Transformer Model for Keyword Spotting Based on MFCC Features

1st Fengshuo Song (fs2776)
Columbia University
New York, NY
fs2776@columbia.edu

***Abstract*—Keyword spotting is an essential part in many intelligent systems that involves voice interaction with the user. It is important that this step needs to be performed efficiently and accurately. In this paper, I proposed an efficient transformer model based on MFCC features that can spot the keywords from the user utterances. I analyzed two different settings of the MFCC features: high-resolution and low-resolution, and compared their performance. I evaluated my models from two aspects: accuracy and computational complexity. I also compared my results with the state-of-the-art result.**

***Index Terms*—Transformer, MFCC, Keyword Spotting, Self-attention**

I. Introduction

A. Problem Description

There have been many developments in Automatic Speech Recognition (ASR) over the years, and one of the most significant developments was the introduction of neural networks, which allowed for more accurate recognition of speech. It has made ASR systems much better at understanding conversations and the different ways that people talk about the same thing. Recent work in machine learning has further pushed the performance to a new level that is similar to the performance of a human [1].

In particular, keyword spotting (KWS), which is used to detect specific keywords in speech, plays an important part in voice control system. The advancements of ASR technology and other Machine Learning technology have made it reliable, and more and more people are beginning to use it. KWS is commonly used in smart devices or intelligent systems to identify if a particular word or phrase that has been spoken, and the detection of these keywords can lead to subsequent actions. For example, the Apple's AI conversational assistant uses the phrase "Hey Siri" to activate the system and the Google uses " Okay Google" to start a voice search. It plays an essential role in device wake-up and allows hands free interaction with the devices. Therefore, it is important to design efficient algorithms to solve the problem of KWS. In this paper, I introduced a new model that can detect keyword from human speech accurately and efficiently.

B. State of the art in handling this problem or similar problems

The introduction of Deep Neural Network (DNN) makes KWS model outperform the traditional methods such as Hidden Markov Models (HMM) [2] by having higher accuracy and lower detection latency [3]. The problem with DNN-based KWS model is that it ignores the spatial structure of the spectrum in the audio signal and the dependency structure of the context. To solve this problem, the idea of CNN-based models were proposed. The CNN model, which usually consists of repeated blocks of a certain structure and has residual connections within the block, is used to capture features from images, and it can be applied to KWS. Several models such as Convolutional Recurrent Neural Network [4], max-pooling based model with long short-term memory (LSTM) [5], CNN-attention model [6], are proposed. These models are good at finding spatially local correlation and temporal features from the spectrum, and they have achieved better results than the purely DNN-based model. Despite the state-of-the-art results, these model architectures are more complex and require more computational and memory resources. This will be a problem when it needs to run on devices that require low latency. In addition, these models typically have a large number of parameters which makes the training process long and are prone to structure-based overfitting. It is significant to consider the model complexity and remove unnecessary structures to ensure efficiency.

To reduce the model complexity, several methods have been proposed. One of them is the Broadcasted Residual Learning [7], which uses 1D temporal convolution as the residual function while still allowing 2D convolution by a broadcasted-residual connection. Other methods remove the convolutional layers to reduce model size. The Audio Spectrogram Transformer (AST) model [8], which is used for audio classifications, shows that the CNNs are not required for learning the audio signal, and it can achieve better results than CNN-based models. Motivated by the success of AST, it is reasonable to further reduce the complexity by changing the input representation from Mel spectrum to Mel-frequency cepstral coefficients (MFCCs) features. MFCC is a compressible representation of the audio spectrogram that is widely used in Speech Recognition models. It contains low level features and ignores the unnecessary fine details in the spectral structures, which allows it to be trained more efficiently.

