IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 February 2024, accepted 14 March 2024, date of publication 21 March 2024, date of current version 27 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3380350

RESEARCH ARTICLE

Efficient Real-Time Smart Keyword Spotting Using Spectrogram-Based Hybrid CNN-LSTM for Edge System

INFALL SYAFALNI[1,2,3], (Member, IEEE), CLARENCE AMADEUS[1], NANA SUTISNA[1,2,3], (Member, IEEE), AND TRIO ADIONO[1,2], (Senior Member, IEEE)

[1]School of Electrical Engineering and Informatics, Bandung Institute of Technology, Bandung 40132, Indonesia
[2]University Center of Excellence on Microelectronics, Bandung Institute of Technology, Bandung 40132, Indonesia
[3]Interuniversity Microelectronics Centre (IMEC), 3001 Leuven, Belgium

Corresponding author: Infall Syafalni (infall@ieee.org)

This work was supported by Riset Unggulan Pusat dan Pusat Penelitian (RU3P), Institut Teknologi Bandung (ITB).

ABSTRACT Keyword Spotting (KWS) is the task of recognizing spoken command words from a database. With recent application human-machine interactions, KWS systems require real-time performance, where edge computing is a preferable option. To allow KWS systems to work on fast and real-time implementation, a low-complexity yet high-accurate AI model is mandatory. In this paper, we propose a comprehensive voice command recognition system design and its hardware implementation. The proposed AI model considered in this system is SpectroNet-based and an efficient hybrid CNN-LSTM architecture with low complexity. Jetson Xavier NX is an edge device because of its strong computational power as an embedded device. The implementation result shows the proposed method offers quite good in terms of accuracy, indicated by no accuracy drop between the model implemented in PC and Jetson Xavier. However, the inference time is quite high, which is 180 ms/step. To improve the speed of the system, the TensorRT library is used to further optimize the model. Optimization of the model is found effective, reducing 59.35% of the total operation performed in SpectroNet when FP32 precision is used, and 59.63% when FP16 precision is used. The model is also sped up by 45% if FP32 precision mode is used and 62% if FP16 precision mode is used. However, there is a slight accuracy drop of 2.68% if FP32 precision mode is used and 4.84% if FP16 precision mode is used. This slight drop in accuracy is considered negligible compared to the performance boost that TensorRT gives. The work is useful for intelligent control systems such as smart vehicles, smartphones, computers, and smart communications.

INDEX TERMS Edge computing, hybrid CNN-LSTM, keyword spotting, real-time, embedded devices.

I. INTRODUCTION

Interaction with machines, particularly using voice commands has gained much attention in both academia and industry with the advancement of speech recognition and Artificial intelligence (AI) technology. Voice assistants such as Amazon's Alexa, Apple's Siri, Google's Assistant, and Microsoft's Cortana are some application examples of keyword-spotting using the Automatic Speech Recognition (ASR) technique [1].

Automatic Speech Recognition (ASR) refers to the task of recognizing spoken words from an audio input. One of the essential tasks within ASR in speech-enabled devices is Keyword Spotting (KWS) [2]. In KWS, a specific set of keywords is going to be recognized or detected. However, ASR covers a wide range of techniques and applications in converting spoken language to text. Keyword Spotting (KWS) is the task of recognizing spoken words from a limited set of predefined word options [1]. By fast-growing deep learning techniques and algorithms, KWS allows the activation of voice assistants in small electronic devices [1], [3]. Unlike full voice recognition systems such as ASR, KWS focuses on classifying spoken words without the need for transcribing them into text. Moreover, KWS focuses only on a small set of pre-defined keywords. Thus, KWS systems can perform on almost all speech-enabled devices

The associate editor coordinating the review of this manuscript and approving it for publication was M. Venkateshkumar.

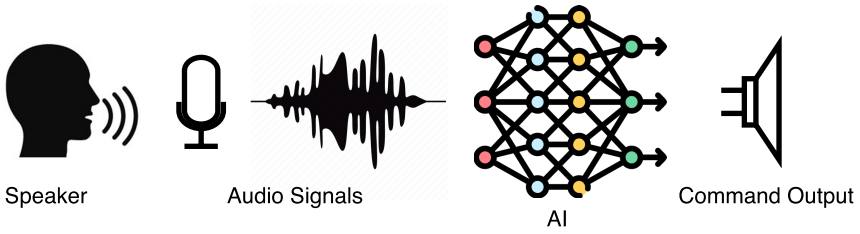

FIGURE 1. Keyword spotting using artificial intelligent illustration.

especially small devices with low computational complexity [4]. Figure 1 shows an illustration of a keyword spotting flow using artificial intelligence where a speaker produces words that are converted to audio signals and processed in an intelligent system. Finally, the main goal is that the system can classify the spoken words accurately.

Keyword Spotting (KWS) has become more crucial for human-machine interaction since it allows a user to interact more naturally with their own devices by leveraging their voice, which usually uses a portable or mobile device. However, the implementation of this algorithm on an embedded system is limited by several constraints. These include limited power, memory, and computational capacity [5]. Consequently, KWS models do not require the same complexity as more comprehensive voice recognition systems and must prioritize low complexity to enable real-time performance, especially in embedded systems.

KWS can be implemented through two main paradigms: cloud computing and edge computing. Cloud computing involves utilizing powerful cloud servers to process and classify spoken words, requiring less computational power on the edge side. However, it depends on a robust internet connection and substantial server resources to maintain real-time performance. On the other hand, edge computing leverages embedded systems to directly store and process spoken words, thus eliminating the need for server and internet dependencies. Several common embedded devices for this purpose are NVIDIA Jetson Nano, NVIDIA Jetson Xavier, and FPGAs.

While edge computing offers independence from server and internet connectivity, its real-time performance can be limited due to the computational capabilities of the embedded device. To address this limitation, a low-complexity model becomes essential. Moreover, certain embedded devices, such as NVIDIA Jetson Xavier, have optimization libraries like TensorRT to enhance inference speed, resulting in faster system performance.

In this paper, we introduce SpectroNet, a KWS model with minimal complexity, which we propose to implement on the Jetson Xavier platform. We train the model using the Google Speech Commands dataset and evaluate its performance using real spoken words. To further boost the model's performance, we employ TensorRT for optimization, focusing on the impact of different data precisions available within TensorRT. Lastly, we incorporate an LED indicator to display the system's output.''

In summary, we list our **main contributions** as follows:

1) We employ CNN-LSTM architecture based on Mel Spectrogram to allow receiving raw speech signal input and does not require a different Spectrogram conversion process.
2) We use a lower number of parameters while maintaining high accuracy.
3) Our method offers lower inference time since it uses a smaller number of parameters and optimizes Tensor RT.
4) Finally, we implement and evaluate our proposed method in Jetson Xavier GPU to confirm the real-time processing using real voice data (not a voice data set).

The rest of this paper is organized as follows: Section II discusses previous works. Section III explains basic definitions and properties to provide fundamental knowledge to the reader. In section IV, we describe our proposed KSW Systems, covering deep learning model development, data set preparation, and hardware-software implementations. Section V explains experimental evaluations and discussion on the obtained performance results. Finally, the conclusions are elaborated in Section VI.

II. PREVIOUS WORKS

The literature study done in this paper is divided into 4 major topics, which are Keyword Spotting methods, Keyword Spotting databases, Cloud vs Edge computing, Model Compression, Model Optimization, and Embedded Devices.

A. KEYWORD SPOTTING METHODS

As one of the most popular topics, Keyword Spotting (KWS) problems have been solved using several methods. One of the most popular methods is using Artificial Neural Networks (ANN) [6], [7], [8]. The recent success of Deep Learning (DL) methods in other topics, such as image classification [9], [10], [11], [12], medical signal analysis [13], [14], [15], [16], and weather forecasting [17], [18] has inspired researchers to apply Deep Learning methods to KWS problems. The examples of the most useful DL methods to perform KWS are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). CNN is a good network to classify images, so applying DL methods to solve KWS involves a feature

TABLE 1. Performance and details comparison of the existing CNN-RNN architectures.

Paper	Model	Parameter Size	Accuracy	Database
[20]	CTC	12.21M	98%	custom
[21]	Hybrid CNN-RNN	229K	97.15%	custom
[22]	DenseNet-Bi-LSTM	223K	97.3%	Google Speech Commandsv2
[23]	Gated Convolutional RNN	154.53 K	90.6%	Google Speech Commands
[24]	CNN	-	27-44% improvement to DNN	custom
[25]	CNN-LSTM	-	89% - 97%	3 Subsets of Google Speech Commands

extraction method to convert audio data into image data called Spectrograms [19]. Another DL method that has been used for KWS is Recurrent Neural Network (RNN). The advantage of using RNN is that it needs no phonetic transcription, and it is easier to modify the keyword list without retraining [20].

Besides CNN and RNN alone, works also have been done to combine CNN and RNN into a hybrid CNN-RNN architecture. This architecture is created to exploit the local structure and long-range context of speech data [21]. Some examples of these combinations are CNN-RNN, CNN-LSTM [22], and Gated Convolutional LSTM [23]. Furthermore, a performance comparison of KWS using various Deep Learning methods is shown in Table 1.

B. KEYWORD SPOTTING DATABASES

Keyword Spotting systems need a lot of training data to be able to perform well. Therefore, there are also works to develop keyword-spotting databases. Some of them are documented in papers, but some of them are not available. Keyword-spotting databases usually have a specific topic, such as home automation, wake word detection, etc. Some examples of keyword spotting databases are shown in Table 2.

TABLE 2. Some examples of commonly used KWS dataset.

Paper	Dataset Name	# of class	Topics
[26]	GSC	24	Common Words
[27]	Hey Snapdragon	-	Wake word detection
[29]	Hey Snips	-	Wake word detection
[30]	TIDIGITS	10	Spoken Digit Number

C. CLOUD VS EDGE COMPUTING

In the field of deep learning application, there are 2 major ideas, which are cloud and edge computing [30]. Cloud computing benefits from using less computational resources, but suffers from latency, security, and scalability problems [31]. This makes edge computing a more realistic solution. In the edge computing idea, all processes in the system are computed offline in the computing device.

For edge computing applications, the system needs to be able to perform in real time. This requires less resource-intensive deep learning models. To create less resource-intensive models, several techniques, such as model compression and model optimization can be used. Moreover, specific integrated circuits for keyword spotting are also proposed for edge computing applications [32], [33], [34].

D. MODEL COMPRESSION

Model compression is a technique to identify a model that has as few parameters and as little accuracy drop [35]. Some of the most popular model compression techniques are pruning, knowledge distillation, and quantization [36].

