Enhancing Spoken Word Detection for Real-World Applications

1st Melissa Hoang
Department of Computer Science
University of Calgary
Calgary, Canada
melissa.hoang@ucalgary.ca

***Abstract*—Spoken word detection is a crucial task in various practical applications, including voice assistants, transcription services, and audio content analysis. This research project aims to develop an efficient and adaptable spoken word detection system to meet the growing demand for natural language interfaces and voice-based services. The system will focus on robustness to noise and speaker variability while leveraging state-of-the-art machine learning and speech processing techniques.**

The proposed approach utilizes a modified version of the M5 architecture, incorporating convolutional layers with batch normalization, max pooling, and fully connected layers. This architecture allows for the automatic learning of discriminative features and temporal patterns from audio data. To enhance the system's performance, data augmentation techniques and transfer learning strategies will be explored, enabling the model to generalize well in real-world scenarios and low-resource settings.

To evaluate the effectiveness of the proposed system, comprehensive experiments will be conducted on diverse tasks and datasets relevant to spoken word detection. The evaluation will emphasize practical applications to assess the system's utility in real-world scenarios.

***Index Terms*—AI, Classification models**

I. Introduction

The field of machine learning has witnessed significant advancements in recent years, revolutionizing various domains and applications. One area that has garnered substantial attention is the detection of spoken words, which plays a crucial role in numerous applications such as speech recognition, natural language processing, and audio analysis. The goal of this research project is to develop a speech command classification system by leveraging state-of-the-art machine learning techniques.

The importance of spoken word detection stems from the increasing demand for voice-controlled interfaces [1], transcription services [2], and audio content analysis [3]. Voice assistants and smart devices have become pervasive in our daily lives, necessitating accurate and efficient systems for understanding and responding to spoken commands and queries [4]. Furthermore, spoken word detection is essential in areas such as call center analytics, voice-based security systems, and audio surveillance, where automatic identification and extraction of meaningful information from speech are critical [5].

One fundamental aspect of spoken word detection is speech feature extraction, which involves transforming raw audio signals into meaningful representations that capture relevant information for classification tasks. Speech feature extraction is a critical step in the pipeline of spoken word detection systems, as it helps to identify discriminative patterns and characteristics that distinguish different words or phonemes. Traditionally, speech feature extraction has relied on handcrafted features such as Mel-frequency cepstral coefficients (MFCCs), linear predictive coding (LPC), and filter banks [7]. These features capture spectral and temporal properties of speech signals and have been widely used in various speech processing tasks. However, these handcrafted features may not fully exploit the rich information contained in the audio data, limiting the performance of spoken word detection systems.

Recent research conducted by Smith et al. [6] provides a comprehensive overview of deep learning approaches for spoken keyword spotting (KWS). The goal of KWS is to detect specific keywords or phrases in continuous speech signals. Their work demonstrates how deep learning techniques have significantly improved KWS performance compared to traditional methods. Building upon the work of Smith et al., this research project aims to explore the deep learning techniques for spoken keyword. The findings and methodologies presented in the paper serve as a valuable reference and inspiration for our project.

II. Model Architecture

The M5 model is a deep learning architecture specifically designed for speech command classification tasks. It is a modified version of the original M5 model introduced in the webpage "Convolutional Neural Networks for Small-footprint Keyword Spotting" by Sainath et al [8].

```
class M5(nn.Module):
    def __init__(self, n_input, n_output, stride
        =16, n_channel=32):
        super().__init__()
        self.conv1 = nn.Conv1d(n_input, n_channel,
            kernel_size=10, stride=stride)
        self.bn1 = nn.BatchNorm1d(n_channel)
        self.pool1 = nn.MaxPool1d(4, stride=2,
            padding=1, ceil_mode=True)
        self.conv2 = nn.Conv1d(n_channel, n_channel
            , kernel_size=3)
        self.bn2 = nn.BatchNorm1d(n_channel)
        self.pool2 = nn.MaxPool1d(4, stride=2,
            padding=1, ceil_mode=True)
        self.conv3 = nn.Conv1d(n_channel, 2 *
            n_channel, kernel_size=1)
```

```
        self.bn3 = nn.BatchNorm1d(2 * n_channel)
        self.pool3 = nn.MaxPool1d(1, stride=1)
        self.conv4 = nn.Conv1d(2 * n_channel, 256,
            kernel_size=1)
        self.bn4 = nn.BatchNorm1d(256)
        self.pool4 = nn.MaxPool1d(4, stride=2,
            padding=1, ceil_mode=True)
        self.fc1 = nn.Linear(256, n_output)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(self.bn1(x))
        x = self.pool1(x)
        x = self.conv2(x)
        x = F.relu(self.bn2(x))
        x = self.pool2(x)
        x = self.conv3(x)
        x = F.relu(self.bn3(x))
        x = self.pool3(x)
        x = self.conv4(x)
        x = F.relu(self.bn4(x))
        x = self.pool4(x)
        x = x.mean(dim=2)
        x = self.fc1(x)
        return F.log_softmax(x, dim=1)
```

