"Speech command recognition" Neural Networks for key word spotting

Arina Gepalova, †, Maren Hoschek‡

***Abstract*—The task of speech recognition is not new, but every day it becomes more and more popular and does not lose relevance. Applications range from healthcare to smart homes. However, whatever the task behind the application, it is based on the recognition of individual words spoken by a person. In this work, we will consider approaches to solving a basic problem - recognizing individual words.**

This paper presents an in-depth analysis of various neural network models for keyword spotting (KWS), emphasizing the comparison between Convolutional Neural Networks (CNNs) and Conformer models.

***Index Terms*—Keyword Spotting, Neural Networks, Convolutional Neural Networks, Conformers, Speech Recognition, Deep Learning, Ensemble Learning, Audio Processing.**

I. Introduction

The advent of voice-activated technologies has underscored the importance of efficient and accurate keyword spotting (KWS) systems. With a rich history of employing Convolutional Neural Networks (CNN) for this purpose, recent developments have seen the emergence of Conformer models as a potent alternative, promising enhanced performance by combining CNNs' spatial processing with the temporal dynamism of Transformer models. This paper explores the comparative effectiveness of these neural network architectures in KWS, examining their computational demands, accuracy, and deployment viability in devices with limited resources.

The paper is structured as follows: Section II discusses the current state of the art in keyword spotting and neural network models. Section III details the system setup and data models utilized in our study, while Section IV elaborates on the preprocessing and feature extraction methods. The architecture and implementation of the Convolutional Neural Networks and Conformer models are examined in Section V. Section VI presents the results of our experiments, focusing on model performance and efficiency. Finally, Section VII offers concluding remarks, summarizing the study's contributions to the field of speech recognition and suggesting directions for future research. All implementations of the described models, the results, and the trained models themselves are available in the project's GitHub repository [1].

II. Related Work

The field of speech recognition has seen significant advancements through various approaches to model architecture and dataset utilization.

Warden [2] introduced a dataset aimed at facilitating the training and evaluation of limited-vocabulary speech recognition models. This contribution is pivotal, as it provides a standardized benchmark for assessing the efficacy of keyword spotting systems under varying conditions, thereby enabling a more nuanced comparison of model performances.

Gulati et al.'s Conformer model [3] merges convolutional neural network (CNN) and transformer technologies to enhance speech recognition accuracy, demonstrating superior performance on challenging benchmarks compared to traditional models. This hybrid approach capitalizes on the convolutional layers' ability to capture local features and the transformers' proficiency in modeling global dependencies, setting a new standard for speech recognition tasks.

Similarly, Arik et al. [4] explored the synergy between convolutional and recurrent neural networks (CRNNs) to spot keywords in audio streams efficiently. Their investigation revealed that CRNNs offer a balanced mix of accuracy and computational efficiency, making them particularly suited for real-time applications where resource constraints are a critical factor.

Sainath and Parada [5] delved into the field of Convolutional Neural Networks (CNNs) for keyword spotting, establishing that CNNs not only reduce the model size significantly but also retain the precision offered by more complex architectures. This finding underscores the potential of CNNs in applications where memory and processing power are limited.

Drawing upon these foundational works, the present study tries to push the boundaries of speech recognition further.

III. Processing Pipeline

The processing pipeline begins with the data loading which is then followed by the data padding to ensure that each sample attains a consistent length of 16kHz, a mandatory step to guarantee uniform input shapes for our models. Subsequently, various optional data augmentation techniques may be employed. These techniques encompass the addition of background noises and resampling of the data, with the latter aiming to downsample and consequently diminish the data's dimensionality.

Following augmentation, feature extraction is undertaken. At this juncture, we employ several methodologies to convert the audio into a two-dimensional feature, which then serves as the input to our neural networks. The first method involves utilizing the short-time Fourier transform to convert the audio signal into a spectrogram [6]. The second method entails computing the log Mel spectrogram representation of the

†University of Padova, email: arina.gepalova@studenti.unipd.it
‡University of Padova, email: marenmichelle.hoschek@studenti.unipd.it

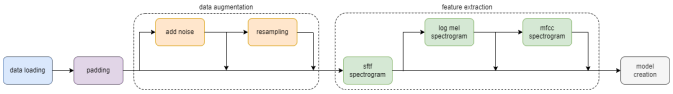

Fig. 1: Processing pipeline

audio signal. This process involves applying Mel filter banks to the spectrogram generated from the short-time Fourier transformation, thereby acquiring the Mel spectrogram representation [7].