Pruning is used to effectively compress the model size. The compression is done by removing some unimportant neuron or connection and then retraining the network without that neuron or connection. This process is repeated until the desired trade-off between accuracy and the pruning objective is achieved. Pruning theoretically can compress model size more than 10× the initial size [37]. One of the important parameters in pruning is the pruning criterion. The pruning criterion is used to decide which connection is pruned. For example, a small weight connection, whose value is below a certain threshold, is considered unimportant and is pruned [38]. The choice of this pruning criterion decides the outcome of the pruning.

Another popular model compression technique is Knowledge Distillation (KD). KD aims to transfer knowledge that is learned by a complex, computationally intensive model to a simpler, task-specific model [39]. One way to do this is to use a larger, complex model to perform inference on a specific unlabeled dataset, then use that result as labeled training data to train the smaller model [40]. KD has already been successfully used to compress the KWS model in a noisy environment [41], [42]. The work in [41] shows a robust Knowledge Distillation (KD) learning method by computing three distinct distance metrics for KD training and feature extraction processes. In [42], babble noise signal is added to evaluate the proposed method using LeNet-5, SqueezeNet, and EfficientNet-B0.

E. MODEL OPTIMIZATION

Besides model compression, another strategy to improve the system's real-time performance is to use model optimization. The main idea of this model optimization is mixed-precision computation and precision reduction, among other optimization strategies such as weight and activation function calibration. Some of the most popular model optimization frameworks are NVIDIA's TensorRT and Google's TFLite [31]. Studies say that the TensorRT optimized model performs faster, but consumes more energy (+40%), while the TFLite optimized model consumes less energy, but its performance is significantly slower (-62%) [30].

One of the important parameters of model optimization is data precision. Some of the most popular data precision

are 32-bit Floating Point precision (FP32), 16-bit Floating Point precision (FP16), and 8-bit Integer precision (INT8). FP32 is the default precision used by Deep Neural Network (DNN) programming frameworks, such as TensorFlow, so if the model is optimized using FP32 precision, no precision reduction is done. Studies say that an FP32 TensorRT optimized model has 107.69% throughput increase, while an FP16 TensorRT optimized model has 746.15% throughput increase, and an INT8 TensorRT optimized model has 1318.15% throughput increase [31]. This indicates that if a less precise mode is used, the model's processing speed increases.

F. EMBEDDED DEVICES

Finally, the choice of the embedded device used to implement the system is also crucial. Some of the popular edge devices are NVIDIA's Jetson platform, and custom hardware implemented in FPGAs. In [43], a Hardware-Software codesign approach is used to make a DNN hardware accelerator, reaching 181ms inference time. For Jetson devices such as Jetson Nano, Jetson Xavier NX, and Jetson Xavier AGX, Jetson Xavier NX has the best performance in terms of FPS/Watt, Jetson Nano has the best performance in terms of FPS/$ [44], and Jetson Xavier AGX has the best performance in terms of calculated FPS [45].

III. BASIC DEFINITIONS AND PROPERTIES

In this section, some basic definitions and properties that are needed to implement this work are discussed. The fundamental concepts that are going to be discussed are voice feature extraction, Convolutional Neural Networks (CNN), and Long Short Term Memory (LSTM). This section aims to give a step-by-step understanding of building our proposed method.

A. VOICE FEATURE EXTRACTION

One of the most popular signal feature extraction is Spectrograms. Spectrograms are the squared magnitude of a Short-Time Fourier Transform (STFT) of a signal [46]. The STFT divides the time-domain signal into some frames. The frames are overlapping and transformed using Fourier Transform. The aim is to get the frequency information of each frame.

Definition 1: The STFT with continuous signal is defined as:

$$X(t,\omega) = \int_{-\infty}^{\infty} x(\tau)\cdot w(\tau - t)\cdot e^{-j\omega\tau} d\tau,$$

where $X(t,\omega)$ is the STFT output, $x(t)$ is a signal in the time domain t, $w(\tau - t)$ is the t-centered window function, and ω is the angular frequency. Note that e is an Euler's number.

Spectrograms represent signals in the time versus frequency domain. An example of the Spectrogram of a noisy sine wave is shown in Figure 2. In Figure 2, the X-axis of the Spectrogram shows the time, while the Y-axis of the Spectrogram shows the frequency. This means the Spectrogram shows which frequency component is present each time. The color of the Spectrogram, as shown in the color bar beside it, shows the magnitude of the corresponding frequency component at each time. From this Spectrogram, we can learn that the signal contains a base signal of 1, 2, and 3 kHz (indicated by a dark red line in 1, 2, and 3) with random noise added (indicated by the orange color in the rest of the frequency and time).

For voice feature extraction, Mel Spectrogram, which is a variant of the original Spectrogram, is a more suitable option. This is because the Mel Spectrogram uses the Mel scale [47] instead of the Hertz scale. Mel scale is a logarithmic scale and is close to how human hearing works.

Definition 2: The formula to convert frequency from the Hertz scale into the Mel scale is defined as:

$$Mel(f) = 1127 \times ln(1 + \frac{f}{700})mels,$$

where $Mel(f)$ is the Mel frequency output and f is the frequency in hertz.

The flowchart to create a Mel Spectrogram, compared to an original spectrogram is shown in Figure 3. In Figure 3, Spectrogram creation involves preprocessing, signal framing and windowing, Short Time Fourier Transform (STFT), and power spectrum calculation. The preprocessing step is often different, depending on the case. Some examples of preprocessing are noise removal, filtering, dc removal, etc. After the data is preprocessed, it is divided into frames, and a windowing function is applied to each frame. Framing and windowing are done to divide the input signal into short, stationary signals [48]. The formula to apply framing and windowing to the input signal is shown in Lemma 1. For voice feature extraction, Mel Spectrogram, which is a variant of the original Spectrogram, is a more suitable option. This is because the Mel Spectrogram uses the Mel scale [47] instead of the Hertz scale. Mel scale is a logarithmic scale and is close to how human hearing works. Furthermore, the formula to convert frequency from the Hertz scale into the Mel scale is shown in Definition 2.

Lemma 1: Furthermore, the framing and windowing for STFT with a discrete signal with N samples is represented as:

$$y(m,n) = \sum_{n=0}^{N-1} x(n + mH) \times w(n)$$

where $y(m,n)$ is the output of the framing & windowing step, $x(n)$ is the input signal, $w(n)$ is the windowing signal, N is the input signal length, n is time index, and m is frame index, and H is the hop size.

The parameters of this step are frame length, which controls the length of each frame, hop size, which controls how much portion of data from each frame will overlap, and window function type. One of the most common window functions, and the one that is used in this work, is the Hamming Window as described in Definition 3.

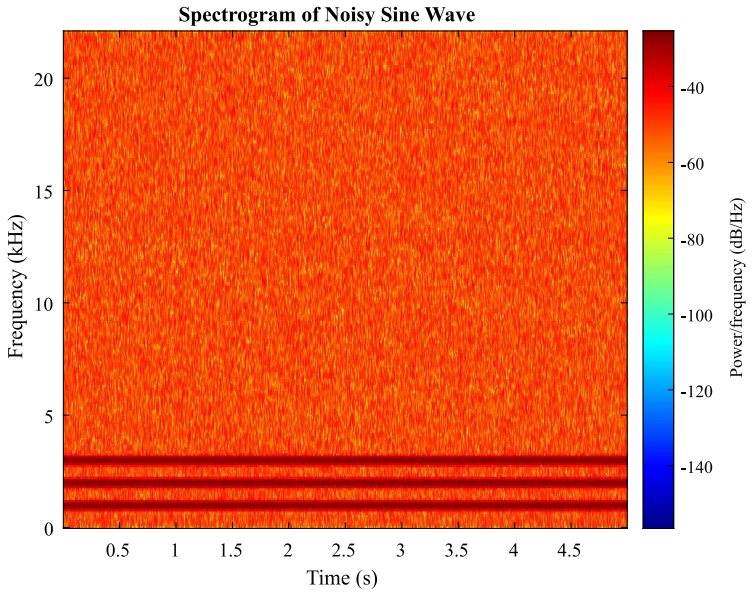

FIGURE 2. An example of spectrogram.

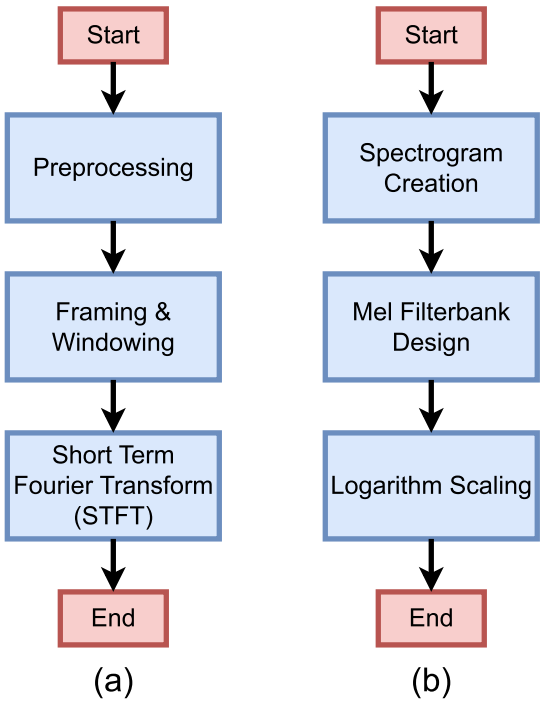

FIGURE 3. Flowchart of (a) Spectrogram creation (b) Mel Spectrogram creation.

Definition 3: The Hamming Window is represented by

$$w(n) = \begin{cases} 0.56 + 0.48 \times cos(\frac{2\pi n}{M}), & if\ 0 \leq n \leq N \\ 0, & otherwise \end{cases}$$

where $w(n)$ is the window function, n is the time index, and M is window length. Usually, it is the same as the frame length.

After the framing and windowing step, the Fast Fourier Transform (FFT) is computed for each frame. The computation of FFT for each frame.

Lemma 2: The formula to compute STFT with discrete signal is as follows:

$$S(m,k) = \sum_{n=0}^{N-1} x(n+mH) \times w(n) \times e^{(\frac{-i2\pi nk}{N})},$$

where $S(m,k)$ is the STFT of the signal, $x(n)$ is the input signal, $w(n)$ is the window signal, N is the frame length, H is hopping size, m is frame count, and k is time count.

Proof: Let $x(n)$ be a discrete signal with n as the time index. As Lemma 1, a localized segment of signal $x(n+mH)$ in a window signal $w(n)$ is obtained in $y(m,n)$ by:

$$y(m,n) = \sum_{n=0}^{N-1} x(n+mH) \times w(n).$$

Note that m is the frame count, H is the hopping size, and n is the current sample index.

