Speech Keyword Spotting using Lightweight CNN, DNN

Jelin Raphael Akkara[†], Francisco Ariel Pasian[‡]

Abstract—**Speech-related technologies are becoming increasingly popular, with Keyword Spotting (KWS) still posing a sizeable challenge. However, SoTA ranking models tend to prioritize accuracy over limiting model complexity, with models ranging from 250k to over 5M parameters. We propose an efficient model with 31k parameters, that attains a decent accuracy (∼93%) while maintaining low memory and computational loads. For this, we experiment with various approaches like Gaussian and Sobel Smoothening Convolutions, Non-Linear LoRA Layers and Channel Attention Blocks. This model can be scaled up to attain SoTA accuracy scores, which requires further analysis.**

Index Terms—**Speech Recognition, KWS, LoRA, Convolutional Neural Networks, Dense Neural Networks**

I. Introduction

The field of speech recognition has undergone significant advancements in recent years, driven by the growing demand for voice-activated systems in applications ranging from virtual assistants to smart home devices. A critical aspect of these systems is Keyword Spotting (KWS), which involves the detection of specific keywords within continuous speech. It is integral for voice-controlled systems that are enabled using predefined trigger words (as is the case with Google Now that uses 'Ok Google'). These well-recognized systems utilize speech recognition to facilitate user interaction, making the need for efficient and accurate keyword spotting (KWS) systems crucial.

A keyword spotting (KWS) system must continuously operate with minimal power consumption, especially important for portable devices like smartphones and wearables. To ensure an optimal user experience, the KWS system should detect spoken keywords with high accuracy and low latency. To quantify and evaluate this challenge, the Google Speech Commands Dataset [1] has established itself in the landscape to rank models based on their accuracy scores. However, while models like KWT-3 [2] and KW-MLP [3] give close to a 97% accuracy score, the former is ranked higher even though the latter model is lighter by 35 times. This calls for evaluation under a better metric that evaluates a model by its efficiency over memory and inference time, even at the cost of a slight accuracy drop.

With the growing demand of compact and efficient systems, and its reliance on quick keyword detection, there is rising need to make lightweight KWS models. We intend to design, implement and evaluate a lightweight CNN-based model for effective keyword recognition. Motivated by [4], we do a comparative analysis of DNN and CNN, proposing that CNNs are better suited to be lightweight as they process spatially-correlated data better. It is shown that there is a parameter-explosion in the case of DNNs when using MFCCs as inputs, which is not desired.

With regards to an efficient CNN model, we pay attention to both pre-processing and architecture, hoping to maintain low parameter and multiplies counts. During pre-processing, we experiment with creating a compact and concise input that is easily understood by the model. Having a compact input helps in decreasing the FLOPs count significantly, making the model more lightweight. We try approaches like adding in white noise and smoothening using Gaussian and Sobel Convolutions, which result in correcting faulty inputs and bettering the convergence rate.

To limit parameters and multiplies while building the CNN architecture, we stick with using just two convolution layers [4] along with both strides and pooling methods. The input undergoes a rapid shrinkage as it goes through these two layers, and the information is captured by a high number of feature maps. We experiment with novel approaches like Non-Linear LoRA Layer, and Channel-Attention Blocks to enhance the model performance while keeping the counts at bay.

We evaluate our models on a subsection of the Google Speech Commands Dataset [5], consisting of 8 keywords, chosen manually to span all syllables. Performance is measure by categorical accuracy and ROC-AUC curves, in which the best model, cnn_spect_CAB, showcases 90% percent accuracy. To account for accuracy, parameter and multiplies counts, all at once, a new measure is defined as the accuracy divided by both the parameter and FLOPs counts. As per this measure, we find that our model (cnn_spect_CAB) trumps all other ranking models in the Google Speech Commands ranklist by one order of magnitude.

This work is structured as follows. In Section II we describe the related work and state-of-the-art models. In Section III, we show a high-level view of the processing pipeline. In Section IV, the pre-processing and feature extraction modules are covered in detail, along with the experimental setup and evaluation metrics used. In Section V, we detail the models used, and in Section VI, we produce the results after some comparative analysis. Section VII concludes the paper with a brief summary of the content and results.

II. Related Work

Research in Keyword Spotting (KWS) has prioritized models with both accuracy and efficiency, with much of the