The third method involves calculating the Mel-Frequency Cepstral Coefficients (MFCCs), which succinctly encapsulate the audio signal's power spectrum. The derivation of the MFCCs from the log Mel spectrogram representation involves implementing a discrete cosine transform [8]. Each of these transformations enhances the perceptual significance of the representation, transitioning from a strictly frequency-based perspective (spectrogram) to a representation that more closely aligns with human auditory perception (MFCCs), resulting in a more compact and distinctive feature set. The complete processing pipeline is depicted in 1.

IV. Signals and Features

A. Dataset

The Speech Commands dataset (version 0.02) comprises 105,829 audio clips of spoken English commands, each lasting no more than one second. These clips are saved as .wav files and feature a total of 35 different commands articulated by a diverse group of individuals. Of these commands, 25 are designated as keywords to be recognized by the keyword spotting system, while the remaining 10 are intended to be disregarded.

This dataset was assembled through a crowdsourcing initiative facilitated by AIY, a Google project, and is available under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. The recordings were obtained in various unstructured environments and subsequently standardized to a 16-bit, little-endian PCM format with a sampling rate of 16,000 Hz [2].

To ensure uniformity, the audio was processed to a consistent duration of one second, primarily using the 'extract_loudest_section' tool to capture the clearest portion of each utterance. Additionally, the dataset was curated to remove instances of silence or incorrect pronunciations. The finalized audio files were methodically organized into folders corresponding to the specific commands they represent [2].

B. Partitioning

The Speech Commands dataset includes two text files, validation_list.txt and testing_list.txt, which list the file paths for the respective validation and testing sets, with each file path appearing on a separate line. Files not listed in these text files are considered part of the training set [2]. To maintain a standard for comparison, we adhere to this predefined division of the data. Consequently, the validation and test sets each constitute roughly 10% of the entire dataset. The distribution is as follows:

- Training set: 80.17% (84,843 samples)
- Validation set: 9.43% (9,981 samples)
- Test set: 10.40% (11,005 samples)

In organizing the data into training, testing, and validation splits, we create a separate folder for each set. Within each folder, we maintain a structured subfolder system, where each subfolder corresponds to a specific utterance. This organization facilitates the straightforward retrieval and management of the dataset for machine learning applications.

C. Padding

The audio clips within the dataset are all one second or shorter, recorded at a sampling rate of 16kHz [2]. Our review of the dataset confirmed that there exist clips that are shorter than one second but none of the clips exceed one second in duration. Exception are the background noise audio samples there handling is described in subsection IV-D1.

To standardize the dataset for neural network training, we ensure that each audio clip is exactly one second in length by padding shorter clips. The majority of the audio clips consist of 16,000 samples, which corresponds to one second of audio at a 16kHz sample rate. We employ TensorFlow's padding function [9] to extend the length of shorter clips, adding zeros to both the beginning and end of the audio waveform to achieve a balanced and symmetrical padding. This process results in a consistent structure across all samples, facilitating efficient processing by the neural network.

D. Data augmentation

1) Background noise: The folder _background_noise_ contains six audio clips of recordings of real-world noise environments, as well as mathematical simulations of noise (pink noise and white noise). Integrating realistic background noise into training datasets is beneficial for developing machine learning models that are robust against noisy environments and helps to avoid overfitting [10].

To this end, we utilize a function named add_noise. This function randomly selects one of the six available noise recordings and blends it into the audio clip at a randomly chosen noise strength. We use different noise strength for real-world and mathematical noises because mathematical noises tend to be more pronounced and have a greater impact on the audio clip. Specifically, the volume level for real-world noises is chosen from a range between 0.2 and 0.5, while for mathematical noises, it is selected from between 0.1 and 0.3. Fig. 2 shows an example of how adding noise effects the samples.