Moreover, for transforming to the frequency domain, the $y(m,n)$ signal is applied to the Fourier transform complex exponent function represented by $e^{-j\omega\tau}$ in a continuous form and $e^{(\frac{-i2\pi nk}{N})}$ in a discrete form. If we multiply the $y(m,n)$ in Lemma 1 with the $e^{-j\omega\tau}$ as Definition 1, we have the following:

$$\begin{aligned} S(m,k) &= y(m,n) \times e^{(\frac{-i2\pi nk}{N})} \\ &= \sum_{n=0}^{N-1} x(n+mH) \times w(n) \times e^{(\frac{-i2\pi nk}{N})}. \end{aligned}$$

Thus, we have the lemma. □

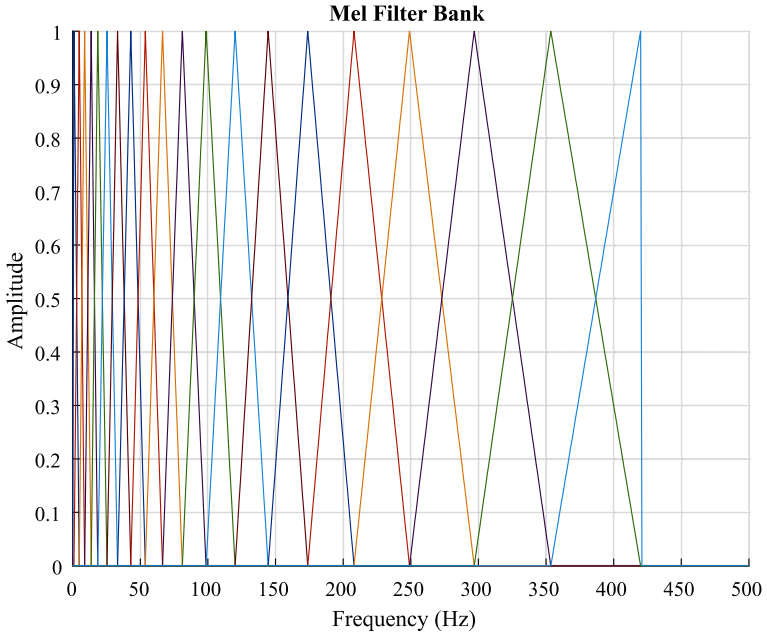

FIGURE 4. An example of mel filterbank used in this work.

Lemma 3: The last step of the Spectrogram creation is to compute the power spectrum. The power spectrum is represented by:

$$S_p(m,k) = S(m,k) \times S(m,k),$$

where $S_p(m,k)$ is the power spectrum and $S(m,k)$ is the STFT signal.

Mel Spectrogram's creation involves the creation of the original Spectrogram. After the Spectrogram of the signal is obtained, it is filtered using a Mel filter bank. A typical Mel filterbank is a set of triangular filters shown in Figure 4. To create this Mel filterbank, Lemma 4) is used [49] for determining the hopping size H.

Lemma 4: Let $f_a = f[m-1]$, $f_b = f[m+1]$, $f_d = (f_b - f_a)$, and $f_p = f[m]$, where f is a frequency in certain position and m is the frame index. Hopping size H is represented by:

$$H_m(k) = \begin{cases} 0 & ,k < f_a \\ \dfrac{2(k-f_a)}{(f_d)(f_p-f_a)} & ,f_a \le k \le f_p \\ \dfrac{2(f_b-k)}{(f_d)(f_b-f_p)} & ,f_p \le k \le f_b \\ 0 & ,k > f_b, \end{cases}$$

where $H_m(k)$ is the m^{th} filter bank, f is the frequency in mel, m is the filter index, and k is the time index.

Filtering is done by taking the dot product of the filter bank and the original Spectrogram filter. After the filtering is done, further logarithmic scaling is done to complete the Mel Spectrogram creation process.

B. CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Network(CNN) is a version of the Deep Learning model that has been widely implemented in image data problems, such as image classification. It uses a convolution process to replace the original matrix multiplication that a usual neural network uses [50]. CNNs work using a certain key feature in images, which is that nearby pixels in an image have bigger correlations than more distant pixels [51]. There are 3 key components that usually exist in a basic CNN, which are the Convolutional Layer, Pooling Layer, and Fully Connected Network Layer.

1) CONVOLUTION LAYER

A CNN layer computes the convolution of an input image with a smaller matrix called a kernel.

Lemma 5: The formula to compute this convolution is as follows:

$$S(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} I(i+m, j+n) \times K(m,n)$$

with $S(i,j)$ is the output matrix of the CNN layer, I is the input image, and K is the kernel. M and N is the size of the image where m and n is the corresponding indexes.

The adjustable parameters of this layer are kernel size and stride. The kernel size decides the size of the kernel used in the convolution process. Furthermore, the stride decides how many pixels are skipped after each convolution process. Some examples of the convolution process applied to an input matrix are shown in Figure 5. Figure 5(a) shows a convolution process with a stride. Moreover, Figure 5(b) shows a convolution process with two strides. All convolution process uses a 4×4 input matrix and 2×2 kernel.

In Figure 5(a), after the first convolution process in $I(0,0) = a$ is done, the next image submatrix that is considered is a 2×2 image submatrix starting from

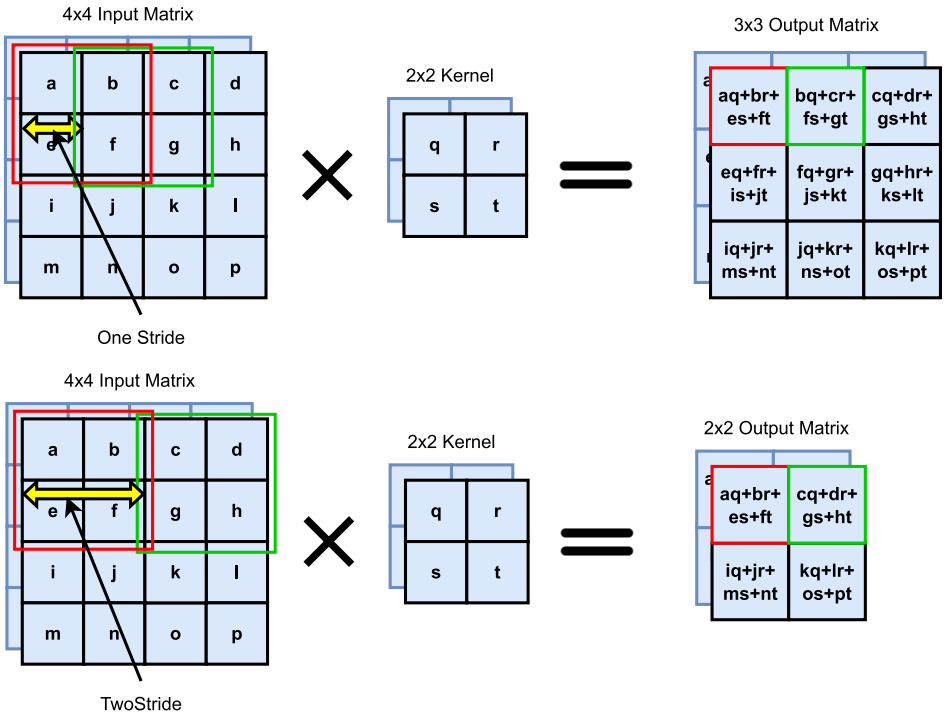

FIGURE 5. An example of convolution operation: One-stride (Top), Two-stride (Bottom).

$I(0, 1 = b)$. This is because the stride is 1 (yellow arrow). When the stride is 1, the kernel moves in the x and y direction with an increment of 1. In Figure 5(b), after the first convolution process in $I(0, 0) = a$, the kernel moves in the x direction with an increment of 2, taking the 2×2 image submatrix starting from $I(0, 2) = c$ because the stride is 2 (yellow arrow).

2) POOLING LAYER

The pooling layer is a special layer that replaces the value of the output image with a summary statistic of nearby outputs [50]. Some of the most popular pooling methods are maximum pooling and average pooling. The max pooling layer replaces a pixel value with the maximum pixel value within a rectangular neighborhood, while the average pooling layer replaces a pixel value with the average pixel value within a rectangular neighborhood.

Definition 4: A max-pooling layer is defined by:

$$S(i, j) = \max A, \tag{1}$$

where the element of array A is arranged by $A(m \times M + n) = I(i + m, j + n)$ *for* $m = 0, \cdots, M - 1$ *and* $n = 0, \cdots, N - 1$, *and* $S(i, j)$ *is the maximum value of an element in the matrix A for* $M \times N$ *input matrix.*

Lemma 6: An average pooling layer is defined by:

$$S(i, j) = \frac{\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i + m, j + n)}{m \times n} \tag{2}$$

where $S(i, j)$ *is the output of the average pooling layer, and I is an* $M \times N$*-sized input matrix.*

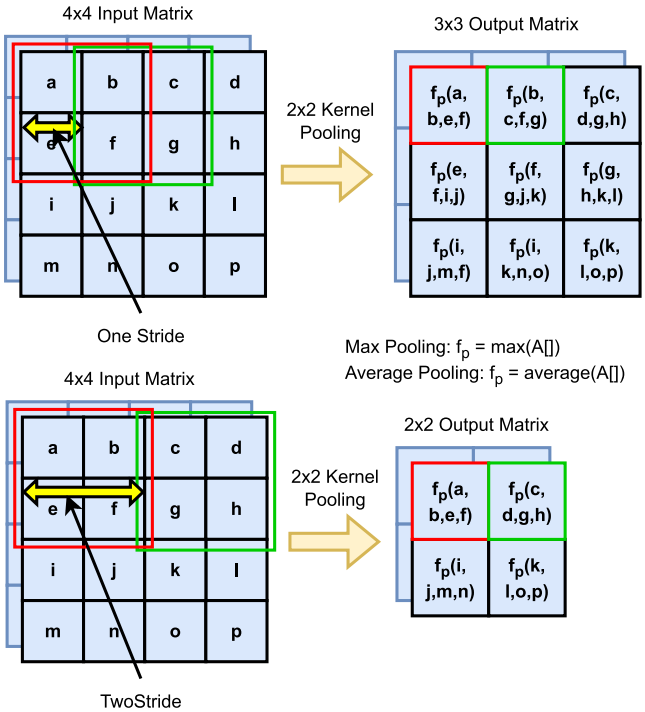

FIGURE 6. Max pooling and average pooling illustrations.