[†]Department of Astronomy and Physics, University of Padova

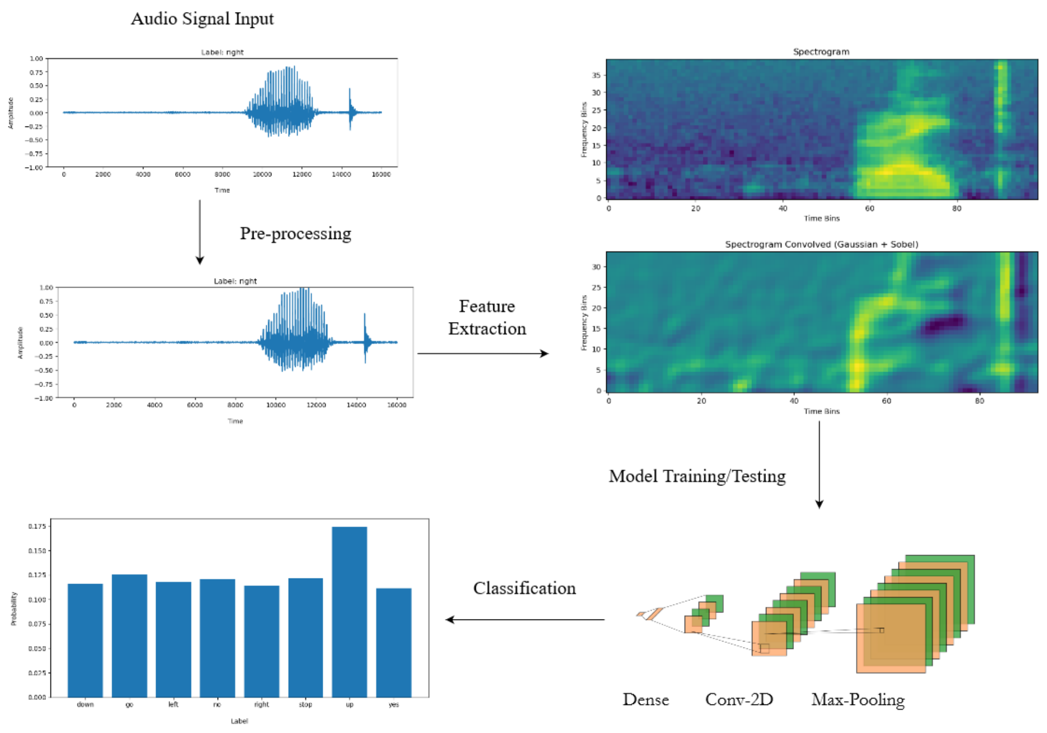

Fig. 1: Pipeline for processing audio signal, where the signal is pre-processed to obtain a clean signal, after which feature extraction is applied to obtain a spectrogram or MFCCs. This is used as input to the CNN or DNN model.

relevant progress occurring around the Google Speech Commands Dataset [5]. This has grown to become the standard benchmark in this task, using which models are compared and ranked. The dataset contains one second long audio clips, each containing a short keyword like 'No' or 'Go'. It has released two versions, one with 30 keywords and the other with 35 keywords. Each of these versions contain a 12 keyword sub-division, where it is required to classify 10 challenging keywords, along with two additional classes for 'unkown' (referring to unkown keywords) and 'silence' (referring to background or inaudible sounds).

The study of keyword spotting has a long history, beginning in the 1960s with early approaches based on Hidden Markov Models (HMMs), which dominated the field for several decades due to their ability to model temporal sequences in speech [6]. As the field evolved, the introduction of Deep Neural Networks (DNNs) in the 2010s marked a significant advancement, offering superior performance over traditional HMMs. However, this period saw the rise in compact mobile gadgets, which demanded much more lightweight models. Although DNNs provided good accuracy scores, they remained expensive and heavy on the systems using them. With works like [4], Convolutional Neural Networks (CNNs) became highly influential in KWS due to their ability to learn hierarchical features from spectrograms using much less parameters. However, the full capabilities of CNNs were still unexplored, with limited optimization of strides and pooling methods.

With the advent of Tranformers [7], and the success of the Vision Transformer (ViT) [8], patch-based transformer models with self-attention like the Keyword Transformer (KWT) [2], obtained state of the art results. These models often utilize knowledge distillation, making smaller versions more efficient and accurate. Recently, there has been a surge in reviving Multi-Layer Perceptron (MLP) models to imitate the attention mechanism while avoiding the troubles of high memory consumption. Models like KW-MLP [3] utilize gated MLPs [9] to inculcate spatial correlation information while remaining lightweight in nature, and manages to achieve performance scores similar to that shown by Transformer models.

In this paper, we try to bring together the advances of both the old and new models, and design a radically lightweight CNN model that utilizes channel-based attention and a few other methods to enhance its performance, while limiting its parameters to less than any of the above models.

III. Processing Pipeline

The raw audio signal is sampled at 16000 samples per second (through padding), normalized and gaussian white noise is added so that there is no sudden breaks in the signal. This helps keep the spectrogram conversion continuous and smooth. The labels are converted to one-hot encoding during this stage. After pre-processing, we extract the features by implementing short-time fourier transforms, obtaining either spectrograms or MFCC matrices (Fig. 1). For MFCC, additional features like the delta, delta-delta and energy coefficients are also optionally added. In case of spectrograms, to cut off the rough pixels and noise spread throughout, we use Gaussian convolutions to smooothen the image.

During Model training, we pass in the input and try limiting the architecture while enhancing its performance. In case of DNN, we use flattened MFCC matrices as input, while in case of CNN, we use either MFCC matrix or Spectrogram as input. However, we find using spectrograms is beneficial when dealing with CNNs as it still has spatially-correlated information contained in it, unlike MFCC matrices. The final layer of the architecture consists of 8 nodes (referring to the 8 classes used) which are passed through a softmax activation, giving us the probability scores for each label. In this manner, we obtain our predictions.