Although the noise clips are sampled at the same rate of 16kHz, they exceed one second in length. As such, each time we add noise to an audio clip, we extract a random one-second segment from the selected noise recording. The resulting noisy signal is generated as follows:

$$\text{data\_noisy} = \text{data} + \text{noise\_strength} \times \text{noise\_segment} \quad (1)$$

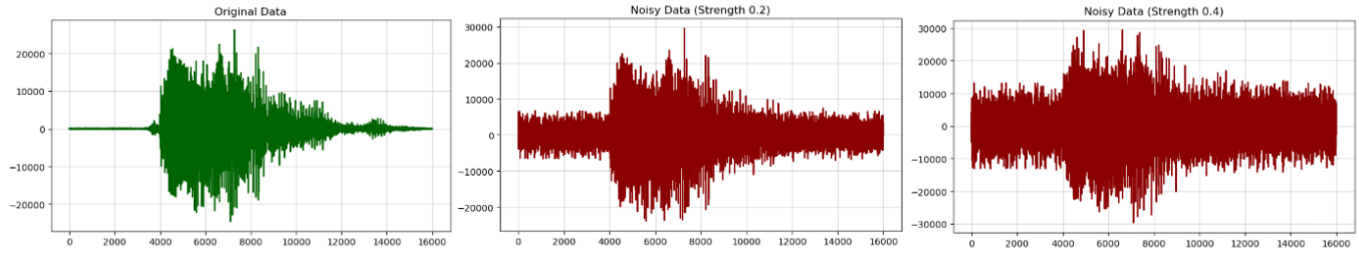

Fig. 2: Audio sample with white noise

2) Resampling: The concept of resampling in our context is implemented to decrease the feature dimensions within the dataset. We downsample the audio from an original frequency of 16kHz to 8kHz. This resampling rate is commonly used in telecommunication technologies and is known to preserve speech intelligibility for human listeners [11]. Our objective is to determine whether the reduced sampling rate of 8kHz will provide sufficient information for our neural network to perform effectively. To accomplish this downsampling, we employ the audio resampling method provided by TensorFlow I/O [12]. During the development of our models, we determined that downsampling significantly compromised our accuracy. Consequently, we ceased further exploration of this approach and excluded it from our final results.

E. Feature extraction

In the preprocessed dataset, the audio waveforms are initially in the time domain. To extract pertinent features from these waveforms, we utilize TensorFlow's tf.signal.stft, linear_to_mel_weight_matrix, and mfccs_from_log_mel_spectrograms functions. The features derived from these processes are used to construct the input vectors for the neural network models we are examining. We have adopted the following methods for audio feature extraction:

- STFT features
- Log Mel-filterbank energy features
- MFCC (Mel-Frequency Cepstral Coefficients) features

The stft features are genrated with a frame length of 255 and frame step of 128 making the spectrogram image almost squares. This appoach is suggested by tesnorflow for audio recognition [13]. For the log mel filterbank spectrogram and the mfcc the 2D feature vectors are generated by segmenting the audio signal into overlapping frames. Each frame is 25 milliseconds in length with a step size of 10 milliseconds between consecutive frames. This framing technique is consistent with the methodology applied by [4] in their analysis.

V. Model Architectures

A. Baseline model

As a baseline model, we utilized the approach presented in TensorFlow's "Recognizing Keywords" tutorial [13]. The model is a CNN with two convolutional layers. The detailed architecture and the number of parameters are outlined in Table 1. The input to the model is an STFT spectrogram with a frame length of 255 and a frame step size of 128, resulting in an input shape of (124x129). No data augmentation was performed on the input data and is done over 10 epochs only.

TABLE 1: Baselinemodel Architecture

Layer	**Output Shape**	**Param #**
Resizing	(None, 32, 32, 1)	0
Normalization	(None, 32, 32, 1)	3
Conv2D	(None, 30, 30, 32)	320
Conv2D	(None, 28, 28, 64)	18,496
MaxPooling2D	(None, 14, 14, 64)	0
Dropout	(None, 14, 14, 64)	0
Flatten	(None, 12544)	0
Dense	(None, 128)	1,605,760
Dropout	(None, 128)	0
Dense	(None, 26)	3,354
Total params:		1,627,933
Trainable params:		1,627,930
Non-trainable params:		3

