (GRU) Architectures. The performances of our proposed CRNN models and their variants were compared with the baseline models.

The proposed CRNN models belong to the 250k parameter budget. The best-performing variants among these were chosen. In an attempt to observe model performances with reduced complexity, a 100k parameter budget variant was created for their respective 250k parameter budget counterparts. The parameter reduction was done by experimenting with the number of units in different layers and introducing batch normalization, although the relative architecture definition remained the same.

All the models were run for 120 epochs on the Google Colab platform, with a learning rate of 0.001, categorical cross-entropy loss function, and the AMSGrad optimizer. The various metrics considered for model evaluation are accuracy, precision, recall, f1 score, the number of False Positives and False Negatives, and Latency.

4.5 Performance Metrics

Accuracy generally describes the performance of the model across all the classes. This metric might be deceptive in scenarios where there is an unbalanced class distribution. Hence, the number of false positives and false negatives, precision, and recall were also chosen for evaluating the performance of the proposed model. These metrics are more relevant to the problem statement.

Precision is the ratio of samples that are correctly classified into a class, to the total number of samples classified into that class as shown in equation (3).

$$\text{Precision} = \text{True Positives} / (\text{True Positives} + \text{False Positives}) \qquad (3)$$
$$\text{Precision} = \text{True Negatives} / (\text{True Negatives} + \text{False Negatives})$$

The recall is the ratio of correctly classified samples in a class to the total number of samples originally present in the class, as shown in equation (4).

$$\text{Recall} = \text{True Positives} / (\text{True Positives} + \text{False Negatives}) \qquad (4)$$
$$\text{Recall} = \text{True Negatives} / (\text{True Negatives} + \text{False Positives})$$

The F1 score is another metric used to estimate a model's accuracy. It takes precision and recall values into account and elaborates on the class-wise performance of the model as shown in equation (5).

F1 score = (2 x Precision x Recall)/(Precision + Recall) (5)

The total number of False Predictions is the sum of the number of false accepts (false positives) and the number of false rejects (false negatives) as shown in equation (6).

False predictions = False Positives (FP) + False Negatives (FN) (6)

The false prediction percentage illustrates the improvement or degradation in model performance with respect to the baseline model as shown in equation (7). A positive value denotes an improvement and a negative value denotes a degradation in model performance.

Reduction in False Prediction % = (False Predictions (Baseline) - False Predictions (Model)) *100 / False Predictions (Baseline) (7)

The final area of focus in this research is model latency. Wake word detection is just a part of the entire process flow of a voice assistant and is expected to happen very quickly. Hence, model latency is a crucial parameter for wake word detection. Model latency varies based on the audio file it runs on. Audio files with more complex acoustic features require more processing time, resulting in higher prediction latency. Hence we need to judge a model's latency on average. We compute the mean latency of the model as the summation of prediction latencies for all audio files in the test dataset divided by the total number of audio files in the test dataset.

5 Results and Discussions

The audio dataset is prepared from the samples in the Google Speech Commands Dataset [25]. This dataset is passed through the wavelet decomposition function for denoising. Following this, data augmentation is performed to increase the dataset size. The proposed models are trained on 10476 true samples and 10614 false samples, and the following observations are recorded. The number of false positives and false negatives, precision, recall, and latency are the metrics chosen for evaluating the performance of our proposed models. On comparing the model performances after training them using MFCCs and LFCCs, we observe that LFCCs consistently outperform MFCCs with significantly higher accuracies for all our models, as shown in Tables 2 and 3.

In the following tables, CRNN_GRU_LSTM is the name assigned to the CRNN model with convolutional layers followed by GRU and LSTM recurrent layers. Similarly, CRNN_GRU_GRU stands for the CRNN model with convolutional layers, followed by two GRU recurrent layers.

Table 2 Comparing model accuracies using types of features

Proposed Model	MFCC	LFCC
DNN	0.7141	0.8113
CNN	0.9374	0.9317
LSTM	0.9075	0.9692
GRU	0.9431	0.9787
CRNN_GRU_LSTM	0.9493	0.9815
CRNN_GRU_GRU	0.9663	0.982

Table 2 shows the comparison between the accuracies obtained by training the baseline models and proposed CRNN models using MFCCs and LFCCs. LFCCs show a significant improvement in the model accuracies when compared to MFCCs in the case of DNN, LSTM, and GRU models. Only in the case of CNN, the accuracy obtained by training the model on MFCCs is slightly higher than the accuracy obtained using LFCCs.