C. Dataset

The datasets I used in my project are Google Speech Commands Dataset Version 1 and Version 2 [9]. Version 1 dataset contains roughly 65,000 utterances from 1,881 different speakers and Version 2 dataset contains around 105,000 utterances from 2,618 different speakers. Each audio sample is one second long with sampling frequency of 16kHz, and each sample is an utterance of keywords. The keywords chosen are common words in English dictionary including digits from 0 to 9 and other useful commands that are used in intelligent systems. Since the KWS model should be speaker-independent, which means it can recognize each speaker equally well, it is important to have audio samples from different speakers. This allows the model to generalize the utterances from the speakers rather than focusing on a specific speaker.

The dataset contains the validation list and test list [9], which I used to split the dataset into training, validation and testing set. I trained my model on the training data, perform hyper-parameter tuning on the validation, and calculate test accuracy on the testing data.

II. Formulation from the State of the Art

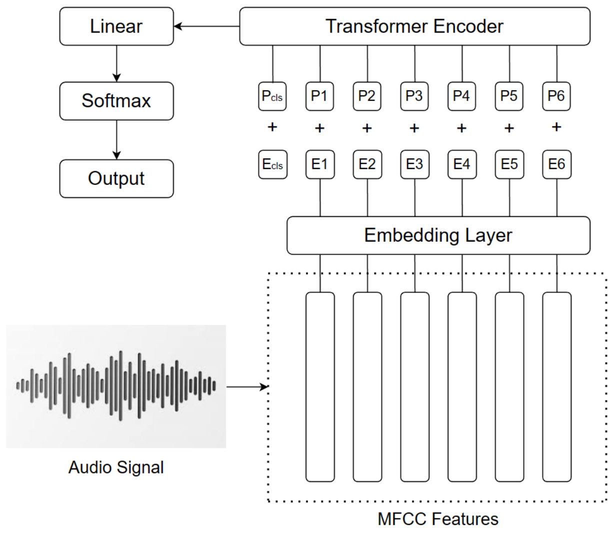

Fig. 1. The MFCC Transformer architecture.

The problem with state-of-the-art methods is the lack of efficiency. These methods typically have complex model structure to learn the features from the spectrogram, which makes it costly to train. To solve this problem, I introduce the MFCC Transformer Model, which is similar to AST, a convolution-free, purely attention-based model [8]. Figure 1 shows the architecture of the entire model. First, for each the input audio signal sample, the model calculates the MFCC. For the MFCC matrix, the number of rows is the number of cepstrums, and the number of columns is the total number of frames. I used Kaldi to compute MFCC, and the default cepstrum number is 13. I also explored other values and see in which setting the model achieves the best result.

Then, I separate each frame and obtain a sequence of column vectors. Each column vector contains the low-level features for that particular frame, and different vectors have temporal correlations. Each vector is then passed to an embedding layer, which performs a projection to a denser representation. Since the Transformer model does not have the information about the input order, I add a positional embedding to each of the embedded vector. In addition, I add a [CLS] token at the start of each sequence, which is very similar to Bidirectional Encoder Representations from Transformers (BERT) [10] models. The purpose of this additional token is to represent the classification of the audio sequence.

The resulting sequence is passed to a transformer encoder, which is composed of a series of layers, each consisting of a multi-head self-attention layer and fully connected feed-forward layer. Since the keyword classification task does not involve generating sequence, the decoder layer of transformer is not needed. There are a number of successful pre-trained encoder models that have achieve state-of-the-art results, such as BERT and A Lite BERT (ALBERT) [11], they focus on the language models and have no understanding of the structure of the audio sequence. Hence, I use the original Transformer encoder described in [12]. The output of the transformer is further passed to a linear layer and a SoftMax activation function to get the probability for each class label.

III. Approach and Formulation

A. Data Preprocessing

The datasets were downloaded from tensorflow. As mentioned above, I built and tested my model on two datasets, and I split training, validation and testing set according to the validation list and testing list. This is the most commonly way train test split, and I could have a better comparison with the state-of-the-art model. For the v1 dataset, there are 51088 training samples, 6798 validation samples and 6835 test samples; for the v2 dataset, there are 84843 training samples, 9981 validation samples and 11005 test samples.