B. CNN

Convolutional Neural Networks (CNNs) have emerged as a potent solution for small-footprint keyword spotting tasks, capitalizing on their ability to outperform Deep Neural Networks (DNNs) with significantly fewer parameters. CNNs are particularly suited for modeling the spectral representations of speech, which exhibit strong local correlations in time and frequency. Unlike DNNs, CNNs efficiently handle translational variances in speech signals, attributable to varying speaking styles, through shared weights across input regions and fewer parameters. This attribute makes CNNs ideal for applications demanding low computational resources while maintaining high accuracy, such as mobile devices' keyword spotting functionalities. Through innovative architectures, CNNs offer substantial improvements in false reject rates over DNNs, demonstrating their capability to deliver superior performance within the constraints of limited computational power and memory footprint.

In our case, the CNN model is designed as a hypermodel that can be used with hyperparameter tuning frameworks to

optimize its configuration for specific tasks. It is structured to process input data through convolutional and pooling layers, followed by dense layers, to perform classification tasks.

After performing hyperparameter tuning using Bayesian Optimization, the model has identified the optimal set of parameters to maximize its performance for the given task. The selected hyperparameters are:

m: 15 (Time filter size)
r: 6 (Frequency filter size)
n: 80 (Number of feature maps)
p: 2 (Pooling in time)
q: 2 (Pooling in frequency)

These parameters were selected to best capture the spatial and temporal features in the input data, which in this case have dimensions of 98x40x1.

Layer (type)	Output Shape	Param #
InputLayer	(None, 98, 40, 1)	0
Conv2D	(None, 84, 35, 80)	7,280
MaxPooling2D	(None, 42, 35, 80)	0
Conv2D	(None, 37, 30, 80)	230,480
MaxPooling2D	(None, 37, 15, 80)	0
Flatten	(None, 44400))	0
ConformerBlock	(None, 98, 128)	1,154,560
Dense	(None, 32)	1,420,832
Dense	(None, 26)	858
Total params:		1,659,450
Trainable params:		1,659,450
Non-trainable params:		0

TABLE 2: Architecture of the CNN Model

C. Conformer

A Conformer model is a type of neural network architecture that combines the strengths of CNNs and Transformers for handling sequential data. In the context of speech recognition, the Conformer model leverages the Transformer's ability to model long-range dependencies in the input audio signal, allowing it to understand the context of spoken words and phrases effectively. At the same time, it uses convolutional layers to capture the local acoustic features of the speech signal, such as phonetic nuances and temporal variations [3]. Fig. 3 shows the architecture of a conformer model.

A single Conformer block employs stacking of four different models for its operation: a feed-forward module, a self-attention module, a convolution module, and a second feed-forward module. These components are systematically stacked following the approach presented by in [3]. Where for an input x_i the output y_i is obtained in the following way.

$$\begin{aligned} \tilde{x}_i &= x_i + \frac{1}{2}\text{FFN}(x_i) \\ x'_i &= \tilde{x}_i + \text{MHSA}(\tilde{x}_i) \\ x''_i &= x'_i + \text{Conv}(\tilde{x}_i) \\ y_i &= \text{Layernorm}\left(x''_i + \frac{1}{2}\text{FFN}(x''_i)\right) \end{aligned} \tag{2}$$

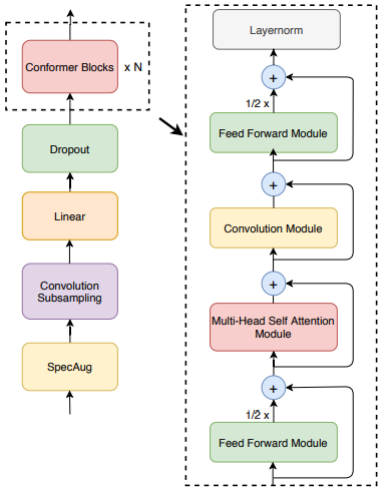

Fig. 3: Conformer architecture [3]

Conformer models have achieved very good results in speech recognition, albeit with a noted increase in complexity and computational requirements [14]. In the context of KWS, where it is crucial to maintain a balance between performance and computational efficiency, we have constrained our Conformer model to a maximum of two million parameters and limited its architecture to two Conformer blocks. To identify the most effective model configuration within these parameter constraints, Bayesian optimization was employed for the hyperparameters of the Conformer model. The obtained architecture of our Conformer model is detailed in table 3.