Table 3 Comparing accuracies of models with attention using types of features

Proposed Model with Multi-Head Attention	MFCC	LFCC
DNN	0.6757	0.7885
CNN	0.9331	0.9346
LSTM	0.9417	0.9753
GRU	0.9545	0.981
CRNN_GRU_LSTM	0.9545	0.9829
CRNN_GRU_GRU	0.9469	0.9829

Table 3 draws a comparison between the model performances with MFCCs and LFCCs in the presence of the multi-head attention model in their neural network architectures. Yet again, it has been proven that LFCCs perform significantly better, compared to MFCCs for our baseline and proposed models. In general, while comparing the accuracies obtained by training the model on LFCCs, we observe that there has been a slight increase in the accuracy in the presence of an attention model.

Earlier research on speaker recognition compares the performance of LFCCs and MFCCs, for male and female audio trials and LFCCs show better performance in the case of female trials [15]. On the other hand, in our research, LFCCs exhibit better performance on both male and female audio samples. This may be attributed to the fact that the audio samples used for wake word detection in our research are around 1 second in length, and the audio samples used for speaker recognition in the previous research are around 2.5 minutes on average. Hence, we establish that LFCCs are a better choice compared to MFCCs, for wake word detection. Moving forward, further experiments were conducted using LFCC features as input.

We observe that the CRNN_GRU_GRU and CRNN_GRU_LSTM models trained on LFCCs show higher accuracy than the other models. They have an accuracy close to the GRU standalone model, but it is important to note that the CRNN_GRU_GRU and CRNN_GRU_LSTM models train faster and hence have an edge even before further experimentation.

Prior research using CRNNs and transformer architectures assessed their model performances based on the false accept ratios (False positives) and false reject ratios (False negatives) [24]. False prediction count, i.e., the sum of False positives and False Negatives, is a crucial metric because its constituents instigate false alarming or false triggering among voice assistants. Existing research emphasizes reducing false predictions, hence bringing down the false alarm rates [8] [20]. On those lines, we assess the performance of our proposed models based on metrics like false positives, false negatives, precision, recall, and f1 score.

Table 4 Metrics of Base Models and Proposed Custom Models

Models	False Positives	False Negatives	Precision	Recall	F1 Score
DNN	338	60	0.92 (0), 0.75 (1)	0.68 (0), 0.94 (1)	0.78 (0), 0.83 (1)
CNN	73	71	0.93 (0), 0.93 (1)	0.93 (0), 0.93 (1)	0.93 (0), 0.93 (1)
LSTM	47	18	0.98 (0), 0.96 (1)	0.96 (0), 0.97 (1)	0.97 (0), 0.97 (1)
GRU	25	20	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)
CRNN_GRU_LSTM	18	21	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)
CRNN_GRU_GRU	18	20	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)

As seen in Table 4, DNN exhibits poor performance, and the model performances gradually improve as we move downwards in the table. CRNN_GRU_LSTM and CRNN_GRU_GRU exhibit the lowest values of false negatives and false positives and high precision, recall, and F1 scores. Hence, the proposed models based on the CRNN architecture are inherently better than our baseline models. Even though the precision, recall, and F1 scores of the GRU baseline model are the same as our proposed CRNN models, there is a significant decrease in the number of false positives and false negatives.

Table 5 Metrics of Base Models and Proposed Custom Models with Multi-Head Attention

WITH ATTENTION	False Positives	False Negatives	Precision	Recall	F1 Score
DNN	418	28	0.96 (0), 0.71 (1)	0.60 (0), 0.97 (1)	0.74 (0), 0.82 (1)
CNN	70	68	0.94 (0), 0.93 (1)	0.93 (0), 0.94 (1)	0.93 (0), 0.93 (1)
LSTM	29	23	0.98 (0), 0.97 (1)	0.97 (0), 0.98 (1)	0.98 (0), 0.98 (1)
GRU	21	19	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)
CRNN_GRU_LSTM	17	19	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)	0.98 (0), 0.98 (1)
CRNN_GRU_GRU	19	14	0.99 (0), 0.98 (1)	0.98 (0), 0.99 (1)	0.98 (0), 0.98 (1)

Multi-head attention was added to the baseline and proposed CRNN models. The observations have been tabulated in Table 5. Table 5 shows that the DNN model again exhibits poor performance, and the general performance improves as we move down the table, where the proposed custom CRNN models yield the least number of false positives and false negatives. Now it has been established that the proposed CRNN_GRU_LSTM and CRNN_GRU_GRU models have shown better performance when compared to the baseline models.