A visualization of a 2×2 max-pooling layer and average pooling is shown in Figure 6 and calculated using Lemmas 4 and 6, respectively. The stride indicates the amount of kernel movement for the next iteration. The input array arrangement of the max-pooling layer and the average pooling is the same with the corresponding pooling function f_p.

3) FULLY CONNECTED LAYER

The convolutional layer and pooling layer are usually used for image feature extraction. After some convolution and pooling processes, the image is often further classified using Fully Connected Layers. The fully connected layers are formed by the connected neurons. First, we define a neuron by the following definition:

Definition 5: A neuron in a fully connected layer is defined as:

$$y = f(\sum_{n=0}^{N-1} w_n x_n + b_n),$$

where y is the output of the neuron, x_n is an element of the input vector $X = [x_0, \ldots, x_{N-1}]$, w_n is an element of the weight vector $W = [w_0, \ldots, w_{N-1}]$, b_n is an element of the bias vector $B = [b_0, \ldots, b_{N-1}]$ and f is an activation function.

Furthermore, the fully connected layers can be implemented by a matrix multiplication. To compute the output of the layer, an activation function is used. The activation function is usually in the form of a nonlinear function that is applied to the matrix.

Lemma 7: The formula of the matrix multiplication is a weighted summary of the input vector, as follows:

$$Y = W \times X + B,$$

where Y is the output matrix, W is the weight matrix, X is the input matrix, and B is the bias matrix. After Y is computed, a nonlinear function is usually applied to it.

Some of the most popular nonlinear functions are sigmoid, tanh, softmax, ReLu, and LeakyReLu. The equation for each function is shown in Definitions (6), (7), (8), (9), (10), respectively.

Definition 6: A sigmoid(x) is represented by:

$$f(x) = \frac{1}{1 + e^{(-x)}},$$

where x is the input value and $f(x)$ is sigmoid function output value.

The sigmoid(x) maps the input to a value between 0 and 1. As shown in Figure 7(a) (blue), the sigmoid function forms an S-shaped curve.

Definition 7: A tanh(x) is represented by:

$$f(x) = tanh(x) = \frac{1 - e^{-2x}}{1 + e^{2x}},$$

where x is the input signal.

The tanh(x) converts the input signal to a value between -1 and 1. Similar to the sigmoid function, the tanh function also forms an S-shaped curve as shown in Figure 7(a) (red).

Definition 8: A ReLu(x) is represented by:

$$f(x) = max(0, x),$$

where $max(0, x)$ is a maximum function that selects the larger value between 0 and x. Note that if the value of x is negative or less than 0, the function selects 0 as the output value.

Definition 9: A LeakyReLu(x) is represented by:

$$f(x) = \begin{cases} ax & for\ x < 0 \\ x & for\ x \geq 0, \end{cases}$$

where x is the input signal, a is the gradient when the value of $x < 0$ and $f(x)$ is x when x is larger or equal than 0.

Definition 10: A softmax(x) is represented by:

$$f_k = \frac{e^{(x_k)}}{\sum_{j=0}^{J-1} e^{(x_j)}},$$

where x is the input signal in the form of vector, J is the number of classes, and x_i is the i-th element of the vector x.

The softmax activation function is commonly used for solving the problem of multiclass classifications. The softmax activation function converts the inputs into a probability distribution function, where the sum of all the outputs equals 1.

In summary, the sigmoid and the tanh convert the input into the range $[0, 1]$ and $[-1, 1]$, respectively. ReLu and LeakyReLu activation functions clip the negative values to 0 and ax, respectively, where a is the gradient when $x < 0$. Softmax, tanh, and ReLu activation functions are often used in hidden layers, while softmax is often used in the output layer to show the probability distribution function of each class in a classifier network.

Figure 8 shows an example of a Fully Connected Layer with a 4×1 vector input and 2 output cells. The output of each cell is a weighted sum of the input and bias, then a nonlinear function f is applied to the result.

Example 1: In Figure 8, X is;

$$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix},$$

W is;

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \\ w_{4,1} & w_{4,2} \end{bmatrix},$$

and B is;

$$B = \begin{bmatrix} b_1 & b_2 \end{bmatrix}.$$

By using Lemma 7, we have the following vector:

$$\begin{aligned} H &= \begin{bmatrix} h_1 & h_2 \end{bmatrix} \\ &= \begin{bmatrix} \sum_{1,j}(x_{1,j} w_{j,1} + b_1) & \sum_{2,j}(x_{2,j} w_{j,2} + b_2) \end{bmatrix} \end{aligned}$$

Finally, we have the output vectors:

$$\begin{aligned} Y &= \begin{bmatrix} y_1 & y_2 \end{bmatrix} \\ &= \begin{bmatrix} f(h_1) & f(h_2) \end{bmatrix}. \end{aligned}$$

where f is an activation function. ■

These activation functions are used in neural networks such as Long Short Term Memory (LSTM). Now, we explain the LSTM and its components.

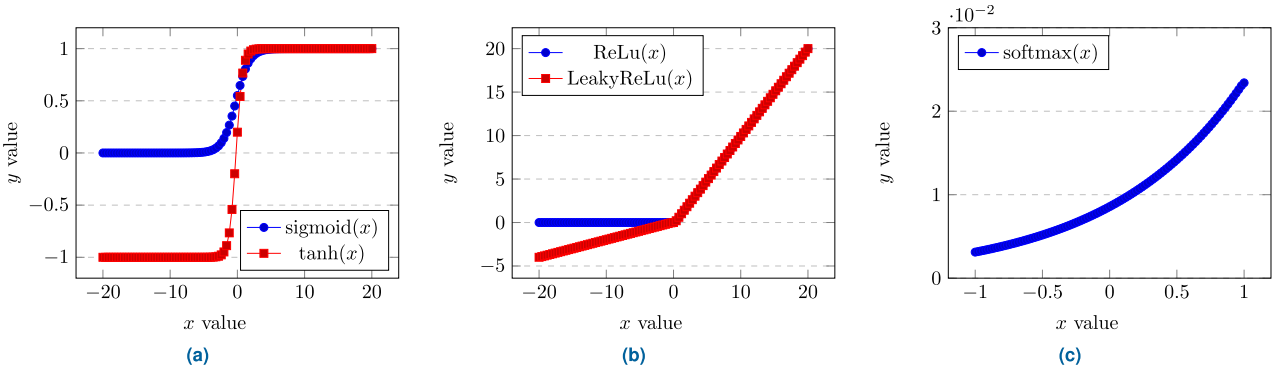

FIGURE 7. Several types of activation functions: (a) sigmoid(x) and tanh(x) functions, (b) ReLu(x) and LeakyReLu(x) functions, (c) softmax(x) function.

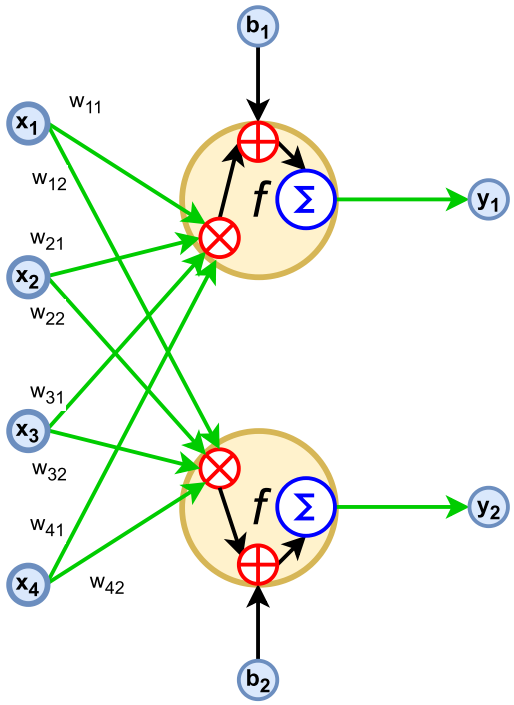

FIGURE 8. Fully connected layer example.

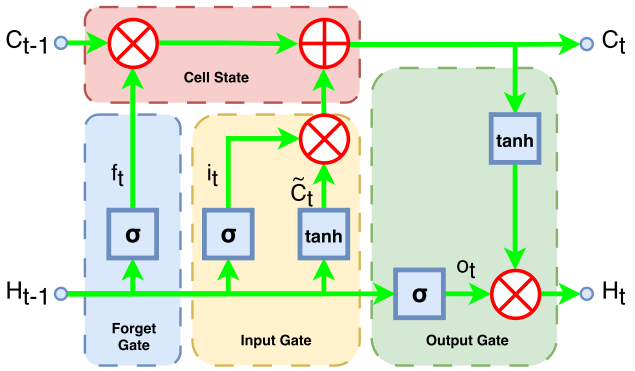

FIGURE 9. Block diagram of A LSTM cell.

C. LONG SHORT TERM MEMORY

Long Short Term Memory (LSTM) is a variant of Recurrent Neural Network (RNN) that consists of 3 gates, which are the input gate, forget gate, and output gate, and 2 states, which are cell state (long-term memory), and hidden state(short term memory). The block diagram of an LSTM is shown in Figure 9

The formulas that are used to compute the gate LSTM gates outputs are shown in Lemmas 8, 9, and 10 [17].

Lemma 8: The forget network is represented by:

$$f_t = \sigma(W_{fi} \times X_t + W_{fh} \times H_{t-1} + B_f),$$

where f_t is the forget gate, X_t is the input of the activation function, W_{fi} is the weight vector connecting the forget gate to the input, W_{fh} is the weight vector connecting the forget gate to the hidden state of the LSTM cell and B_f is the bias of the forget network.

Lemma 9: The input gate is represented by:

$$i_t = \sigma(W_{ii} \times X_t + W_{ih} \times H_{t-1} + B_i),$$

where i_t is the output gate, X_t is the input of the activation function, W_{ii} is the weight vector connecting the input gate to the input of the LSTM cell, and W_{ih} is the weight vector connecting input gate into the hidden state, and B_i is the bias of the input gate.

Lemma 10: The output gate is represented by:

$$o_t = \sigma(W_{oi} \times X_t + W_{oh} \times H_{t-1} + B_o)$$

where o_t is the output gate, X_t is the input of the activation function, W_{oi} is the weight vector connecting the output gate to the input of the LSTM cell and W_{oh} is the weight vector connecting the output gate to the hidden state of the LSTM cell and B_o is the output gate bias.

Lemma 11: The update of the cell state is represented by:

$$C_t = f_t \times C_{t-1} + i_t \times tanh(W_{ci} \times X_t + W_{ch} \times H_{t-1} + B_c)$$

where C_t is the cell state, X_t is the input of the activation function, and tanh() is the tanh activation function defined in Definition 7. W_{ci} and W_{ch} are the weight vectors connecting the cell state to the input and hidden state of the LSTM cell.