Layer (type)	Output Shape	Param #
InputLayer	(None, 98, 40, 1)	0
Conv2D	(None, 98, 40, 128)	2,176
BatchNormalization	(None, 98, 40, 128)	512
Reshape	(None, 98, 5120)	0
TimeDistributed	(None, 98, 128)	655,488
Dropout	(None, 98, 128)	0
ConformerBlock	(None, 98, 128)	1,154,560
GlobalAveragePooling1D	(None, 128)	0
Dense	(None, 26)	3,354
Total params:		1,816,090
Trainable params:		1,815,322
Non-trainable params:		768

TABLE 3: Architecture of the Conformer Model

D. Ensemble conformer

Deep neural networks are frequently utilized in ensembles to bolster prediction accuracy, a method that has proven effective in speech recognition [15]. Consequently, we explored the potential of enhancing accuracy by combining three Conformer models into an ensemble. Given that ensembling necessitates heightened computational resources because of the need to train three separate models. We imposed a limitation on the total number of parameters within the ensemble to

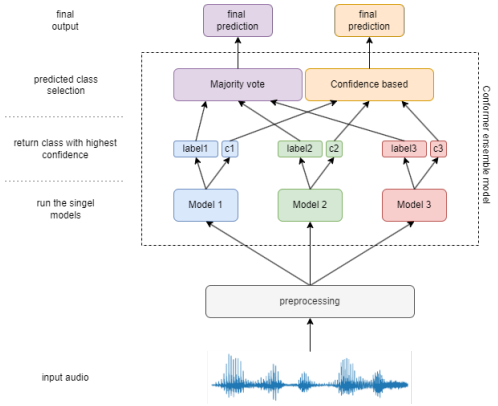

Fig. 4: Ensemble architecture

fewer than 1.8 million. This threshold is notably less than the parameter count of our single Conformer model. Our objective was to assess whether an ensemble of three smaller Conformers could surpass the performance of one larger Conformer model. To ensure diversity within the ensemble, each Conformer model was assigned distinct hyperparameters, preventing the models from becoming overly similar. The parameter distribution for the models is as follows:

- Conformer model 1: 640,106 parameters
- Conformer model 2: 489,626 parameters
- Conformer model 3: 373,490 parameters

For aggregating the predictions from our models, we investigated two approaches: majority voting and confidence-based prediction methods. The architecture of our ensemble is depicted in fig 4.

VI. Learning Frameworks

A. Baysian optimization

For the hyperparameter tuning of both the CNN and the Conformer model, we employ Bayesian optimization. This method is utilized to identify the optimal configuration of a black-box function that is analytically undefined and costly to assess. The optimization process unfolds sequentially, guided by data, with the aim of refining the estimation of the optimal point while minimizing the number of evaluations required from the black-box function [16]. For the implementation of Bayesian optimization, we utilized the Bayesian optimization functionality provided by Keras Tuner. This implementation is advantageous because it permits setting a maximum model size, which is particularly beneficial in KWS, as we aim to avoid overly costly models [17].

B. Loss metrics

For our model training, we employed sparse categorical crossentropy as the loss function. This decision was informed by the nature of our dataset and the specific requirements of multilabel classification tasks. The sparse categorical crossentropy loss function is particularly well-suited for scenarios where each instance is associated with a single label out of many possible categories, but the model needs to predict the probability of each category. This function efficiently handles the complexity of multilabel classification by calculating the loss between the predicted probabilities and the true distribution of labels. Its ability to process label data in a sparse format, where the label vector for each instance is a single integer indicating the class, rather than a full one-hot encoded vector significantly reduces memory consumption and computational cost, making it an ideal choice for our project. By utilizing sparse categorical crossentropy, we aimed to optimize the model's performance in accurately classifying instances into multiple categories.