B. Feature Extraction

The features that are used for building the model are the MFCCs of the utterance, and they are calculated using Kaldi. The MFCC features are calculated per 25ms frames with shifting 10ms each time. I used 2 different configurations for calculating the MFCCs: one is the default setting of 12 cepstrums; the other is high resolution setting with 40 cepstrums. There is no dimension reduction in the 40 cepstrum setting.

The audio samples for training, validation and test set are split into chunks and each chunk has an ark file corresponding to the calculated MFCC features. These files would be used to build the dataset for training and testing the transformer model.

C. Transformer Model

The feature dimension of the input is cepstrum number * sequence length. Since the sequence length of each sample varies, I used zero padding for the samples that have smaller sequence length. The zero padding adds zero vectors to the end of the sample to make sure that each sample has the same sequence length. In addition, in order for classification, I added a vector with all entries being 1 at the beginning of the sequence for each sequence.

Here is the parameter setting of the transformer model with low resolution MFCC settings:

```
– nembed = 32 (low res MFCC); 64 (high res MFCC)
– nhead = 8
– nhid = 256
– nlayers = 4
– dropout = 0
– dropout_positional_encoding = 0
```

The model summary with low resolution MFCC settings for predicting 12 class keywords is shown below:

```
TransformerModel(
  (pos_encoder): PositionalEncoding(
    (dropout): Dropout(p=0, inplace=False)
  )
  (transformer_encoder): TransformerEncoder(
    (layers): ModuleList(
      (0): TransformerEncoderLayer
      (1): TransformerEncoderLayer
      (2): TransformerEncoderLayer
      (3): TransformerEncoderLayer
    )
  )
  (encoder): Linear(in_features=13, out_features=32,
   bias=True)
  (decoder):Linear(in_features=32, out_features=12,
   bias=True)
)
```

The embedding dimension is 32. Unlike word embedding layers where the input is a sequence of numbers representing tokens, the embedding layer in my model is a linear layer that projects the input vectors to a higher dimension.

For the encoder layer, the number of heads in the MultiHeadAttention model is 8; the dimension of the feed forward model is 256. The dropout does not seem to improve the performance of the model, so I set the dropout in both positional encoding and feed forward layer to 0. There are 4 encoder layers in the transformer encoder model.

Here is the structure of each encoder layer:

```
TransformerEncoderLayer(
  (self_attn): MultiheadAttention(
    (out_proj): NonDynamicallyQuantizableLinear(
    in_features=32, out_features=32, bias=True)
  )
  (linear1): Linear(in_features=32, out_features=256,
  bias=True)
  (dropout): Dropout(p=0, inplace=False)
  (linear2): Linear(in_features=256, out_features=32,
  bias=True)
  (norm1): LayerNorm((32,), elementwise_affine=True)
  (norm2): LayerNorm((32,), elementwise_affine=True)
  (dropout1): Dropout(p=0, inplace=False)
  (dropout2): Dropout(p=0, inplace=False)
)
```

The last layer is a linear layer with a softmax function that takes the input of the first position in the sequence after the transformer model, which is the output of the [CLS] token.

IV. Experiments and results

I have performed three experiments to evaluate the performance of the Keyword Spotting model. The first experiment is doing classification on different number of classes. In the second experiment, I evaluated the classification model using the test data from the other dataset (i.e., building the model using training set from dataset v1 and evaluate the model using test set from dataset v2). In addition, I compared the result with the convolutional model in [10]. In the third experiment, I will analyze the model complexity and running time.

A. Keyword Spotting with Different Number of Classes

In this experiment, I used the transformer model described above to train the data. There are 30 classes of keywords in the dataset v1 and 35 classes in the dataset v2. I have created 3 tasks with different number of classes to see if the model performance degrades greatly as the number of classes increases.