Lemma 12: The update of the hidden state is represented by:

$$H_t = o_t \times tanh(C_t),$$

where H_t is the hidden state, and tanh() is the tanh activation function defined in Lemma 7.

The effectiveness of LSTM over RNN is supported by the additional forget gate (Lemma 8) and output gate (Lemma 10) as well as the modified input gate (Lemma 9). In LSTM, the forget gate selects which state to be forgotten while the output gate controls the updated cell state to be exposed as the output.

Moreover, the update of the cell state as represented by Lemma 11 is updated using the forget gate, the input gate, as well as the output gate. The cell state retains the information as a long-term memory. The cell state is updated by selecting the information over time. Finally, the update of the hidden state as represented by Lemma 12 controls relevant information to be outputted at each time step.

IV. REAL-TIME KEYWORD SPOTTING USING SPECTROGRAM-BASED HYBRID CNN-LSTM (SPECTRONET)

The proposed SpectroNet that we propose in this paper is based on the works done in [52]. The concept that we took is the hybrid CNN-LSTM idea, the choice of DenseNet as the CNN architecture, and the one-dimensional convolution concept used in the DenseNet. However, we only use LSTM (not Bidirectional LSTM such as the work in [19]), and the sequence of layers design in the SpectroNet is our contribution. Lastly, we embed a Mel Spectrogram layer into the hybrid CNN-LSTM architecture to enable it to receive audio input directly and perform real-time performance. Finally, the proposed method explained in this section is divided into three parts. The first one is the deep learning model. The second one is the hardware and software specification. And, the third one is the dataset used.

Furthermore, the physical meaning of the proposed SpectroNet block diagram is explained as follows: The detail layer architecture of SpectroNet is described in Table 3 and Figure 10. The SpectroNet accepts audio data as its input. The audio data is then converted into a mel spectrogram in the mel spectrogram layer. The mel spectrogram is then preprocessed using a normalization layer, a 5×1 convolution layer, and a 2×2 average pooling layer. After the preprocessing stage is completed, the voice feature is extracted from the mel spectrogram using DenseNet with 3 Dense Blocks and a single LSTM layer. Once the feature vector is obtained, the output class is determined using a 64-neuron fully connected layer, a dropout layer with $p = 0.5$, and a softmax activation function, which gives the probability of the input data belonging to a class. The class with the largest probability will be taken as the output class.

TABLE 3. SpectroNet layers.

No.	Name	# Filter	Str.	Rep.	Out. Size
1	Input	-	-	-	16000
2	Mel Spec.	-	-	-	118×80×1
3	Norm	-	-	-	118×80×1
4	Conv.	10(5×1)	1×1	1	118×80×10
5	Pool	1(2×2)	2×2	1	59×40×10
6	Dense Block	60(1×1), 15(3x3)	1×1	2	59×40×106
7	Trans	15(1×1)conv, (1×2)pool	1×2	1	59×20×10
8	Dense Block	60(1×1), 15(3×3)	1×1	2	59×20×106
9	Trans	15(1×1)conv, (1×2)pool	1×2	1	59×10×10
10	bn-ReLu-conv	1(3×3)	1	1	59×10×1
11	Reshape	10(3×3)conv	1×1	1	59 × 10
12	LSTM	-	-	-	128
13	FC	-	-	-	64
14	Dropout	p=0.5	-	-	64
15	Softmax	10	-	-	10

A. DEEP LEARNING MODEL

The deep learning model that is used in this paper is SpectroNet, which is a low-complexity hybrid Convolutional Neural Network (CNN)-Recurrent Neural Network (RNN) model. This model has a total of 142,877 trainable parameters, which demonstrates its low complexity. To be able to handle audio data directly, this model has an embedded Mel Spectrogram Layer to eliminate the need to convert the audio input into Mel Spectrogram to extract features from the audio.

Some common deep learning classifiers need a separate process to compute Mel Spectrograms of each input data. This is inefficient, time-consuming, and resource-consuming. SpectroNet has its own embedded custom layer to compute Mel Spectrograms of each input data so audio input data can directly be fed and processed by SpectroNet.

The CNN architecture used in this model is a modified version of DenseNet, inspired by the works done in reference [22]. The RNN architecture used in this model is Long Short Term Memory (LSTM). The choice of Mel Spectrogram, DenseNet, LSTM, and hybrid CNN-RNN architecture are discussed in the next paragraph.

Mel Spectrogram is chosen as the audio feature extraction method and is implemented as a custom layer in the model. This is done because Mel Spectrogram is one of the most common audio feature extraction, especially for speech data. The Mel Spectrogram parameters chosen for this work are shown in Table 4.

TABLE 4. Parameters of mel spectrogram.

Parameter	Value
Window Type	Hanning
Window Size	512
Hop Size	128
# of Mel	80
Lower Frequency	0
Upper Frequency	8000

The hybrid CNN-LSTM model is used to overcome the limitations that CNNs and RNNs have when classifying audio

data. CNNs have the limitations of only learning spatial data, thus they have more layers to perform accurately when classifying speech Spectrograms [23]. Meanwhile, RNNs have the limitations of not learning the local information from speech Spectrograms [22]. In SpectroNet, the CNN model used is DenseNet, because it has feedforward layer connections that can help reduce the need for a deeper model [12]. The RNN model used in SpectroNet is LSTM, which is chosen because of its ability to combat the vanishing gradient problem that the original RNN has [53].

SpectroNet consists of an input audio layer, a Mel Spectrogram layer, a DenseNet with two Dense blocks, an LSTM network, a Fully Connected Network, and a softmax layer to output the prediction class. In detail, the layers of the proposed SpectroNet are shown in Table 3, while the block diagram of SpectroNet is shown in Figure 10.

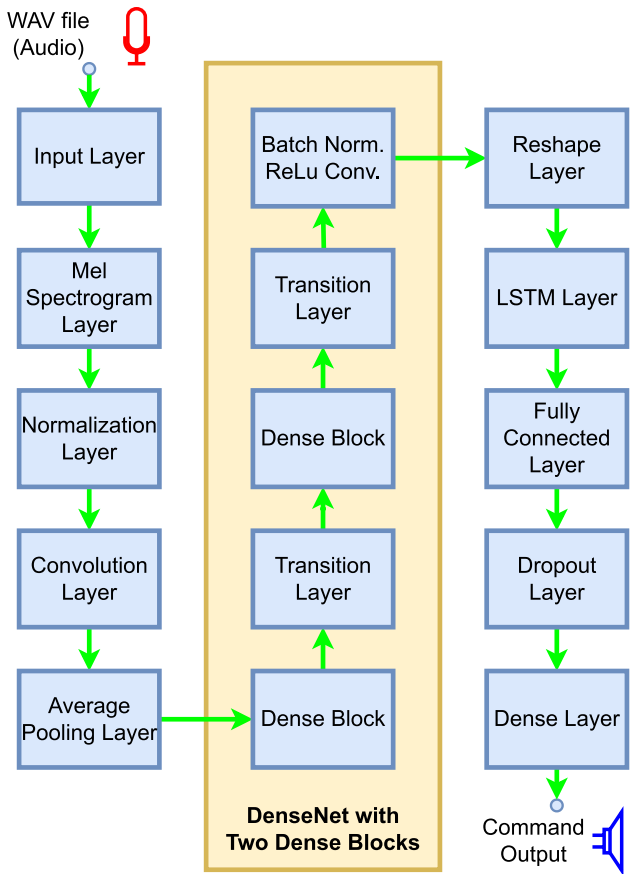

FIGURE 10. The proposed SpectroNet block diagram.

In Figure 10, SpectroNet uses 2 dense blocks. Each dense block consists of a batch normalization layer, a ReLu activation function, and two convolutional layers. And, each transition layer consists of a batch normalization layer, a ReLu activation function, a convolutional layer, and an average pooling layer.

The dense block and transition layer diagram are shown in Figure 11. A dense block consists of a sequence of BN-ReLu-Conv block and is ended by a convolution layer, while a transition layer consists of a sequence of BN-ReLu-Conv block, and is ended by an average pooling layer. The sequence of the BN-ReLu-Conv block stands for a batch norm layer, a ReLu activation function, and a convolution layer. The sequence of these layers is important because each layer has its own purpose. The batch Norm layer normalizes each batch of the input data before the nonlinearity activation function(ReLu) is applied. This is done to improve training stability and add regularization to the model. The ReLU Activation function gives nonlinearity to the data, making it easier for the convolution layer to extract its feature, and reduce the chance of overfitting. Lastly, the closing convolution layer and the average pooling layer are added as additional feature extraction and dimensionality reduction.

B. HARDWARE AND SOFTWARE SPECIFICATION

There are two hardware platforms that are used in this work. They are a Personal Computer (PC) and a GPU Board Jetson Xavier NX. The PC is used to create, train, validate, and test the SpectroNet using the dataset. Furthermore, the Jetson Xavier NX is used to implement the SpectroNet, optimize it using TensorRT, and perform a real-time inference using real human voice. The hardware specifications of the PC and the Jetson Xavier NX are shown in Table 5 and 6, respectively.

TABLE 5. Hardware specification of the PC used to implement SpectroNet.

Hardware	Specification
CPU	Intel(R)Core(TM) i7-9750H CPU
Clock Speed	2.60GHz
RAM	8092MB
GPU	NVIDIA GEFORCE GTX 1650
GPU RAM	4GB

TABLE 6. Hardware specification of jetson xavier NX.

Hardware	Specification
CPU	NVIDIA Carmel ARM®v8.2 64-bit CPU
# of CPU core	6
RAM	8 GB 128-bit LPDDR4x 59.7GB/s
GPU	NVIDIA Volta™ GPU
# of GPU core	384 with 48 Tensor Cores

In Table 5, the PC also has a GPU installed, this means that TensorFlow can be run on this PC in GPU-optimized mode. In Table 6, Jetson Xavier NX has a 6-core Arm CPU, 8GB RAM, and NVIDIA Volta GPU.

SpectroNet is implemented in both the PC and the Jetson Xavier using Python programming language. However, the TensorFlow version used in PC and Jetson Xavier is different. The PC uses the Windows 10 Operating System, so the maximum TensorFlow version that supports GPU optimization is 2.10.0. And, the Jetson Xavier NX uses Jetpack 5.1.2 TensorFlow version. Therefore, the maximum TensorFlow version that supports GPU optimization is 2.12.0. The recapitulation of the software specification of the PC and Jetson Xavier implementation is shown in Table 7 and 8, respectively.