C. Evaluation metrics

For the evaluation of our models, we selected precision, recall, F1 score, and accuracy as the primary evaluation metrics. These metrics were chosen for their ability to provide a comprehensive assessment of model performance from different perspectives. Precision measures the accuracy of the model in identifying relevant instances among the predicted labels, Recall evaluates the model's ability to identify all relevant instances, critical for not missing any crucial keywords. The F1 score combines precision and recall into a single metric, Accuracy, while more general, gives an overall success rate of the model across all classifications, offering a straightforward assessment of performance. Together, these metrics allow for analysis of a model's strengths and weaknesses in keyword spotting, where the correct identification of keywords (precision), the comprehensive detection of all relevant keywords (recall), and the balance between these aspects (F1 score) are paramount, alongside an overall effectiveness measure (accuracy). Additional we will evaluate the model in terms of GPU and CPU usage, the model complexity (number of parameters) and the time needed for training and testing.

VII. Results

A. Baseline

The baseline model, following the approach outlined in [13], achieves low accuracy at 0.3381, with other performance metrics also showing less than promising results. This model underwent training for only 10 epochs, mirroring the methodology employed in [13]. Further improvements might have been possible with additional training epochs. The CPU usage reached 11.402 GB, primarily due to the normalization layer preceding the actual model, while GPU resources were minimally utilized. Detailed statistics for the baseline model are presented in Table 4.

B. CNN

CNN model demonstrates efficient GPU utilization during both training and testing, with a reasonable training time and a quick testing phase, although the result for accuracy score is not high for the testing model.

C. Conformer

Hyperparameter tuning for the Conformer model was conducted using Bayesian optimization. The optimal hyperparameter combination identified within our search space, targeting models with up to 2 million parameters, was as follows:

Metric	Value
Accuracy	0.338119
Precision	0.548313
Recall	0.266889
F1 Score	0.241465
Number of Model Parameters	1,627,933
Max CPU Usage (Training)	11.402 GB
Max GPU Usage (Training)	0.372 GB
Total Training Time	164.835 seconds
Testing Time per Step	152 ms/step

TABLE 4: Model Performance and Resource Usage Baseline model

Metric	Value
Accuracy	0.50786
Precision	0.515534
Recall	0.487152
F1 Score	0.4864835
Number of Model Parameters	1,659,450
Max CPU Usage (Training)	3.8472 GB
Max GPU Usage (Training)	0.897 GB
Total Training Time	1014.637 seconds
Testing Time per Step	56.461 ms/step

TABLE 5: Model Performance and Resource Usage CNN model

- Model dimensions: 128
- Number of heads: 8
- Kernel size: 8
- Dropout rate: 0.3
- Number of Conformer blocks: 1

This configuration enabled the Conformer model to achieve an accuracy of 0.8526, demonstrating a significant improvement over the baseline model across various performance metrics. The model underwent training for 32 number of epochs, resulting in longer training times compared to the baseline model. One drawback is the increased consumption of GPU resources by the Conformer model. However, it utilizes fewer CPU resources due to the application of batch normalization instead of a complete normalization layer. Additionally, the model exhibits enhanced efficiency in test data prediction, requiring only 44 ms/step, which is less than half the time needed by the baseline model. This speed is a substantial advantage for keyword spotting KWS, where minimizing the latency between keyword utterance and detection is critical. Overall, the Conformer model markedly outperforms the baseline model. The higher GPU usage should not pose a problem if the model is not trained on the KWS device itself but is merely loaded onto or called by the device. If direct training on the device is necessary, a model with lower GPU usage may need to be considered. Detailed performance metrics are provided in Table 6.

Metric	Value
Accuracy	0.852612
Precision	0.886727
Recall	0.832624
F1 Score	0.849247
Number of Model Parameters	1,816,090
Max CPU Usage (Training)	3.984 GB
Max GPU Usage (Training)	9.682 GB
Total Training Time	3359.828 seconds
Testing Time per Step	44 ms/step

TABLE 6: Model Performance and Resource Usage of Conformer model

D. Conformer ensemble

We sought to enhance the performance of our Conformer model by developing an ensemble of Conformers. We experimented with a small ensemble consisting of three Conformer models. To determine the final prediction, we explored two methods: majority voting and confidence-based prediction. In our experiments, these two approaches demonstrated no difference in performance when limited to an ensemble of three models; hence, the subsequent results are applicable to both methods.