IV. Signals and Features

A. Experimental Setup

We consider a subsection of the Google Speech Commands Dataset [5], manually selecting 8 keywords that span through most english syllables: ['down', 'go', 'left', 'no', 'right', 'stop', 'up', 'yes']. While testing, we also consider other sets of keywords for robust evaluation. Each keyword consists of 3000 samples, maintaining equipartition among the classes, and we set the sample rate at 16k Hz. The dataset (of 24000 audio samples) is split into training (19200 samples), validation (2400 samples) and testing (2400 samples) sets. We use a batch-size of 64, and train the models for 50 epochs with early-stopping enabled (patience = 20).

B. Pre-Processing

Most audio samples are sampled at 16k Hz, with a few (0.09%) lower samples being zero-padded to maintain the sample rate. To avoid sudden breaks in the audio (like due to zero-padding), we add in a standard gaussian white noise with a maximum amplitude that is 100 times less than the audio peak. This helps further down the line, in making the spectrogram smooth and continuous (Fig.2). The audio data is also normalized to keep all samples consistent. As for the labels, they are converted to one-hot encoded vectors.

C. Feature extraction

We convert the pre-processed signal into either spectrogram or MFCC matrix using the Short-Time Fourier Transform (STFT). This divides the signal into time frames and conducts Fourier transform on each of these slices, after which the slices are concatenated together to give a time-frequency correlated matrix. We choose a 25 ms time frame (400 samples) and a 10 ms time stride (160 samples), as these are standard values corresponding with the average time taken for uttering a syllable.

The **Spectrogram** is derived by applying log-mel scaling to the STFT results, using 40 mel bins to limit the spectrogram size and computational load. The resulting matrix has 100 time bins and 40 log-mel bins, but it contains noise patches that

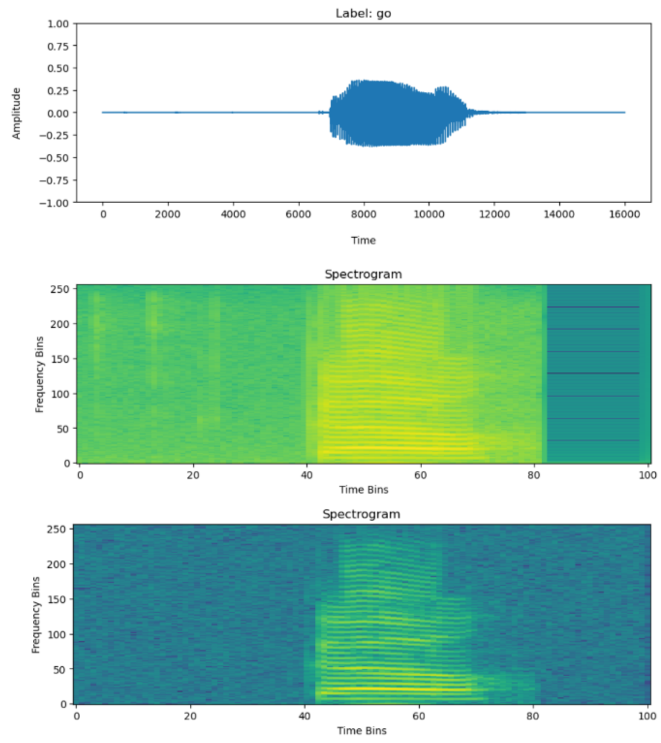

Fig. 2: We have the original audio signal that breaks off towards the end. Shown are the spectrograms before (middle) and after (bottom) adding white noise.

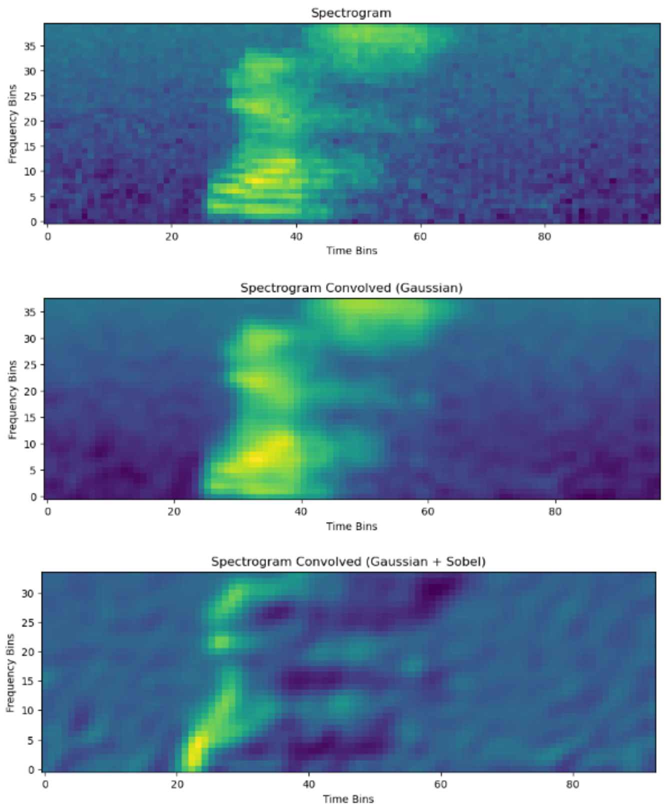

Fig. 3: Applying Gaussian and Sobel Filters to the Spectrogram to smooth out rough noisy patches.

need smoothing. We propose using Gaussian and Sobel convolutions to smooth these patches and enhance edges (Fig.3). A small Gaussian kernel (size = 2, std = 1) is chosen to smooth local noise without affecting the overall image. Sobel filters applied along both axes and adding the results, will highlight edges, possibly aiding model learning and convergence. To note, we use valid padding (no zero padding) to block out the hazy edges that occur due to convolutions. This reduces the spectrogram dimensions to (98, 38).