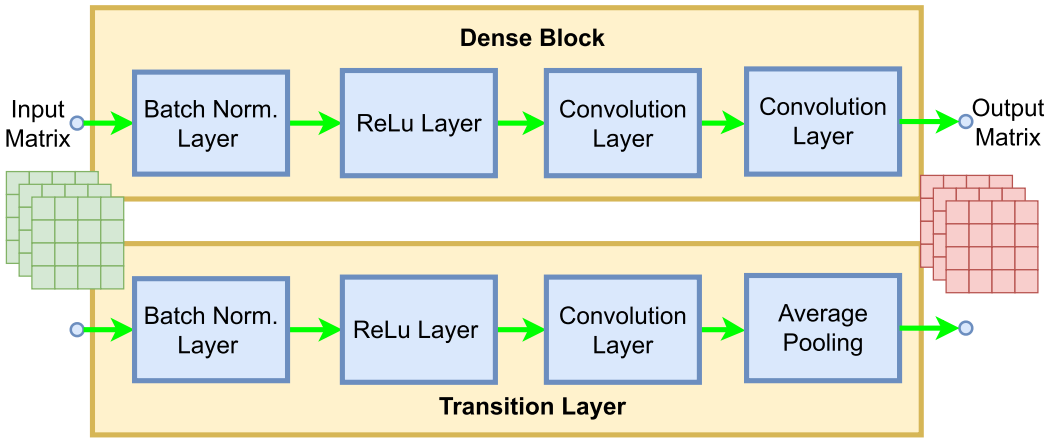

FIGURE 11. Dense block diagram and transition layer diagram.

C. DATASET

The dataset that is used to train, validate, and test the model is the Google Speech Commands dataset [26]. From that dataset, 10 words (out of 24 command words) are chosen. The spoken words are "go", "on", "off", "yes", "no", "up", "down", "left", "right", and "stop". From this selected 10 words, a total of 38,546 speech data is used. These data are then separated into 80% training data, 10% validation data, and 10% test data.

TABLE 7. Software specification of the PC used to implement SpectroNet.

Software	Specification
OS	Intel(R)Core(TM) i7-9750H CPU
Python version	3.9.13
TensorFlow version	2.12.0

TABLE 8. Software specification of jetson xavier NX.

Software	Specification
OS	Jetpack 5.1.2
Python version	3.9.0
TensorFlow version	2.12.0

D. BLOCK DIAGRAM OF THE SYSTEM

The work done in this paper can be explained by a block diagram, which can be seen in Figure 12. In Figure 12, a Jetson Xavier NX is used as the embedded device chosen to implement SpectroNet. A microphone is connected to the Jetson Xavier NX via a USB soundcard. 10 LEDs are connected to the GPIO port of the Jetson Xavier NX to indicate the output of the system.

The real implemented system also can be seen in Figure 12. In Figure 12, the system consists of a Jetson Xavier NX, a VGA Cable to output display to an external monitor, a USB sound card to provide an audio peripheral, a headset to provide audio Input and Output, and a set of LED as the class indicator.

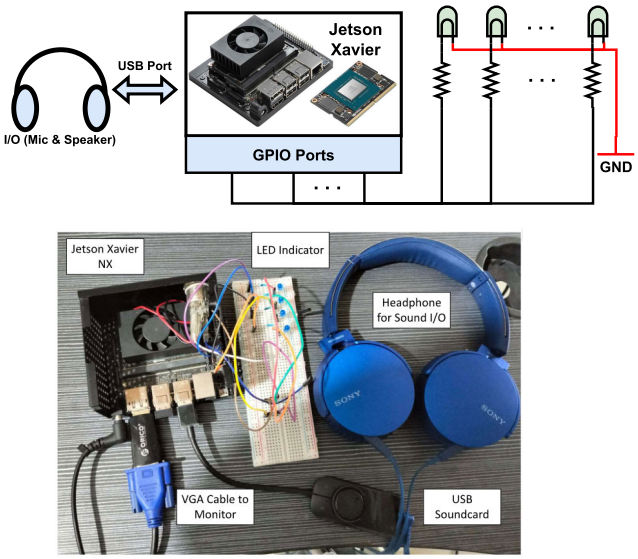

FIGURE 12. Implemented system for the proposed SpectroNet.

V. EXPERIMENTAL RESULTS AND DISCUSSION

In this paper, we divide the experiments into two phases. The first is deep learning model design. The second is deep learning model optimization using TensorRT. In the design phase, we design SpectroNet. It is trained, validated, and tested using a subset of Google Speech Commands Dataset v2. In this phase, the training accuracy, validation accuracy, training loss, validation loss, as well as the execution time of the corresponding metric will be observed. To further highlight the performance of SpectroNet, we trained several other deep learning models with various architectures and compared their performance. In the implementation phase, the trained SpectroNet model is implemented in the Jetson Xavier NX, and then the performance of the implemented model will be tested using a subset of Google Speech Commands Dataset v2. The testing accuracy, as well as the inference time of the system, will be observed. Finally, in the optimization phase, SpectroNet is optimized using the TensorRT library to speed up the inference time. The testing

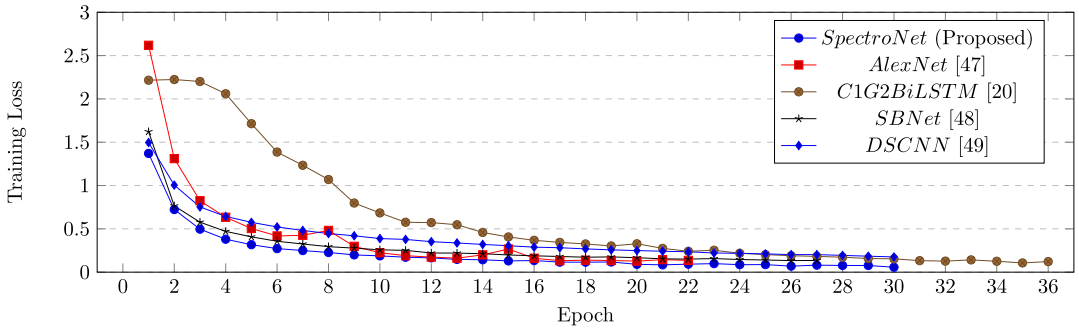

FIGURE 13. Training loss vs epoch plot for various deep learning models.

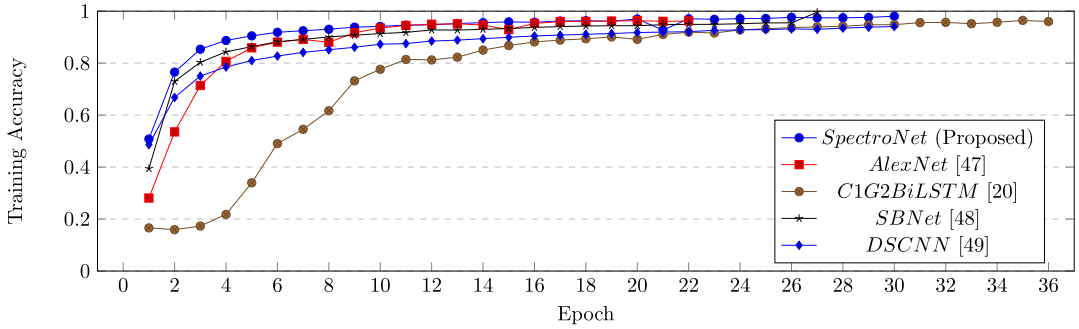

FIGURE 14. Training accuracy vs epoch plot for various deep learning models.

accuracy, as well as the inference time, will be observed in this phase.

A. DEEP LEARNING MODEL DESIGN

As stated in the opening of this section, the objective of this phase is to create a low-complexity CNN-LSTM model to do Keyword Spotting on the Google Speech Commands Dataset v2. To achieve this, SpectroNet was created. To further assess the performance of SpectroNet, we select several Deep Learning for Keyword Spotting models and train them with our dataset using as similar parameters as possible to the parameters stated in their respective papers. The models are AlexNet, which is the CNN model that we previously proposed in [54], SBNet, which is a CNN model that was created for environmental sound classification and is proposed in [55], DSCNN, a CNN model which uses depthwise separable convolution layer which is proposed in [56], and C1G2BiLSTM, a CNN-LSTM model proposed in [23]. They are all trained using the parameters stated in Table 9. The training phase is done using the PC.

TABLE 9. Training parameters used to train the deep learning models.

Parameter	Value
Max Epoch	100 epoch
Init. Learn Rate	0.001
Batch Size	64
Optimizer	Adam
Loss	Sparse Categorical Crossentropy
Val. Patience	10 epoch

In Table 9, all of the models are trained using the same parameters. All models are trained using Adam Optimizer with an initial learning rate of 0.001. The loss function used is Sparse Categorical Crossentropy, to match the class label that uses integer coding instead of one hot encoding. The data is batched with 64 batch size. Finally, all the models are trained for 100 epochs maximum, but validation patience of 10 epochs is set to avoid overfitting the model.

Table 10 shows the final result of the training and validation phase of each model. In Table 10, the training accuracy of each model reaches above 90% and their training loss is below 0.2. This indicates that all the models have good accuracy. However, there are overfitting problems. DSCNN has suffered the most from the overfitting problem, while AlexNet has suffered the least from the overfitting problem. The proposed model, SpectroNet, has the second lowest overfitting percentage of 2.84 %, only lost to AlexNet with 2.29% overfit.

In terms of the training epoch needed, all of the models never reach the maximum training epoch of 100%. AlexNet uses the least training epoch of 22 epochs, while C1G2BiLSTM uses the most training epoch of 36 epochs. The proposed model, which is SpectroNet, needs 27 epochs to complete training. This number is second least, being only more than AlexNet.

Figures 13, 14, 15, and 16 show the loss vs epoch and accuracy vs epoch of the Deep Learning Models during the training and validation processes. Figure 13 shows the training loss vs epoch plot while Figure 14 shows the training accuracy vs epoch plot. Figure 15 shows the validation loss

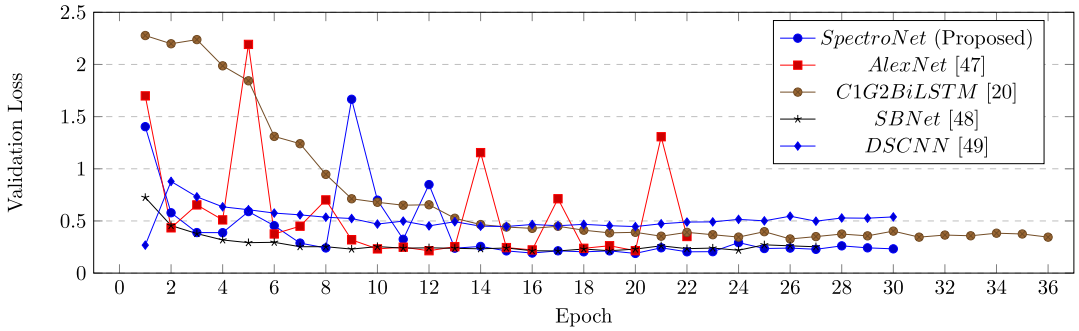

FIGURE 15. Validation loss vs epoch plot for various deep learning models.