Our ensemble model achieved an accuracy of 0.8504, which is very close to that of the single Conformer model. The performance metrics of the ensemble model are also similar to the single model. By aggregating the number of parameters from each model in the ensemble, we reached a total of 1,503,222 parameters. This total is less than the number of parameters of the baseline model. However, since three models require training, both GPU usage and training time are increased. In generating predictions for the test data, the ensemble model required 129 ms/step to create predictions with all three models, which is still faster than the baseline model. Should the models operate predictions in parallel, the time would reduce to 44 ms/step, as this represents the longest prediction time per step among the three models in the ensemble. Overall also the conformer ensemble model presents superior performance to the baseline model. Detailed results of the ensemble Conformer model are presented in Table 7.

Metric	Value
Accuracy	0.850432
Precision	0.895685
Recall	0.826013
F1 Score	0.843624
Number of Model Parameters	1,503,222
Max CPU Usage (Training)	4.331 GB
Max GPU Usage (Training)	12.624 GB
Total Training Time	12,498.357 seconds
Testing Time per Step	129 ms/step

TABLE 7: Model Performance and Resource Usage of Conformer ensemble model

E. Comparison

Regarding the models accuracy, a single Conformer model surpasses other models, while the Conformer ensemble model is only marginally outperformed. The Conformer ensemble model delivers higher precision, and the single Conformer model excels in terms of recall, resulting in nearly identical F1 scores. This equivalence in F1 scores suggests that, from a performance standpoint, both models are equally suitable for the keyword spotting task and clearly outperform our tested CNN and baseline model. A detailed comparison of performance metrics is available in Table 8.

Model	Accuracy	Precision	Recall	F1 Score
Baseline	0.338	0.548	0.267	0.241
CNN	0.508	0.516	0.487	0.486
Conformer	0.853	0.887	0.833	0.849
Conformer Ensemble	0.850	0.896	0.826	0.844

TABLE 8: Model Performance Comparison

Additionally, when considering computational resources and model complexity, a comprehensive overview for this is provided in Table 9. The Conformer ensemble utilizes the fewest parameters and achieves performance metrics comparable to other models.Its suitability for keyword spotting depends on the device's architecture. If the device supports parallel processing, allowing for simultaneous predictions across multiple models, an ensemble model could enhance robustness against noise, variations in speakers, and dialects. However, if the keyword-spotting device cannot process tasks in parallel, an ensemble model may introduce excessive latency in detecting a keyword, as all models must be executed sequentially before arriving at a final prediction. This issue could become more pronounced with ensembles larger than three models. In such cases, a single, larger Conformer model might be preferable. Because of high GPU usage during the training of the conformer model it is best suitable for KWS when it is not trained on the KWS device but rather loaded pre-trained or accessed remotely. If the model is to be trained directly on the KWS device, a CNN model may be a preferable option due to its lower memory requirements. However, achieving usable results would necessitate more refined fine-tuning than what was accomplished within the scope of our project.

VIII. Concluding Remarks

In this project, we compared the recently emerged Conformer architectures with more traditional CNN architectures for keyword spotting, a specialized task in speech recognition. Our approach involved transforming audio into various 2D spectrogram representations, upon which we trained different models. We began with traditional CNN architectures and then progressed to the more recent Conformer architecture. We aimed to enhance our model's performance through Bayesian hyperparameter optimization and by combining different Conformer models into an ensemble. We believe our work makes a contribution to understanding the trade-off between model accuracy, computational complexity, and the training and testing times of the models. This is particularly relevant in the context of keyword spotting, where devices may have memory constraints and the quick detection of keywords is crucial.

Future research could further explore the idea of ensemble learners by employing techniques like bagging and boosting to create different training sets for each model in the ensemble. Experimenting with the number of models in the ensemble and considering the feasibility of running these models in parallel on a keyword spotting device would also be valuable. Additionally, given that ResNets have proven effective in speech recognition in recent years, it would be interesting to evaluate their performance specifically for keyword spotting.