Further experiments were conducted on the proposed custom CRNN model architecture. The performance metrics for the proposed custom CRNN_GRU_GRU model were tabulated in the presence and absence of different combinations of multi-head attention, L2 regularizer, and an extra dense layer with 32 units, as shown in Table 6. We perform this experiment further to improve the performance of proposed custom CRNN models, and hence we compare them with their respective proposed baseline CRNN models (CRNN_GRU_GRU and CRNN_GRU_LSTM).

Table 6 Comparison of metrics for various proposed CRNN_GRU_GRU model variants

#	Proposed CRNN_GRU_GRU Variants	Accuracy	False Positives (FP)	False Negatives (FN)	False Predictions (FP + FN)	Reduction in False Predictions (%)
1	Without Extra Dense layer, Without Attention	0.982	18	20	38	Baseline
2	Without Extra Dense layer, With Attention	0.9844	19	14	33	13.157
3	Without Attention, With Regularizer	0.9815	21	18	39	-2.631
4	With Attention, With Regularizer	0.981	19	21	40	-5.2631
5	With Extra Dense layer, Without Attention	0.9725	17	41	58	-52.631
6	With Extra Dense layer, With Attention	0.9825	25	12	37	2.631
7	With Extra Dense layer, With Regularizer, Without Attention	0.9791	25	19	44	-15.789
8	With Extra Dense, With Regularizer, With Attention	0.9806	22	19	41	-7.894

*From Table 8 onwards, the nomenclature CRNN_GRU_GRU variant #, will be based on the numbering given in Table 6.

Amongst the custom CRNN_GRU_GRU model variants, we observe that model variant #2 in Table 6, produces the highest accuracy (0.9844) with 19 false positives, 14 false negatives, and a 13.15% reduction in false predictions with respect to the CRNN_GRU_GRU baseline model. The second-best performer is model variant #6, which produces an accuracy of 0.9825 with 25 false positives, 12 false negatives, and an overall reduction of 2.63% in false predictions. The other models show an increase in false predictions compared to the baseline, making themselves undesirable for our application. The total number of positive and negative samples used for testing the model are 1049 and 1060, respectively (only testing data). Our best-performing model among the CRNN_GRU_GRU variants, i.e., model variant #2, has outputs only 19 false positives out of 1060 negative samples and 14 false negatives out of 1049 positive samples.

Table 7 Comparison of metrics for various proposed CRNN_GRU_LSTM model variants

#	Proposed CRNN_GRU_LSTM Variants	Accuracy	False Positives (FP)	False Negatives (FN)	False Predictions (FP+FN)	Reduction in False Predictions (%)
1	Without attention, without extra Dense layer	0.9815	18	21	39	Baseline
2	With attention, without extra Dense layer	0.9829	17	19	36	7.692
3	Without attention, with regularizer	0.981	22	18	40	-2.564
4	With attention, with regularizer	0.9768	30	19	49	-25.641
5	With an Extra Dense layer, without Attention	0.9853	19	12	31	20.512
6	With an Extra Dense layer, Attention	0.981	21	19	40	-2.564
7	With an Extra Dense layer, With Regularizer, without Attention	0.9791	27	17	44	-12.820
8	With an Extra Dense layer, With Regularizer, with Attention	0.9829	18	18	36	7.692

*From Table 8 onwards, the nomenclature CRNN_GRU_LSTM variant #, will be based on the numbering given in Table 7.

Similar experiments were performed on the CRNN GRU-LSTM architecture in the presence and absence of different combinations of an extra Dense layer, L2 Regularizer, and Multi-head Attention, and the observations were tabulated as shown in Table 7. Model variant #5 from Table 7, produces the highest accuracy of 0.9853, with 19 false positives out of 1060 negative samples, 12 false negatives out of 1049 positive samples, and an overall reduction of 20.51% in false predictions with respect to the CRNN_GRU_LSTM baseline model. Following this, model variant #8 produces the next highest accuracy of 0.9829, with 18 false positives and 18 false negatives out of 1049 positive samples. Model variant #2 also produces an accuracy of 0.9829, with 17 false positives and 19 false negatives. Model variants #2 and #7 output an overall reduction of 7.69% in false predictions. Of the two models making the same accuracy of 0.9829, model variant #2 proves to be better because it has a lower model complexity than the latter (based on the number of parameters). The other model variants output an increased number of false predictions and are not suitable for our application. Only a slight improvement is observed among the accuracies for certain model variants, after experimenting with the CRNN_GRU_GRU and CRNN_GRU_LSTM models, but, a substantial reduction is observed in the false prediction percentage.