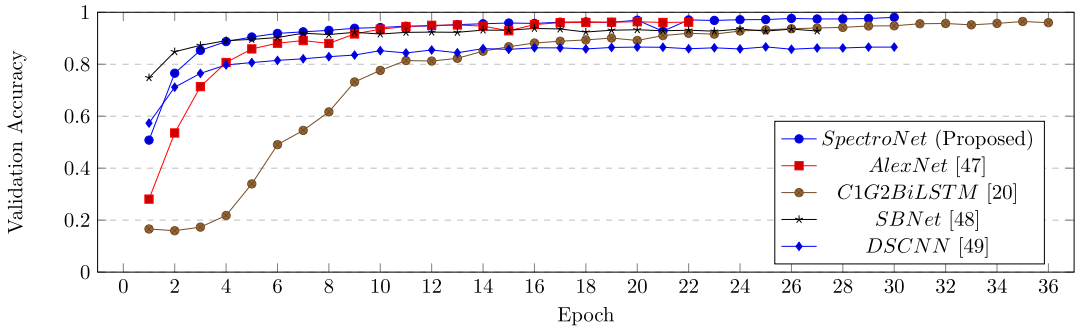

FIGURE 16. Validation accuracy vs epoch plot for various deep learning models.

TABLE 10. Training and validation result of each deep learning model.

Parameter	AlexNet [54]	SBNet [55]	DSCNN [56]	C1G2BiLSTM [23]	SpectroNet (Proposed)
Epoch Finished(epoch)	22	27	30	36	30
Training Loss	0.1360	0.1350	0.1759	0.1221	0.1360
Training Acc(%)	96.21	99.56	94.09	96.00	**98.05**
Val Loss	0.3509	0.2525	0.5388	0.3446	**0.2325**
Val Acc(%)	93.92	92.81	86.61	90.65	**95.21**
% overfit	2.29	6.75	7.68	5.35	**2.84**

TABLE 11. Model performance comparison of various deep learning models and SpectroNet.

Ref.	Model	Test Acc(%)	Param.	Inf Time(ms)
T.W.	SpectroNet	**93.33**	**89,241**	61
[54]	AlexNet	93.71	58,304,589	120
[55]	SBNet	92.32	874,266	60
[56]	DSCNN	85.40	36,320	30
[23]	C1G2BiLSTM	89.69	149,898	106

vs epoch plot and Figure 16 shows the validation accuracy vs epoch plot. In Figures 13 and 14, the training accuracy and loss of C1G2BiLSTM take the longest time to reach good value, while the training accuracy and loss of SpectroNet take the shortest time to reach good value. This also translates to their validation loss and accuracy. Moreover, the validation loss and accuracy of SpectroNet and AlexNet are not stable as depicted in Figures 15 and 16. However, training them with a lower initial learning rate, for example, an initial learning rate of 0.0001 increases their training time significantly while not improving their performance significantly.

After the training and validation phases are done, a testing phase is performed on each of the Deep Learning models. The test is done using Jetson Xavier NX. In this phase, Test Accuracy, along with Inference time, and total parameters needed in the model are compared. The comparison can be seen in Table 11.

As in Table 11, AlexNet has the best accuracy, while DSCNN has the worst accuracy. In terms of latency, DSCNN is the fastest model, while AlexNet has the biggest latency or inference time. This corresponds to their total number of parameters, with DSCNN having the least number of parameters and AlexNet having the most number of parameters.

Overall, AlexNet is the best-performing model, since it has the highest accuracy, fastest training time (in epoch), and least overfitting percentage. This further increases the superiority of AlexNet in image recognition problems for voice Mel Spectrogram images. However, AlexNet has a very high complexity. It has almost 60M total parameters and has the most latency. This makes AlexNet unsuitable for real-time keyword-spotting applications.

In contrast to AlexNet, DSCNN is the least complex model. It needs the least amount of total parameters and the least latency. However, It doesn't have a good performance in

terms of accuracy. The accuracy of DSCNN, especially its test accuracy, is very low, making it unreliable for the Keyword Spotting system.

Therefore, we select SpectroNet as the model for our real-time Keyword Spotting system. This selection is based on its good accuracy and low complexity. With 99.847% fewer parameters than AlexNet, it can achieve comparable accuracy, while having 28.57% less latency.

SpectroNet can perform with such high accuracy while having a low number of parameters because of its good feature extraction method. The DenseNet effectively limits the depth of the CNN needed to extract spatial data. It also uses one-dimensional convolution to avoid convolution along the time axis, thus it reduces the total number of parameters and avoids losing any temporal data that can be extracted using LSTM.

B. SPECTRONET OPTIMIZATION USING TENSORRT

To further optimize the performance of SpectroNet, further optimization technique needs to be applied. In this paper, TensorRT will be used to further optimize the SpectroNet. Optimization is performed using FP-32 and FP-16 precision. The summary of the optimization process is shown in Table 12.

TABLE 12. Optimization summary.

Parameter	TensorRT FP32	TensorRT FP16
% OP converted	52.57%	53.97%
# OP converted/total	(193/369)	(197/365)
Build Time	258.763 sec.	430.232 sec.

As shown in Table 12, TensorRT converts 52.57% of the total operation performed in SpectroNet when FP32 precision is used, and 53.97% of the total operation performed in SpectroNet when FP16 precision is used. This indicates that TensorRT has successfully converted more than 50% of the total operations performed in SpectroNet. Furthermore, TensorRT manages to reduce the total number of operations needed in SpectroNet from 369 to 365 operations when FP16 precision is used.

After the optimization is complete, the optimized model is tested using a real human voice. Each word is tested 10 times by the model, so there are a total of 100 real human voices tested. Model accuracy and inference time are observed. A comparison of optimization results is shown in Table 13.

In Table 13, the accuracy of the model slightly drops after optimization, but the performance drop is insignificant. Despite the performance drop, the inference time decreases quite significantly. FP32 optimization gives 10% speedup and FP16 optimization gives 14.75% speedup from the non-optimized model. This makes the FP16 optimization the best option for optimizing SpectroNet in Jetson Xavier NX. However, the downside of using FP16 precision is longer build time, as shown in Table 12, where FP16 precision needs approximately 66.29% more time to complete building the model. Finally, the real-time demo is demonstrated in a video that can be seen using this link https://youtu.be/1jDInCVdyvU.

TABLE 13. Optimization result comparation.

Param.	Testing Acc.	Inf. Time
Jetson(no TRT)	93.33%	61ms
Jetson(TRT,FP32)	93%	54.9ms
Jetson(TRT,FP16)	93%	52.0ms

VI. CONCLUSION

In this paper, we have successfully created an efficient deep-learning architecture called SpectroNet. SpectroNet reaches 93% real-time accuracy after being trained using a subset of the Google Speech Commands v2 dataset. This is a good performance considering it only uses around 70% less trainable parameters than other common deep learning architecture, making it a good choice for real-time implementation of Keyword Spotting systems. To improve the real-time performance of the network, a model optimization technique will be applied.

TensorRT is used in this paper to optimize the model. FP32 and FP16 precision is chosen as the data precision used in the optimization process. TensorRT successfully optimizes the model by speeding up the inference time by 10% if FP32 precision mode is used and 14.75% if FP16 precision mode is used. However, there is a slight accuracy drop of 0.33% after optimization. This drop is insignificant and the optimization process is considered successful and worth it. Compared to other software-implemented networks, optimized SpectroNet with FP16 precision has 40%-50% less inference time.

For future works, a specific topic may be chosen to be implemented using SpectroNet in Jetson Xavier NX. For example, a smart house system and music player can be good topic options. When applying this system to a certain topic, issues such as dataset creation and noise robustness should be addressed. Moreover, implementations in other platforms such as FPGA and IC are also very interesting to be explored.

REFERENCES

[1] I. López-Espejo, Z. H. Tan, J. H. L. Hansen, and J. Jensen, "Deep spoken keyword spotting: An overview," *IEEE Access*, vol. 10, pp. 4169–4199, 2022.

[2] A. Hurwitz Michaely, X. Zhang, G. Simko, C. Parada, and P. Aleksic, "Keyword spotting for Google assistant using contextual speech recognition," in *Proc. IEEE Autom. Speech Recognit. Understand. Workshop (ASRU)*, Dec. 2017, pp. 272–278.

[3] P. Vitolo, R. Liguori, L. Di Benedetto, A. Rubino, and G. D. Licciardo, "Automatic audio feature extraction for keyword spotting," *IEEE Signal Process. Lett.*, vol. 31, pp. 161–165, 2024.

[4] D. B. de Souza, K. J. Bakri, F. d. S. Ferreira, and J. Inacio, "Multitaper-mel spectrograms for keyword spotting," *IEEE Signal Process. Lett.*, vol. 29, pp. 2028–2032, 2022.

[5] J. S. P. Giraldo and M. Verhelst, "Hardware acceleration for embedded keyword spotting: Tutorial and survey," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 6, pp. 1–25, Nov. 2021.

[6] S.-G. Leem, I.-C. Yoo, and D. Yook, "Multitask learning of deep neural network-based keyword spotting for IoT devices," *IEEE Trans. Consum. Electron.*, vol. 65, no. 2, pp. 188–194, May 2019.

[7] H. Fuketa, "Time-delay-neural-network-based audio feature extractor for ultra-low power keyword spotting," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 2, pp. 334–338, Feb. 2022.

[8] C. Bernardo and L. E. Martins de Lima, "Anti-balance load control system applied to an overhead crane prototype activated by voice commands," *IEEE Latin Amer. Trans.*, vol. 19, no. 5, pp. 834–843, May 2021.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2012, pp. 1097–1105.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1–9.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[12] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016, *arXiv:1608.06993*.

[13] S. Roy et al., "Deep learning for classification and localization of COVID-19 markers in point-of-care lung ultrasound," *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2676–2687, Aug. 2020.

[14] V. Skaramagkas, A. Pentari, Z. Kefalopoulou, and M. Tsiknakis, "Multi-modal deep learning diagnosis of Parkinson's disease—A systematic review," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 31, pp. 2399–2423, 2023.