Fig. 1. M5 Architecture

Layer	Layer Type	of Filters	Sizes	Dimensions	Dimensions
self.conv1	nn.Conv1d	32	10	13	32
self.bn1	nn.BatchNorm1d	32		32	32
self.pool1	nn.MaxPool1d		4	32	
self.conv2	nn.Conv1d	32	3	32	32
self.bn2	nn.BatchNorm1d	32		32	32
self.pool2	nn.MaxPool1d		4	32	
self.conv3	nn.Conv1d	64	1	32	64
self.bn3	nn.BatchNorm1d	64		64	64
self.pool3	nn.MaxPool1d		1	64	
self.conv4	nn.Conv1d	256	1	64	256
self.bn4	nn.BatchNorm1d	256		256	256
self.pool4	nn.MaxPool1d		4	256	
self.fc1	Linear	25		256	25

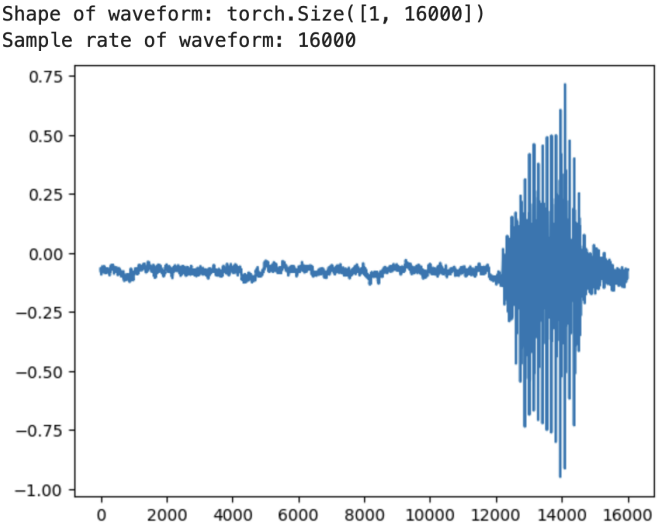
The model begins with a 1D convolutional layer (conv1) that takes an input of n_input channels. This layer applies a convolution operation with n_channel filters and a kernel size of 10. The stride parameter controls the stride value for this layer. The output of this layer undergoes batch normalization (bn1) to normalize the activations and improve the model's training stability. Following conv1, the model employs a max pooling layer (pool1) to reduce the spatial dimensions of the feature maps. It performs 1D max pooling with a kernel size of 4, a stride of 2, and padding of 1. This helps to downsample the input and capture important local features. The model continues with further convolutional layers and max pooling layers. The output from pool4 is flattened and fed into a fully connected layer (fc1). This layer maps the extracted features to the desired output space, with the number of units specified by n_output. A log softmax activation function is applied to the output of fc1 to obtain the final predicted probabilities for each class.

The M5 model architecture is designed to be computationally efficient while maintaining good performance. Additionally, the M5 model utilizes multiple convolutional layers with small kernel sizes, which are effective at capturing local patterns and features in the audio spectrogram.

III. Dataset

The dataset used in this project is the Speech Commands dataset from Google, which is commonly used for speech command classification tasks. It consists of spoken commands from various individuals, recorded in different environments. The dataset includes a diverse set of commands such as "yes," "no," "up," "down," and many others. Each data contains infromation of its waveform, sample_rate, label, speaker_id, and utterance_number. The waveform represents the raw audio signal of spoken commands. It is a one-dimensional time-series data that captures the amplitude variations of the sound wave over time.

Fig. 2. Waveform of Speech Commands dataset



The dataset has been split into three subsets: training, testing, and validation. The sizes of the subsets are as follows:

- Training set size: 84,843 samples
- Test set size: 11,005 samples
- Validation set size: 9,981 samples

A. Data Preprocessing

1) Resample: The waveforms in the dataset may have different sample rates. To ensure consistency, the torchaudio.transforms.Resample function is used to resample the waveforms to a common sample rate of 16,000 Hz. This step ensures that all waveforms have the same frequency resolution.