We reiterate that we performed the comparisons shown in Tables 6 and 7, to explore the possibilities of performance enhancement in the proposed custom CRNN models when compared to their respective proposed baseline CRNN models (CRNN_GRU_GRU #1 and CRNN_GRU_LSTM). Following this, the best-performing CRNN model variants were chosen from Tables 6 and 7. The parameter counts for these models fall close to 250k parameters, which is a reasonable parameter size when compared to the TinyCRNN model's parameters. In Table 8, we compare the performance of these top five CRNN models with respect to CNN and DNN baselines.

Table 8 Comparison of the 250k parameter budget CRNN models with CNN and DNN baselines

			False Predictions count (FP + FN)			**Reduction in False Predictions (%)**	
Proposed Models & TinyCRNN Model	**Accuracy**	**Parameters**	**False Positives (FP)**	**False Negatives (FN)**	**FP + FN**	**With respect to CNN Baseline**	**With respect to DNN Baseline**
CRNN_GRU_LSTM #5	0.9853	220770	19	12	31	77.536	92.211
CRNN_GRU_LSTM #2	0.9829	332034	17	19	36	73.913	90.954
CRNN_GRU_LSTM #8	0.9829	335970	18	18	36	73.913	90.954
CRNN_GRU_GRU #2	0.9844	249986	19	14	33	76.086	91.708
CRNN_GRU_GRU #6	0.9825	303458	25	12	37	73.188	90.703
TinyCRNN (MO Khursheed et. al.)	0.9782	247218	25	21	46	66.667	88.442

The results of Table 8 indicate that the best-performing model with respect to the CNN and DNN baselines is the CRNN_GRU_LSTM variant #5. This is followed by the CRNN_GRU_GRU variant #2 and CRNN_GRU_LSTM variants #2 and #8. We also observe very clearly that all the proposed CRNN variants perform significantly better than the TinyCRNN model [24] with respect to CNN and DNN baselines. To compare the performance of the TinyCRNN architecture with our proposed CRNN models, we first construct the model based on the specified architecture [24] and then train it on the LFCCs we extracted from the audio files in the Google dataset. Since the feature types and dimensions used in our research are different, we obtain slightly different parameters.

We further experimented with these variants to see if parameter reduction helps in boosting performance with a secondary goal of reducing model complexity. The 250k parameter budget models mentioned in Table 8 were chosen, and the performances of their respective lighter versions (with a 100k parameter budget) are compared with the DNN and CNN baselines. The results of this comparison has been presented in Table 9.

Table 9 Comparison of 100k parameter budget CRNN models with CNN and DNN baselines

			False Predictions count (FP + FN)			**Reduction in False Predictions (%)**	
Proposed Models & TinyCRNN Model	**Accuracy**	**Parameters**	**False Positives (FP)**	**False Negatives (FN)**	**FP + FN**	**With respect to CNN Baseline**	**With respect to DNN Baseline**
CRNN_GRU_LSTM #5 (TurboCRNN)	0.9839	87330	19	15	34	75.362	91.457
CRNN_GRU_LSTM #2	0.9825	102834	21	16	37	73.188	90.703
CRNN_GRU_LSTM #8	0.9815	161538	22	17	39	71.739	90.201
CRNN_GRU_GRU #2	0.9839	134242	15	19	34	75.362	91.457
CRNN_GRU_GRU #6	0.9829	125698	19	17	36	73.913	90.954
TinyCRNN (MO Khursheed et. al.)	0.9758	137362	24	27	51	63.043	87.185

The results shown in Table 9 show that all our proposed custom CRNN models within the 100k parameter budget also exhibit better performance than the TinyCRNN model [24]. The model variants CRNN_GRU_LSTM #5, and CRNN_GRU_GRU #2 are the top performers among the models within the 100k parameter budget, with a 75% reduction in false predictions over the CNN baseline and a 91% reduction in false predictions over the DNN baseline. They are followed by the other 3 model variants, which show over 70% reduction in false predictions over the CNN baseline and more than 90% reduction in false predictions over the DNN baseline. The CRNN_GRU_LSTM #5 and CRNN_GRU_GRU #2 model variants exhibit equally good performance, but considering model complexity, we consider the CRNN_GRU_LSTM #5 model variant to be the overall best performer, since it is capable of achieving the same results as the CRNN_GRU_GRU #2 model variant, with a relatively lesser number of model parameters.