Mel-Frequency Cepstral Coefficients (MFCC) is a more compact representation of the short-term power spectrum, with each coefficient corresponding to characteristics like the energy, pitch and loudness of the audio within a time frame. The norm is to consider the first 13 coefficients, ignoring the first one. However, these 13 coefficients are confined to each slice, with the temporal information not accounted for. For this, there are two options. One, we can choose stacking, where we stack the previous and future time slices onto the current feature vector. However, this leads to unnecessary redundancy and an explosive input shape, both of which are not desired. Second, we can include additional features like the delta, delta-delta and energy coefficients which account for the rate of change of these features. Adding these additional features, the feature vector for one time slice is 39-dimensional. We get an output matrix of (98, 39), which is similar to the output we get from the spectrogram, (98, 38).

D. Evaluation Metrics

The model performance is evaluated using categorical accuracy, F1-score and ROC curves. Categorical accuracy refers to the mean of the accuracies per label (category). F1-score is the harmonic mean of precision (proportion of correctly identified keywords out of all predicted keywords) and recall (proportion of correctly identified keywords out of all actual keywords). ROC curve gives a diagrammatic view of the area under the curve, by which one can judge the model across all confidence thresholds.

Apart from the above, we propose a new model evaluation metric that includes the accuracy, parameter count and the FLOPs score (corresponding to the multiplies and additions), with an appropriate linear scaling of the denominator (like $1e11$). We define it as:

$$Model\_Score = \frac{accuracy}{(n\_param * FLOPs)}$$

These metrics offer a comprehensive evaluation of the performance of the proposed models, ensuring proper evaluation of the stability, accuracy and computational load of the model.

V. Learning Framework

We use two approaches in building a lightweight architecture for efficient keyword detection. One, we use a standard DNN and try various methods like stacking and adding delta features, and we show DNNs fail at being both compact and accurate. Second, we focus on developing a lightweight CNN that has low parameter and multiplies counts, even at the cost of slight accuracy drops. Our objective is to develop an efficient model, not just an accurate one. For this, we propose various methods like Non-Linear LoRA layers and Channel-Attention Blocks to enhance the performance while keeping track of the computational load.

A. Deep Neural Networks (DNN)

DNNs are feedforward neural networks with multiple hidden layers between the input and output layers. Although they have been successfully applied to speech recognition previously, they consisted of heavy models with less than par accuracy scores. This is because they do not inherently model the temporal and spatial structure of speech, which limits their performance. Here, we take a standard DNN architecture and show how the performance and parameters affect each other.

- **Input:** Flatten the MFCC matrix, eg. (98, 12) to (1176,), so it is accessible to the DNN.
- **Hidden Layers:** Three layers, each with 128 nodes and ReLU activation. Each of these layers are also paired with dropout (0.25) and L2-regularization, so as to limit overfitting.
- **Output Layer:** Made of 8 nodes (corresponding to the 8 labels), and has a softmax activation, providing a probabilistic output.

This model has a **parameter count of 184k**. When trying to increase the feature vector size through enhancement methods like stacking or additional features like delta, delta-delta and energy features, we see that the model size explodes (Fig.4). We notice that the only feasible enhancement is to concatenate only delta features (ignoring delta-delta and energy), but we further show that this does not result in a significant increase in performance.

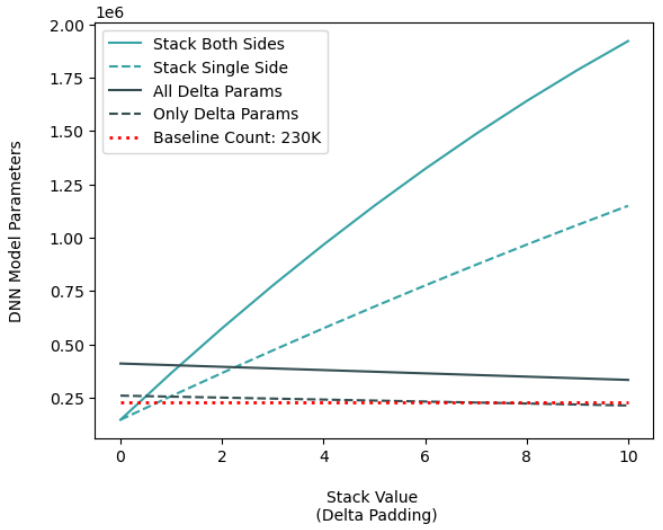

Fig. 4: DNN model size increases with stacking (in both cases). Concatenating just delta features remains close to the baseline parameter count, which seems to be the only feasible enhancement. Here, delta padding refers to number of time slices we consider before and after the current slice.