IX. Project Conclusion

A. What we have learned

In our project, we learned the significance of an efficient preprocessing pipeline and data loading to ensure smooth training sessions. We discovered that bottlenecks in these areas could negate the benefits of using GPU resources, leading to slower training times. Additionally, we found that accuracy alone is not a sufficient metric for evaluating multilabel classification tasks. It became clear that hyperparameter optimization is a time-intensive process, emphasizing the importance of establishing a manageable search space to avoid unnecessary delays. Planning which models to explore was crucial due to the vast array of approaches available in speech recognition and KWS literature. Given our limited project time, making informed decisions on which methodologies to test was essential. We also learned to evaluate the complexity of models that performed well in the literature, ensuring they were feasible for KWS within our resource constraints. This process helped us to not only focus our efforts more effectively but also to consider practicality alongside theoretical performance.

B. Difficulties encountered

In our project, we faced both technical and organizational challenges. On the technical front, slow data loading, especially with WAV files, initially hindered our preprocessing efforts. We solved this by pre-loading data into the Colab runtime, which sped up our workflow. We also struggled with defining an effective search space for hyperparameter tuning via Bayesian optimization, due to limited detailed guidance in existing literature.

Organizational challenges revolved around selecting the most appropriate models to implement, given the vast variety available in speech recognition and keyword spotting. Choosing which models to focus on required careful consideration of our project's objectives, resources, and the potential efficacy of each model.

References

[1] "Github repository of the project." https://github.com/theMaren/HDA-KWS-Project/tree/main, 2024.

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

Model	# Parameters	Training Time (s)	CPU Usage	GPU Usage	Prediction Time/Step
Baseline	1,627,933	164.835	11.402 GB	0.372 GB	152 ms/step
CNN	1,659,450	1014.637	3.847 GB	0.563 GB	56 ms/step
Conformer	1,816,090	3359.828	3.984 GB	9.682 GB	44 ms/step
Conformer ensemble	1,503,222	12498.357	4.331 GB	12.624 GB	129 ms/step

TABLE 9: Complexity and Resource Consumption Comparison

[3] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," 2020.

[4] S. Ö. 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," *CoRR*, vol. abs/1703.05390, 2017.

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

[6] Z. X. Ong, "Exploring the short-time fourier transform: Analyzing time-varying audio signals." https://medium.com/@ongzhixuan/exploring-the-short-time-fourier-transform-analyzing-time-varying-audio-signals-98157d1b9a12, 2023. Accessed: 2024-02-14.

[7] L. Roberts, "Understanding the mel-spectrogram." https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53, 2020. Accessed: 2024-02-14.

[8] E. Deruty, "Intuitive understanding of mfccs." https://medium.com/@derutycsl/intuitive-understanding-of-mfccs-836d36a1f779, 2022. Accessed: 2024-02-14.

[9] TensorFlow Team, "Documentation tf.pad." https://www.tensorflow.org/api˙docs/python/tf/pad, 2024. Accessed: 2024-02-14.

[10] J. C. Duarte and S. Colcher, "Building a noisy audio dataset to evaluate machine learning approaches for automatic speech recognition systems," *CoRR*, vol. abs/2110.01425, 2021.

[11] Institute of Phonetics and Speech Processing, "Speech processing cookbook." https://www.phonetik.uni-muenchen.de/forschung/BITS/TP1/Cookbook/node61.html, 2004. Accessed: 2024-02-14.

[12] TensorFlow I/O Team, "Documentation tfio.audio.resample." https://www.tensorflow.org/io/api˙docs/python/tfio/audio/resample, 2024. Accessed: 2024-02-14.

[13] T. Team, "Simple audio recognition: Recognizing keywords." https://www.tensorflow.org/tutorials/audio/simple˙audio, 2024. Accessed: 2024-02-14.

[14] M. Burchi and V. Vielzeuf, "Efficient conformer: Progressive downsampling and grouped attention for automatic speech recognition," in *2021 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 8–15, 2021.

[15] I. Gitman, V. Lavrukhin, A. Laptev, and B. Ginsburg, "Confidence-based ensembles of end-to-end speech recognition models," in *INTERSPEECH 2023*, interspeech˙2023, ISCA, Aug. 2023.

[16] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," 2012.

[17] K. Team, "Keras Tuner Bayesian Optimization." https://keras.io/api/keras˙tuner/tuners/bayesian/, 2024. Accessed: [Insert date here].