For the final metric, i.e., model latencies, the best-performing proposed models in each architecture were chosen and a comparison was drawn among the mean latencies for each with all the data samples and the observations having been tabulated as shown in Table 10. Prior research works have created lightweight models and have proposed methods to deploy the wake word detection models on embedded devices like Raspberry Pi, and ARM Cortex M4 processors [19-22]. In this direction, to understand if there is an impact in case the model is run on advanced hardware, say a microcontroller with a Graphics Processing Unit (GPU) capable of hardware acceleration like Jetson Nano or Beaglebone Black, we perform the same latency computation by running the model in the presence and absence of a GPU.

Table 10 Model Latency Comparison with and without GPU

	Latency (in ms) - 250k Parameter budget model		**Latency (in ms) - 100k Parameter budget model**	
Model	**With GPU**	**Without GPU**	**With GPU**	**Without GPU**
DNN with Attention (Baseline)	66.04	79.61	66.04	79.61
CNN with Attention (Baseline)	76.32	91.63	76.32	91.63
LSTM with Attention (Baseline)	76.21	93.03	76.21	93.03
GRU with Attention (Baseline)	73.51	104.19	73.51	104.19
CRNN_GRU_LSTM #2 (Proposed)	74.99	66.86	70.6	70.82
CRNN_GRU_LSTM #5 (Proposed)	75.4	70.77	73.9	80.57
CRNN_GRU_LSTM #8 (Proposed)	75.31	87.57	75.68	73.82
CRNN_GRU_GRU #2 (Proposed)	74.55	63.45	73.24	69.99
CRNN_GRU_GRU #6 (Proposed)	75.95	65.75	71.11	93.31
Tiny CRNN (MO Khursheed et. al.)	72.91	125.49	72.02	95.36

Table 10 compares the latencies of the baseline models (respective architectures) and the best-performing proposed custom CRNN model variants. The latencies for models with 250k parameter budget and 100k parameter budget have been tabulated as shown in Table 10. The latencies of all our proposed CRNN model variants are less than the baseline models. Prior research has proposed a model with knowledge distillation and phone-aligned training, which is capable of bringing down the latency for wake word detection below 250ms [22]. On the other hand, all the proposed custom CRNN models in our research have a latency below 90ms. Among the models within the 100k parameter budget, the CRNN_GRU_LSTM variant #2 produces the least latency of 70ms, which is less than that of the TinyCRNN model, in the presence and absence of GPU. Among the models within the 250k parameter budget the CRNN_GRU_GRU variant #6 outputs low latencies of 74.55ms and 63.45ms in the presence and absence of GPU.

As expected, the latencies of baseline models in DNN, CNN, and RNN architectures increase in the absence of a GPU. On the other hand, in some cases, the latencies of the proposed custom CRNN models slightly reduce in the absence of a GPU. When running a model based on the CRNN architecture, the choice of hardware can have an impact on the latency and processing time. In general, GPUs are designed to handle parallel computations, which means they can process multiple tasks simultaneously. However, when it comes to tasks that require a high degree of sequential processing, such as the forward propagation of data through the proposed model in the CRNN architecture, the overhead involved in coordinating the parallel processing can slow down the overall computation.

In this case, running a CRNN on a CPU instead of GPU results in lower latency. This is because CPUs are better suited to handling sequential computations. The CPU can execute one task at a time but can do so quickly and efficiently. In contrast, a GPU may have to spend more time coordinating the parallel processing of data between its many processing cores. But we also observe that in the case of the CRNN_GRU_LSTM variant #8, the absence of GPU worsens the latency. L2 Regularizer adds a penalty term to the loss function. This requires additional computations during training and inference to calculate and update the gradients. This increases the computational requirements of the model and makes it slower to run on a CPU when compared to the other variants. The proposed custom CRNN models exhibit a lower latency when compared to the baseline models. They have the least latency both in the presence and absence of GPUs. This proves that the proposed models are the most consistent among the models designed in their respective architectures irrespective of hardware.