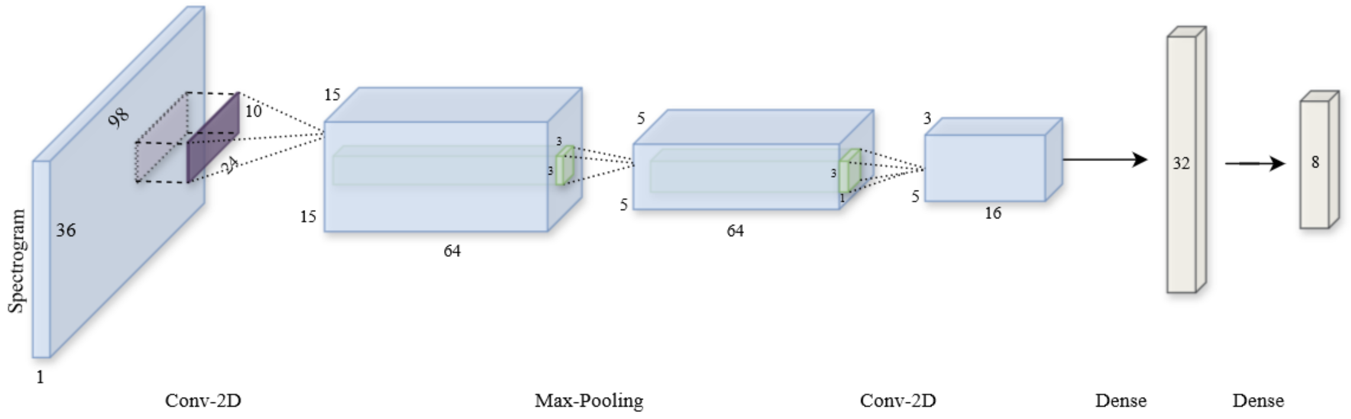

Fig. 5: Pipeline for cnn_spect model, consisting of two Convolution Layers and two Dense Layers.

Thus, in the case of DNN, good accuracy performance comes at the cost through a parameter increase. This is because it needs much more parameters to model the temporal and spatial correlations, which the CNNs seamlessly do with much less parameters.

B. Convolutional Neural Networks (CNN)

We propose a simple and efficient baseline CNN model, cnn_spect, made of 26K parameters and it takes in spectrograms as input. The reasoning behind choosing this architecture will be explained further below, and all other enhancements and experiments will be done using this as a base model.

*1) **Baseline Model (cnn_spect)**:* We propose a quick and efficient shrinkage of the input using as few layers as possible. To limit the number of multiplies, we constrain the convolutional layers to two layers, along with using less filters paired with both pooling and striding. This shrinks the input massively from one layer to the next. In total, the model is made of 26,488 parameters. The architecture is as follows (5):

- **Input:** We use spectrogram as input, with input shape (96, 38).
- **Conv2D:** We use a kernel size of (24, 10), one-fourth of the image shape, and a stride (5, 2), that is one-fifth the kernel size. The kernel size corresponds to the average utterance time of a syllable ($\sim$250 ms), and the stride makes sure we don't miss too much information along time ($\sim$65 ms). This shape also focuses on the global trends along time axis and the local trends along the frequency axis, as the kernel is rectangular in nature. To keep the counts at bay, we settle for a value of 64 for feature maps. This outputs a shape, (15, 15, 64) and contains 15424 parameters.
- **MaxPooling:** To scale down on the input, thereby reducing the parameter and multiplies counts, we implement max pooling and reduce the input shape to one-fifth its size. Using a pool-size of (3, 3) and stride of (3, 3), as we do not desire any overlaps [4], we get an output shape of (5, 5, 64).
- **Conv2D:** Now that we have a rapidly condensed input, we do away with strides and pooling as this might result in losing too much information. We choose a kernel size of (1, 3) to focus more along the frequency axis in this stage. We use 16 feature maps. A dropout value of 0.25 is added to avoid overfitting. This outputs a shape, (5, 3, 16) and contains 3088 parameters.
- **Dense:** The input is flattened, and fed into a dense layer of 32 nodes and gelu activation. This also contains a dropout of 0.25 to limit overfitting. This outputs a 32-dimensional vector and contains 7712 parameters.
- **Dense:** This is the output dense layer, made of 8 nodes and softmax activation, providing a probabilistic output and contains 264 parameters.

*2) **CNN with MFCC Model (cnn_mfcc)**:* Although using a spectrogram as input to CNN makes sense, we wanted to experiment on whether having MFCC matrix (with only delta features) as input would be helpful. The input should would be (94, 24). We use the baseline model architecture to build off, constraining ourselves to two layers of Convolutional layers, implementing L2 regularization and dropout (0.25) throughout. This model has a total of 226,424 parameters. The minor changes made are as follows:

- **Convolutional Layers:** Since each feature values are unique and relevant, we don't stride or pool along the frequency axis. Also, the first layer's kernel size is changed from (24, 10) to (24, 6) so as to focus more on the local values along the frequency axis.
- **Dense Layers:** We use an additional dense layer of 64 nodes, right after the input is flattened. This is done to ease the flow of information.