2) Mel Frequency Cepstral Coefficients (MFCC) transformation: The MFCC transformation converts the audio waveform into a sequence of MFCC feature vectors, capturing important spectral information. MFCCs are widely used features in speech and audio processing task, which are derived from the short-term power spectrum of a signal and capture important characteristics of the underlying audio. We chose MFCC for our KWS task because of its robustness to noise and channel variations as well as effective feature discrimination, making it a great fit for the Speech Commands dataset.

3) Normalization: After computing the MFCCs, the code performs data normalization to make the features more suitable for training the neural network, reducing the impact of feature scale differences across different recordings.

B. Data biases and limitations

The dataset contains limited vocabularies, which can limit its application. It also contains recordings from a relatively small number of speakers. This limited speaker diversity may not adequately capture the variability in speech patterns, accents, and speaking styles present in real-world scenarios. The dataset includes samples with different levels of background noise, but the distribution of noise types and intensities may not fully represent the wide range of noise conditions encountered in practical applications. This can impact the model's robustness to different types of environmental noise.

IV. Training Procedure

The training is conducted useing Google Colab as it provides pre-installed libraries and frameworks commonly used in machine learning and deep learning, such as PyTorch, TensorFlow, and scikit-learn. It also offers access to computational resources such as GPU, enabling efficient model training and experimentation.

The optimizer used is Adam, which helps update the model parameters based on the gradients computed during backpropagation. The learning rate is set to 0.01, which determines the step size for each update during optimization. A StepLR scheduler is used to decrease the learning rate after every 20 epochs. The weight decay is set to 0.0001, which helps prevent overfitting by adding a penalty term to the loss function proportional to the magnitude of the weights.

To further prevent overfitting, early stopping is employed with a patience of 10, meaning that the training terminates when the validation loss does not improve after 10 epochs, and the best model is saved for use.

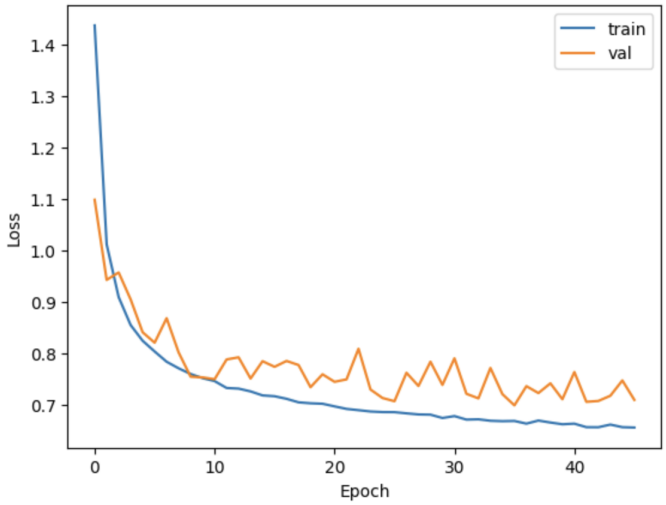
The predefined number of epochs for training is set to 200. However, it was terminated at epoch 46 due to early stopping. In each epoch, the code performs the training phase and the validation phase. In training phase, the loss is computed using the negative log-likelihood loss (F.nll_loss) and gradients are updated (optimizer.step()). In validation state, the model is used on data that it has not been exposed to during training and calculates a validation loss, to identify potential overfitting.

V. Evaluation

The evaluation procedure involves assessing the model's performance on the test dataset. The evaluation metric used is accuracy, which measures the percentage of correctly predicted labels out of the total predictions. The test accuracy of 75% is satisfactory for our light-weight model.

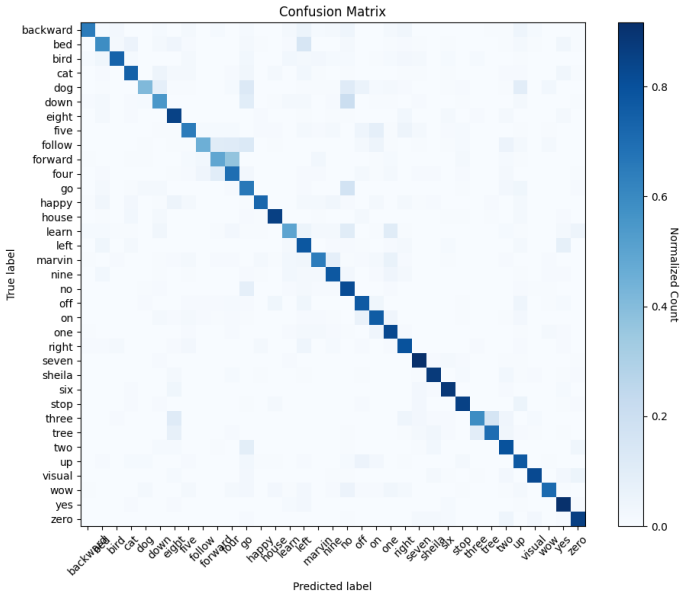
To further investigate the model performance, a confusion matrix is plotted. By examining the matrix, we can gain insights into how well the model is performing for each class and identify any specific misclassifications that may be occurring. Identifying diagonal elements with lower values and