To summarize the experiments performed, the experimentation began with a comparison between the baseline and proposed model performances when trained on MFCCs, and LFCCs. Following this, an analysis of the false positives, false negatives, precision, recall, and f1 score was presented for the baseline and proposed models. Despite obtaining an accuracy of 0.98, further research was done on the model architectures. In an attempt to reduce false positives and false negatives, model variants were created by experimenting with multi-head attention, regularizer, and an extra dense layer. Following this, a comparison of the reduction in false prediction percentage for proposed CRNN model variants, with respect to their respective baseline CRNN models, was presented. The best-performing CRNN model variants were chosen and they were compared with the CNN and DNN baseline models and the TinyCRNN model [24]. All the best-performing models chosen for this step had a 250k parameter budget. To verify whether a reduction in model complexity would lead to better results, 100k parameter budget variants were constructed for their respective 250k parameter budget counterparts. Finally, a comparison of prediction latencies has been presented between the TinyCRNN model [24], the baseline models, and our best-performing models.

After thoroughly analyzing all the results obtained via experimentation, we deduce that the proposed TurboCRNN, i.e., the CRNN_GRU_LSTM variant #5 (87k parameters) designed based on the architecture shown in Figure 9, is the ideal architecture for wake word detection. Among all the proposed models, it has the least number of False Predictions, Model Parameters, and a very low Latency with respect to the baseline models. The TurboCRNN model also exhibits better performance than the TinyCRNN model in terms of the aforementioned metrics, i.e. False prediction percentage, model complexity, and latency.

In short, experiments were conducted on feature extraction, data preprocessing, and various model architectures from which important metrics like accuracy, false positives, false negatives, precision, recall, and latency were recorded. In the following section, we judge the best models in terms of the metrics obtained from these experiments.

6 Conclusion

This paper proposes a novel TurboCRNN architecture trained on LFCC features obtained from audio files denoised by the Wavelet Decomposition Technique using 'db4' wavelets, followed by data augmentation to increase the quantity of audio data for training. Experiments conducted on the model architectures using different combinations of extra-dense layers, multi-head attention, and regularizers as shown in Figures 9 and 10, revealed significant differences in model performance. The model performances were compared based on false prediction percentage, model complexity, and latency. On introducing LFCCs, the model accuracies show a remarkable increase, compared to the improvements observed on training the model with MFCCs. Denoising data using Wavelet decomposition also aids the process of accuracy improvement and false prediction minimization. All our proposed CRNN model variants show a significant decrease in the number of parameters (decrease in model complexity), and false prediction percentages, compared to the TinyCRNN model [24]. The best performers (in terms of accuracy and False Predictions) among the 250k parameter budget and 100k parameter budget variants are the CRNN_GRU_LSTM variant #5 (220k parameters) and TurboCRNN respectively. Overall, the TurboCRNN exhibits the highest average reduction in false predictions of 75% and 91%, over the CNN and DNN baselines respectively. TurboCRNN exhibits low false prediction rates and the least model complexity, compared to the other proposed model variants and the TinyCRNN baseline. The results also show that TurboCRNN outputs a lower latency compared to the conventional wake word detection techniques. The proposed TurboCRNN model displays optimal performance in all aspects, i.e. low false predictions, latency, and model complexity, and is an ideal solution for Wake Word detection.

Statements and Declarations

Competing Interests: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] Rose R J, Paul D (1990) A hidden Markov model-based keyword recognition system. *International Conference on Acoustics, Speech, and Signal Processing*. https://doi.org/10.1109/icassp.1990.115555

[2] Kumatani K, Panchapagesan S, Wu M, Kim M, Strom N, Tiwari G, Mandai A (2017) Direct modeling of raw audio with DNNS for wake word detection. *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. https://doi.org/10.1109/asru.2017.8268943

[3] Rebai, I., Ayed, Y. B., & Mahdi, W. (2019). Spoken keyword search system using improved ASR engine and novel template-based keyword scoring. *Multimedia Tools and Applications*, 78(2):1495–1510. https://doi.org/10.1007/s11042-018-6276-y

[4] Jose C, Mishchenko Y, Senechal T, Shah A, Escott A, Vitaladevuni S N P (2020) Accurate Detection of Wake Word Start and End Using a CNN. *ArXiv (Cornell University)*. https://doi.org/10.21437/interspeech.2020-1491

[5] Oruh J, Viriri S, Adegun A (2022) Long Short-Term Memory Recurrent Neural Network for Automatic Speech Recognition. *IEEE Access*, *10*:30069–30079. https://doi.org/10.1109/access.2022.3159339