*3) **Non-Linear LoRA Layer**:* We propose a new layer that updates weight matrices using a low-dimensional space, possibly making the model lighter computationally. Suppose we have a 128-dimensional feature vector, and we usually pass it through a dense layer of 64 nodes. This would have a (128, 64) shaped weight matrix. Suppose we are not able to afford this layer due to high parameter load, we can use the non-linear LoRA layer to still use 64 nodes with a drop in

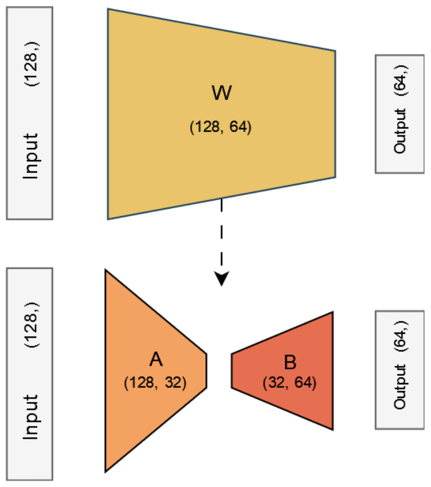

Fig. 6: Replacing weight matrix with low-rank matrices

parameter counts. This is done by replacing the weight matrix by two other matrices (A and B), each of a lower rank, say 32. Thus, matrix A would have shape (128, 32) and matrix B would be (32, 64). See FigBriefly put:

$$o = \sigma \left(I \cdot W + b \right) \quad (1)$$

$$o = \sigma \left(I \cdot \left(A \cdot B \right) + b \right) \quad (2)$$

Replacing Eq.1 with Eq.2, we get a non-linear low-rank dense layer. Here, I refers to the input (128,), b is the bias (64,) and σ is the activation function.

Thus, this layer takes in inputs as number of final units, lower rank value, and non-linear activation function. This allows us to utilize 64 nodes but update in the 32-dimensional space, for a slight increase in the parameter count. We propose that this enhances the performance of the model.

Insertion: This is used in cnn_spect and cnn_mfcc by replacing the dense(32) layer with LoRA(64, 32, 'gelu') layer.

*4) **Channel-Attention Block (CAB)**:* We propose an extension to Squeeze-and-Excite method [10], where instead of scoring each channel by its global average pooled value, we allow the model to learn the channel importance using a dense network. That is, we propose learning based on inter-channel information flow. In CNNs, the kernel weights are updated spatially across all channels but there is no inter-channel communication happening. Utilizing the latter can possibly improve performance by weighing the relevant channels more.

For this, we first make a copy of the input, say of shape (15, 15, 64), and it is passed through a normalization layer. It is then sent through a Dense layer with 64 nodes (same as number of channels) and Sigmoid activation. This layer only acts on the channels, with shared spatial weights for each channel. Each of the channels are connected to each of the dense node, enabling inter-channel information flow. The resultant tensor is of the same shape as the input, (15, 15, 64).

This new tensor is then multiplied with the original input, producing the output of the CAB. We think this action pushes the model to frame the copy as importance scores for each of the channels. Multiplying with the input then weights the channels accordingly.

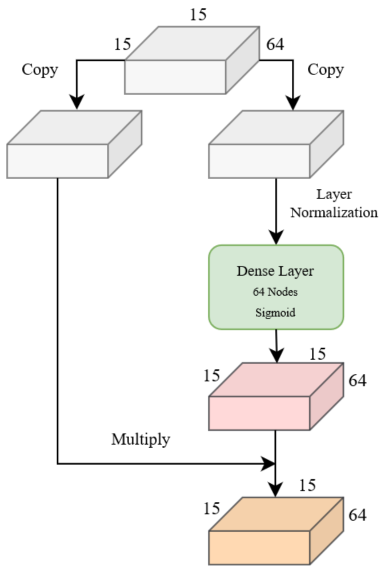

Fig. 7: Pipeline for Channel Attention Block

Insertion: We insert the CAB right after each Conv-2D layers.

VI. Results

A. Comparing Smoothening Filters

On Fig. 8, we have a comparison of smoothening filters. We see that the gaussian filter and no filter are better than using sobel filter, likely due to the information loss while using Sobel Filters. This might mean that using no filter is a better approach than using Gaussian, however, the latter shows to be a more stable convergence. Thus, using Gaussian Smoothening provides us with a more stable convergence and reliable results.