Fig. 3. Train and validation loss during training



off-diagonal elements with higher values indicates the classes that the model struggles to classify. For instance, class "three" can be classified as "tree." The reason could be that they share similar acoustic features. The word "forward" also has a high possibility of being classified as "four." The model may struggle with accurately segmenting and recognizing compound words like "forward" as a single entity. Further model improvements can be made by collecting data and employing augmentation for these classes.

Fig. 4. Confusion matrix



VI. Discussion

The training terminated quite early and validation loss could not be further improved after 46 epochs. This suggests that the model may have reached its capacity and is not able to learn more from the given data. Increasing the model's complexity

can be a potential solution to improve its performance. This can be done by increasing the number of layers or using larger filters. Alternatively, we can also explore more advanced architectures using transfer learning.

The project currently does not include explicit data augmentation techniques. However, it is common practice to apply various data augmentation techniques to improve model generalization and robustness. Some commonly used techniques for speech data augmentation include adding background noise, time stretching, pitch shifting, and speed perturbation. These techniques should be considered for future improvements to help the model handle different environmental conditions, speaker variations, and distortions.

VII. Conclusion

In conclusion, the goal of the project is to develop a speech recognition system using a CNN trained on the Speech Commands dataset. The trained CNN model achieved a satisfactory level of accuracy in classifying speech commands, with a test accuracy of 75%. This demonstrates the effectiveness of the chosen architecture and data preprocessing techniques. The data preprocessing steps, including MFCC extraction and padding, played a crucial role in preparing the input data for the CNN model. These techniques proved effective in capturing relevant acoustic features and enabling the model to learn discriminative patterns. With further improvements, the model can be used in various applications such as voice-controlled interfaces.

References

[1] H. Liu, T. Fang, T. Zhou, Y. Wang, and L. Wang, "Deep learning-based multimodal control interface for human-robot collaboration," Procedia CIRP, vol. 72, pp. 3–8, 2018. doi:10.1016/j.procir.2018.03.224

[2] M. Negrão and P. Domingues, "SpeechToText: An open-source software for automatic detection and transcription of Voice Recordings in Digital Forensics," Forensic Science International: Digital Investigation, vol. 38, p. 301223, 2021. doi:10.1016/j.fsidi.2021.301223

[3] L. Lu and A. Hanjalic, "Audio Keywords Discovery for Text-Like Audio Content Analysis and Retrieval," IEEE Transactions on Multimedia, vol. 10, no. 1, pp. 74–85, Jan. 2008, doi: https://doi.org/10.1109/tmm.2007.911304.

[4] "Jarvis: Artificial Intelligence-based Voice Assistant," international journal of engineering technology and management sciences, vol. 6, no. 6, pp. 50–54, Nov. 2022, doi: https://doi.org/10.46647/ijetms.2022.v06i06.008.

[5] Wahyudi, W. Astuti, and S. Mohamed, "Intelligent Voice-Based Door Access Control System Using Adaptive-Network-based Fuzzy Inference Systems (ANFIS) for Building Security," Journal of Computer Science, vol. 3, no. 5, pp. 274–280, May 2007, doi: https://doi.org/10.3844/jcssp.2007.274.280.

[6] Iván López-Espejo, J. Jensen, John, and J. Jensen, "Deep Spoken Keyword Spotting: An Overview," vol. 10, pp. 4169–4199, Jan. 2022, doi: https://doi.org/10.1109/access.2021.3139508.

[7] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 28, pp. 357–366, 198

[8] "Speech Command Classification with torchaudio — PyTorch Tutorials 1.13.1+cu117 documentation," pytorch.org. https://pytorch.org/tutorials/intermediate/speech$_command_classification_with_torchaudio_tutorial.html (accessed Jun. 09, 2023).