[6] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser L, Polosukhin I (2017) Attention is All you Need. *Neural Information Processing Systems*, *30*:5998–6008. https://arxiv.org/pdf/1706.03762v5

[7] India M, Safari P, Hernando J (2019) Self Multi-Head Attention for Speaker Recognition. *Conference of the International Speech Communication Association*. https://doi.org/10.21437/interspeech.2019-2616

[8] Kumar R, Rodehorst M, Wang J, Gu J, Kulis B (2020) Building a Robust Word-Level Wakeword Verification Network. *Conference of the International Speech Communication Association*. https://doi.org/10.21437/interspeech.2020-2018

[9] Graps A L (1995) An introduction to wavelets. *IEEE Computational Science and Engineering*, *2*(2):50–61. https://doi.org/10.1109/99.388960

[10] Wang L, Zheng W, Ma X, Lin S (2021) Denoising Speech Based on Deep Learning and Wavelet Decomposition. *Scientific Programming*, *2021*, 1–10. https://doi.org/10.1155/2021/8677043

[11] Salamon J, Bello J P (2017) Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification. *IEEE Signal Processing Letters*, *24*(3):279–283. https://doi.org/10.1109/lsp.2017.2657381

[12] Fang Z, GuoLiang Z, Zhanjiang S (2001) Comparison of different implementations of MFCC. *Journal of Computer Science and Technology*, *16*(6):582–589. https://doi.org/10.1007/bf02943243

[13] Dua S, Kumar S S, Albagory Y, Ramalingam R, Dumka A, Singh R, Rashid M, Gehlot A, Alshamrani S S, AlGhamdi A S (2022) Developing a Speech Recognition System for Recognizing Tonal Speech Signals Using a Convolutional Neural Network. *Applied Sciences*, *12*(12). https://doi.org/10.3390/app12126223

[14] Lei H, Gonzalo E L (2009) Mel, linear, and antimel frequency cepstral coefficients in broad phonetic regions for telephone speaker recognition. *Conference of the International Speech Communication Association*. https://doi.org/10.21437/interspeech.2009-389

[15] Zhou X, Garcia-Romero D, Duraiswami R, Espy-Wilson C Y, Shamma S A (2011) Linear versus mel frequency cepstral coefficients for speaker recognition. *IEEE Automatic Speech Recognition and Understanding Workshop*. https://doi.org/10.1109/asru.2011.6163888

[16] Kingma D P, Ba J (2014) Adam: A Method for Stochastic Optimization. *ArXiv (Cornell University)*. https://www.arxiv.org/pdf/1412.6980

[17] Dozat T, "Incorporating Nesterov Momentum into Adam," *OpenReview*, Feb. 18, 2016. https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ

[18] Reddi S J, Kale S, Kumar S (2019) On the Convergence of Adam and Beyond. *ArXiv (Cornell University)*. https://arxiv.org/pdf/1904.09237.pdf

[19] Yungen W, Gong Z, Shunzhi Y, Ye K, Wen Y (2021) EdgeCRNN: an edge-computing oriented model of acoustic feature enhancement for keyword spotting. *Journal of Ambient Intelligence and Humanized Computing*. https://doi.org/10.1007/s12652-021-03022-1

[20] Sørensen P. M, Epp B, May T. (2020) A depthwise separable convolutional neural network for keyword spotting on an embedded system. *Eurasip Journal on Audio, Speech, and Music Processing*, *2020*(1). https://doi.org/10.1186/s13636-020-00176-2

[21] Tsai, T., & Lin, X. (2023). Speech densely connected convolutional networks for small-footprint keyword spotting. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-023-14617-5

[22] Ulkar M G, Okman O E (2021) Ultra-Low Power Keyword Spotting at the Edge. *ArXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2111.04988

[23] Ghosh A, Fuhs M C, Bagchi D, Farahani B, Woszczyna M (2022) Low-resource Low-footprint Wake-word Detection using Knowledge Distillation. *ArXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2207.03331

[24] Khursheed M, Jose C, Kumar R, Fu G, Kulis B, Cheekatmalla S. K (2021) Tiny-CRNN: Streaming Wakeword Detection in a Low Footprint Setting. *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. https://doi.org/10.1109/asru51503.2021.9688299

[25] Warden P (2018) "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *arXiv.org*. https://arxiv.org/abs/1804.03209v1