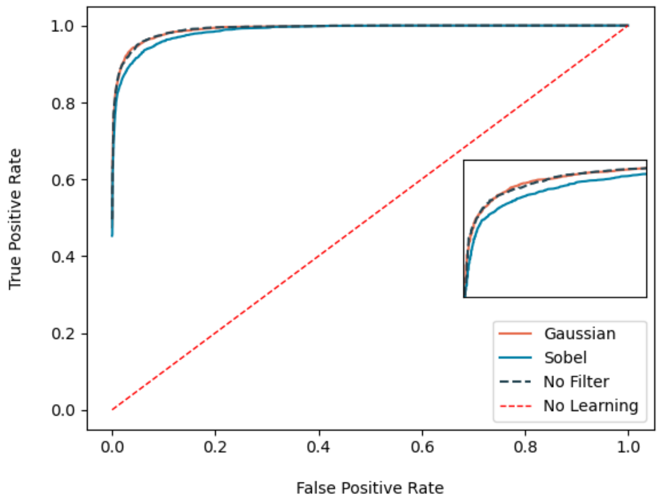

Fig. 8: Varying Smoothening Filters for cnn_spect

B. Comparing DNN Enhancements

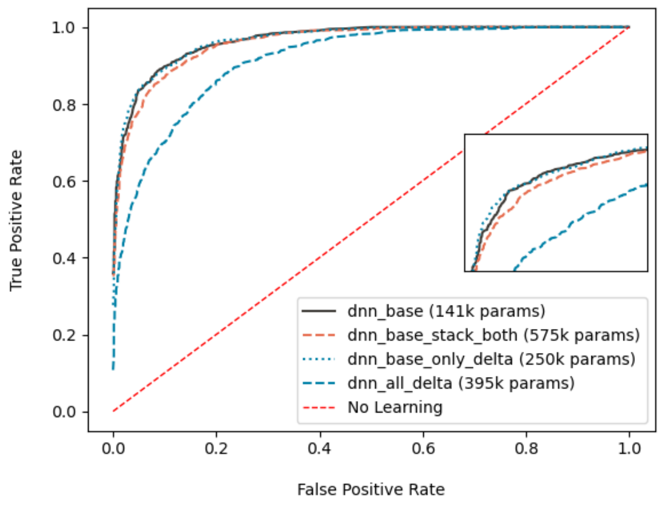

Fig. 9: Comparison of DNN Enhancements

The standard DNN architecture with minimal layers is not able to perform better even with enhancements. This calls for deeper dense architectures which is not desired as it increases the parameter counts. Our notions from Fig.4 are confirmed here by the results in Fig. 9.

Stacking does not improve performance, as it might make the model confused on the large feature vectors. Having a deeper network might be better for stacking, but that is not desired due to our constraints. Using all delta features results in a downgrade in performance, likely due to the large, compact, information-rich feature vectors which couldn't be processed due to the limiting depth and parameter count of the DNN. When using only delta features, which was the only feasible option as per Fig. 4, we don't see much improvement. All this tells us that DNN enhancements do not provide us with a model that is both accuracte and compact.

C. Using Non-Linear LoRA Layers

Firstly, the ROC curves (Fig. 10) for both cnn_spect and cnn_mfcc are similar, with the categorical accuracy scores also remaining similar, at 90% (± 0.16). However, note that the parameter count for cnn_mfcc is around four times higher. Thus, cnn_spect achieves similar accuracy with much less parameters.

Secondly, the LoRA layers do not seem to contribute much, although a slight improvement is noted. The accuracy values increase by 0.5%. This seems promising, and this layer might be better suited for large networks that require a necessary decrease in computational load. In lightweight models, this layer does not help.

D. Using Channel Attention Blocks (CAB)

From Fig. 11, there is significant improvement in performance, in both cases for cnn_spect and cnn_mfcc, when applying the CAB. For cnn_spect, the categorical accuracy

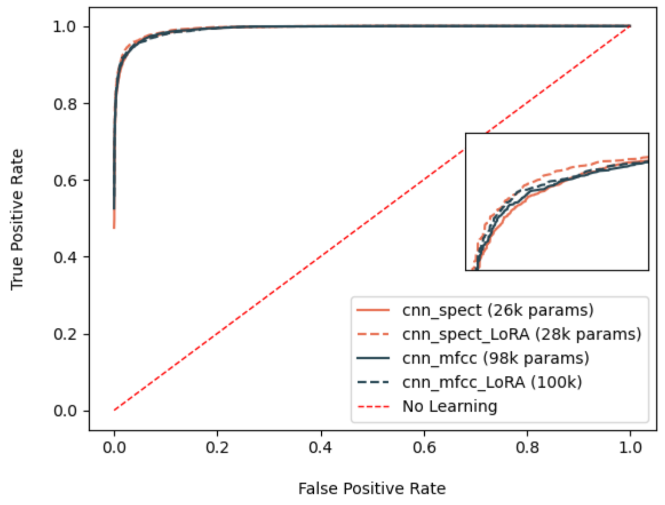

Fig. 10: Comparison of LoRA with cnn_spect and cnn_mfcc

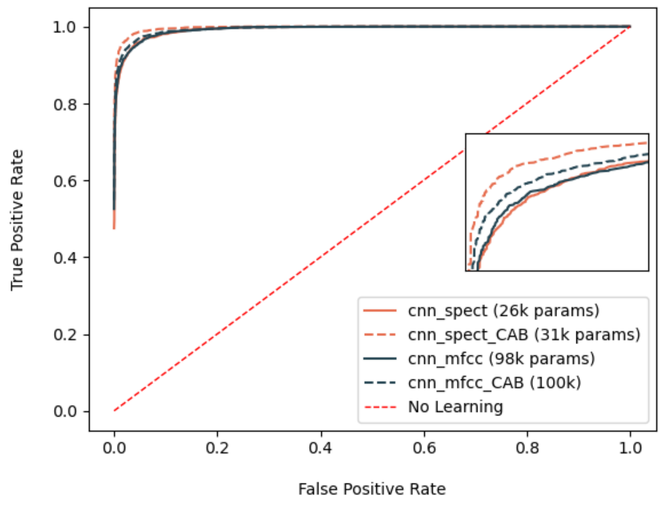

Fig. 11: Comparison of Channel Attention Block with cnn_spect and cnn_mfcc

increased to 93.8% (± 0.167). This shows that apart from the spatial analysis done using CNNs, pairing it up with a channel-based analysis helps improve performance.