[15] X. Xu, S. Jeong, and J. Li, "Interpretation of electrocardiogram (ECG) rhythm by combined CNN and BiLSTM," *IEEE Access*, vol. 8, pp. 125380–125388, 2020.

[16] S. U. Amin, M. Alsulaiman, G. Muhammad, M. A. Bencherif, and M. S. Hossain, "Multilevel weighted feature fusion using convolutional neural networks for EEG motor imagery classification," *IEEE Access*, vol. 7, pp. 18940–18950, 2019.

[17] Q. Zhang, Y. Han, V. O. K. Li, and J. C. K. Lam, "Deep-AIR: A hybrid CNN-LSTM framework for fine-grained air pollution estimation and forecast in metropolitan cities," *IEEE Access*, vol. 10, pp. 55818–55841, 2022.

[18] M. A. R. Suleman and S. Shridevi, "Short-term weather forecasting using spatial feature attention based LSTM model," *IEEE Access*, vol. 10, pp. 82456–82468, 2022.

[19] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014.

[20] K. Hwang, M. Lee, and W. Sung, "Online keyword spotting with a character-level recurrent neural network," 2015, *arXiv:1512.08903*.

[21] S. O. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional recurrent neural networks for small-footprint keyword spotting," 2017, *arXiv:1703.05390*.

[22] M. Zeng and N. Xiao, "Effective combination of DenseNet and BiLSTM for keyword spotting," *IEEE Access*, vol. 7, pp. 10767–10775, 2019.

[23] D. Wang, S. Lv, X. Wang, and X. Lin, "Gated convolutional LSTM for speech commands recognition," in *Computational Science—ICCS 2018*, Y. Shi, H. Fu, Y. Tian, V. V. Krzhizhanovskaya, M. H. Lees, J. Dongarra, and P. M. A. Sloot, Eds. Cham, Switzerland: Springer, 2018, pp. 669–681.

[24] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Interspeech*, Sep. 2015, pp. 1–5.

[25] Y. A. Wubet and K.-Y. Lian, "Voice conversion based augmentation and a hybrid CNN-LSTM model for improving speaker-independent keyword recognition on limited datasets," *IEEE Access*, vol. 10, pp. 89170–89180, 2022.

[26] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018, *arXiv:1804.03209*.

[27] B. Kim, M. Lee, J. Lee, Y. Kim, and K. Hwang, "Query-by-example on-device keyword spotting," 2019, *arXiv:1910.05171*.

[28] A. Coucke, M. Chlieh, T. Gisselbrecht, D. Leroy, M. Poumeyrol, and T. Lavril, "Efficient keyword spotting using dilated convolutions and gating," 2018, *arXiv:1811.07684*.

[29] R. Leonard, "A database for speaker-independent digit recognition," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, vol. 9, Mar. 1984, pp. 328–331.

[30] A. Koubaa, A. Ammar, A. Kanhouch, and Y. AlHabashi, "Cloud versus edge deployment strategies of real-time face recognition inference," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 1, pp. 143–160, Jan. 2022.

[31] S. Manzoor, E.-J. Kim, S.-H. Joo, S.-H. Bae, G.-G. In, K.-J. Joo, J.-H. Choi, and T.-Y. Kuc, "Edge deployment framework of GuardBot for optimized face mask recognition with real-time inference using deep learning," *IEEE Access*, vol. 10, pp. 77898–77921, 2022.

[32] H. Yang, J.-H. Seol, R. Rothe, Z. Fan, Q. Zhang, H.-S. Kim, D. Blaauw, and D. Sylvester, "A 1.5-μW fully-integrated keyword spotting SoC in 28-nm CMOS with skip-RNN and fast-settling analog frontend for adaptive frame skipping," *IEEE J. Solid-State Circuits*, vol. 59, no. 1, pp. 29–39, Jan. 2024.

[33] C. Li, H. Zhi, K. Yang, J. Qian, Z. Yan, L. Zhu, C. Chen, X. Wang, and W. Shan, "A 0.61-μW fully integrated keyword-spotting ASIC with real-point serial FFT-based MFCC and temporal depthwise separable CNN," *IEEE J. Solid-State Circuits*, vol. 59, no. 3, pp. 867–877, Mar. 2024.

[34] F. Tan, W.-H. Yu, K.-F. Un, R. P. Martins, and P.-I. Mak, "A 0.05-mm^2 2.91-nJ/decision keyword-spotting (KWS) chip featuring an always-retention 5T-SRAM in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 59, no. 2, pp. 626–635, Feb. 2024.

[35] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*.

[36] M. Mohaimenuzzaman, C. Bergmeir, and B. Meyer, "Pruning vs XNOR-Net: A comprehensive study of deep learning for audio classification on edge-devices," *IEEE Access*, vol. 10, pp. 6696–6707, 2022.

[37] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2016, *arXiv:1611.06440*.

[38] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent.*, San Juan, Puerto Rico, Y. Bengio and Y. LeCun, Eds., 2016, pp. 1–14.

[39] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[40] C. Bucila, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2006, pp. 535–541. [Online]. Available: https://api.semanticscholar.org/CorpusID, doi: 10.1145/1150402.1150464.

[41] D. Kim, G. Kim, B. Lee, and H. Ko, "Prototypical knowledge distillation for noise robust keyword spotting," *IEEE Signal Process. Lett.*, vol. 29, pp. 2298–2302, 2022.

[42] P. H. Pereira, W. Beccaro, and M. A. Ramírez, "Evaluating robustness to noise and compression of deep neural networks for keyword spotting," *IEEE Access*, vol. 11, pp. 53224–53236, 2023.

[43] E. Manor and S. Greenberg, "Using HW/SW codesign for deep neural network hardware accelerator targeting low-resources embedded processors," *IEEE Access*, vol. 10, pp. 22274–22287, 2022.

[44] D. Kolosov, V. Kelefouras, P. Kourtessis, and I. Mporas, "Anatomy of deep learning image classification and object detection on commercial edge devices: A case study on face mask detection," *IEEE Access*, vol. 10, pp. 109167–109186, 2022.

[45] E. Güney, C. Bayilmis, and B. Çakan, "An implementation of real-time traffic signs and road objects detection based on mobile GPU platforms," *IEEE Access*, vol. 10, pp. 86191–86203, 2022.

[46] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 1999.

[47] S. S. Stevens, J. Volkmann, and E. B. Newman, "A scale for the measurement of the psychological magnitude pitch," *J. Acoust. Soc. Amer.*, vol. 8, no. 3, pp. 185–190, Jan. 1937.

[48] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ, USA: Prentice-Hall, 2009.

[49] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (Adaptive computation and machine learning). Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: https://books.google.co.in/books?id=Np9SDQAAQBAJ

[51] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). New York, NY, USA: Springer, 2007. [Online]. Available: http://www.amazon.com/Pattern-Recognition-Learning-Information-Statistics/dp/0387310738%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0387310738

[52] C. Amadeus, I. Syafalni, N. Sutisna, and T. Adiono, "SpectroNet: A low complexity CNN-LSTM architecture for keyword spotting application," in *Proc. IEEE 66th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2023, pp. 1060–1064.
[53] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2012, *arXiv:1211.5063*.
[54] C. Amadeus, I. Syafalni, N. Sutisna, and T. Adiono, "Digit-number speech-recognition using spectrogram-based convolutional neural network," in *Proc. Int. Symp. Electron. Smart Devices (ISESD)*, Nov. 2022, pp. 1–6.
[55] J. Salamon and J. Pablo Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," 2016, *arXiv:1608.04363*.
[56] Y. Lu, W. Shan, and J. Xu, "A depthwise separable convolution neural network for small-footprint keyword spotting using approximate MAC unit and streaming convolution reuse," in *Proc. IEEE Asia–Pacific Conf. Circuits Syst. (APCCAS)*, Nov. 2019, pp. 309–312.

INFALL SYAFALNI (Member, IEEE) received the B.Eng. degree in electrical engineering from Institut Teknologi Bandung (ITB), Bandung, Indonesia, in 2008, the M.Sc. degree in electronics engineering from the University of Science Malaysia (USM), Penang, Malaysia, in 2011, and the Dr.Eng. degree in engineering from Kyushu Institute of Technology (KIT), Iizuka, Japan, in 2014.

From 2014 to 2015, he held a research position with KIT. From 2015 to 2018, he was an ASIC Engineer with the ASIC Development Group, Logic Research Company Ltd., Fukuoka, Japan. In 2019, he joined ITB, where he is currently an Assistant Professor with the School of Electrical Engineering and Informatics and a Researcher with the University Center of Excellence on Microelectronics, ITB. In 2023, he was also a Visiting Scholar with the System Technology Co-Optimization (STCO) Program, Interuniversity Microelectronics Centre (IMEC), Leuven, Belgium. His research interests include logic synthesis, logic design, VLSI design, and efficient circuits and algorithms.

CLARENCE AMADEUS received the degree in electrical engineering from Maranatha Christian University, Bandung, Indonesia, in 2021. He is currently pursuing the master's degree in microelectronics major with Bandung Institute of Technology, Bandung. He is also an Active Laboratory Assistant with Maranatha Christian University. His research interests include embedded systems, digital signal processing, and voice recognition systems.

NANA SUTISNA (Member, IEEE) received the B.S. degree in electrical engineering and the M.S. degree in microelectronics from Bandung Institute of Technology, Indonesia, in 2005 and 2011, respectively, and the Ph.D. degree in computer science and electronics from Kyushu Institute of Technology (KIT), in 2017. From 2017 to 2020, he was a Postdoctoral Fellow with the Department of Computer Science and System Engineering, KIT. In 2019, he joined Institut Teknologi Bandung, where he is currently an Assistant Professor. He is also with the System Technology Co-Optimization (STCO) Department, IMEC Belgium, as a Visiting Researcher, working on an RL-based intelligent hybrid memory management system. His research interests include VLSI design, baseband wireless system design, AI processor design, and HW/SW co-design and co-verification.

TRIO ADIONO (Senior Member, IEEE) received the B.Eng. degree in electrical engineering and the M.Eng. degree in microelectronics from Institut Teknologi Bandung (ITB), Indonesia, in 1994 and 1996, respectively, and the Ph.D. degree in VLSI design from Tokyo Institute of Technology, Japan, in 2002. He is currently a Professor with the School of Electrical Engineering and Informatics and also the Head of the IC Design Laboratory, Microelectronics Center, ITB. He holds a Japanese patent on a high-quality video compression system. His research interests include VLSI design, signal and image processing, VLC, smart cards, and electronics solution design and integration.