E. Model Scoring and Time-Memory Analysis

The average inference time of running the cnn_spect_CAB model is **39.96 ms** and the memory usage is **90KB.**

Inference Time: Comparing to the reactive time for humans with auditory stimuli, which is 280 ms, our model performs satisfactorily. Compared to the state of the art KWS model [11], which has a measure of 20 ms, and the average inference time of a KWS model being 10-50ms, our model is up to the mark.

Memory Consumption: The average memory consumption tends to be a few hundreds of KBs [12], with the state of the art Google Model, Hello Edge, is designed to fit within 230KBs [12].

Model	Parameters (in K)	Accuracy	Score
cnn_spect_cab	31	93.8	**30.18**
Att-RNN	202	96.9	4.797
Res-15	237	98	4.135
KW-MLP	213	97.03	4.55
KW-1	607	97.72	1.61
KW-3	5361	98.54	0.184

TABLE 1: Model Scores using accuracy and parameters

Model Score: To quantify the model in all fronts, including accuracy, parameter count and FLOPs score, we use a new measure as mentioned in Sec.IV-D. Our best model, cnn_spect_CAB, provides a score of **32.36** while a SoTA model, MHAtt-RNN [13], occupying a rank of 10, obtained a score of **3.73**. This shows that when considering accuracy, parameter and FLOPs, our model trumps by one order of degree.

Since it is hard to find FLOPs scores for each model, while the parameter counts are easily available [3], we can modify the above measure to only include the accuracy and parameter count. Going by this measure, our model tops all other current ranking models, as given in Table 1.

F. Testing Model

Evaluating on the test set, we get a categorical accuracy of **92.5%** and an AUC of 0.9943. The confusion matrix is as given in Fig.12. We notice that the errors occur mostly with the syllable of 'o', with words 'go', 'no' and 'down' being the most confused ones.

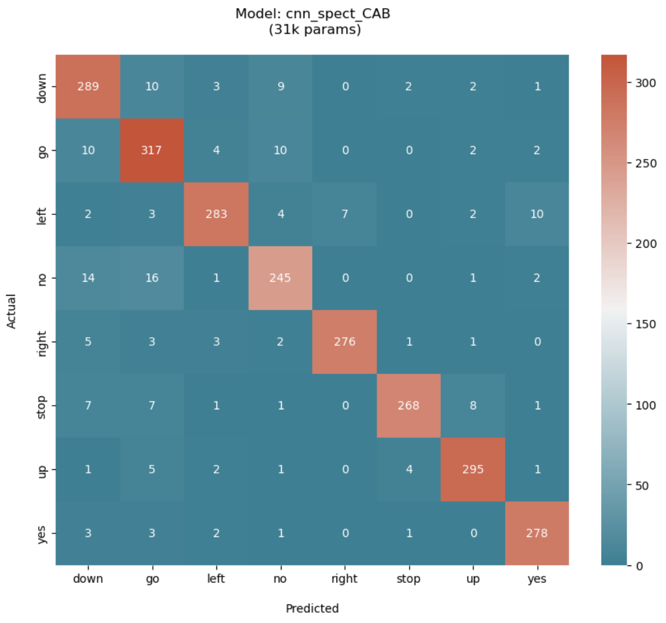

Fig. 12: Confusion Matrix for best model, cnn_spect_CAB

VII. Concluding Remarks

With the objective of building an accurate and lightweight model, we have cnn_spect_CAB which outperforms all current ranking models with respect to the defined model scores, in VI-E. It provides a categorical accuracy of 93.8% with merely 31,080 parameters. While using LoRA Layers did not improve performance, usage of Channel Attention Blocks resulted in a 4% accuracy spike.

Further analysis can be done to scale up the model and achieve higher SoTA accuracy levels, while keeping the parameter count and FLOPs as limited as possible. Using measures like the defined model scores (in Sec. IV-D) will help rank compact and efficient models higher and lead to development in this direction.

References

[1] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018.

[2] A. Berg, M. Connor, and M. T. Cruz, "Keyword transformer: A self-attention model for keyword spotting," in *Interspeech 2021*, interspeech˙2021, ISCA, Aug. 2021.

[3] M. M. Morshed and A. O. Ahsan, "Attention-free keyword spotting," 2022.

[4] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Interspeech 2015*, pp. 1478–1482, 2015.

[5] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," 2018.

[6] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164 – 171, 1970.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[9] H. Liu, Z. Dai, D. R. So, and Q. V. Le, "Pay attention to mlps," 2021.

[10] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2019.

[11] J. Yoon, D. Lee, N. Kim, S.-J. Lee, G.-H. Kwak, and T.-H. Kim, "A real-time keyword spotting system based on an end-to-end binary convolutional neural network in fpga," in *2023 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, pp. 1–3, 2023.

[12] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *CoRR*, vol. abs/1711.07128, 2017.

[13] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," in *Interspeech 2020*, interspeech˙2020, ISCA, Oct. 2020.