The first task contains 12 classes (10 classes from the original dataset, 1 class for other keywords and 1 class for the noise). The 10 classes chosen for classification are “Yes”, “No”, “Up”, “Down”, “Left”, “Right”, “On”, “Off”, “Stop” and “Go”. They are standard set of the Keyword Spotting problem, and I used the result to compare with other models. The noise class are sampled from the background noise files. There are 6 files in the noise folder, and each audio is around 1 minute long. The data are sampled randomly from the files with 1 second long, and added to the training, validation and testing data. The second task contains 21 classes (top 20 occurrence classes from the original dataset and 1 class for other keywords). The third task is the all-class keyword classification.

All three tasks were be performed using the dataset v1 and v2, and the features are low resolution MFCCs or high resolution MFCCs. There are 12 models in total. All models are trained with 100 epochs with early stopping patience of 15. The batch size is 32.

TABLE I
TEST ACCURACIES

	12-class	21-class	all-class
v1 (low-res)	92.84%	**89.77%**	89.22%
v2 (low-res)	92.16%	87.93%	87.43%
v1 (high-res)	92.84%	89.22%	**89.52%**
v2 (high-res)	**92.95%**	87.86%	87.60%

Table I shows the prediction accuracies on the test set. The accuracies of 12-class classification are higher than the other two tasks, and the model using dataset v2 is slightly better. In 21-class and all-class classification, the model using dataset v1 is better and there is not much difference in accuracy for the two tasks. The results are as expected. As the number of classes to predict increases, the accuracy lowers. The model does not degrade too much for increasing number of classes.

The high resolution MFCC does not seem to show any improvements to the classification. The low resolution MFCC features are very good for the classification, and the extra dimensions in the high resolution MFCC seems unnecessary. Figures 2, 3 and 4 show the accuracies during the training process using low resolution MFCCs. Training on dataset v2 takes more steps, since it has more data and more classes in the all-class classification problem. When the training process terminates, the training accuracies are slightly higher than validation accuracies. There is no obvious under-fitting or over-fitting.

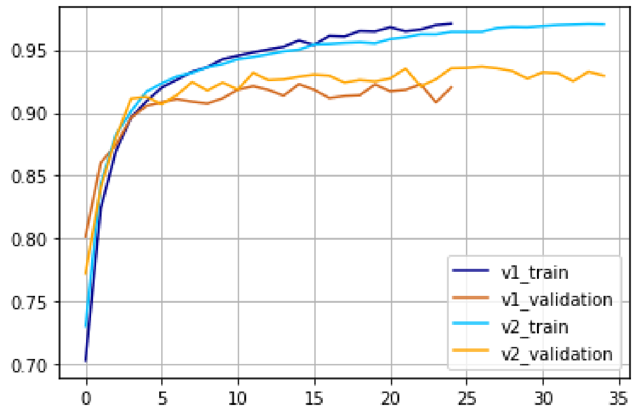

Fig. 2. Training and validation accuracies during each epoch of training in the 12-class classification problem.

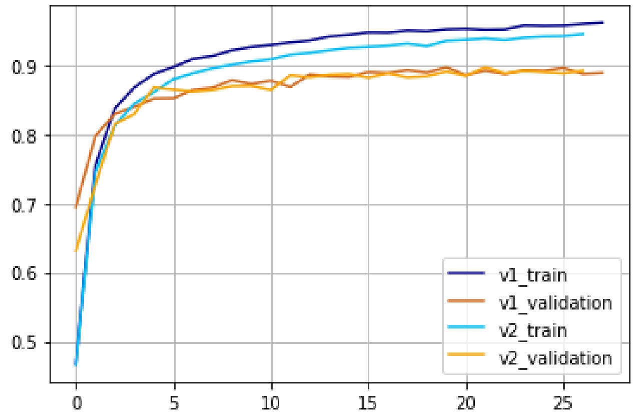

Fig. 3. Training and validation accuracies during each epoch of training in the 21-class classification problem.

Compared to the state-of-the-art-result in Table II, my transformer model does not perform very well. One of the potential reasons is that the state-of-the-art model usually uses pretrained models to discover high-level features inside the spectrogram. On the other hand, my model uses MFCC features, which may ignore some information about the utterance that is helpful to prediction. There are also not many helpful

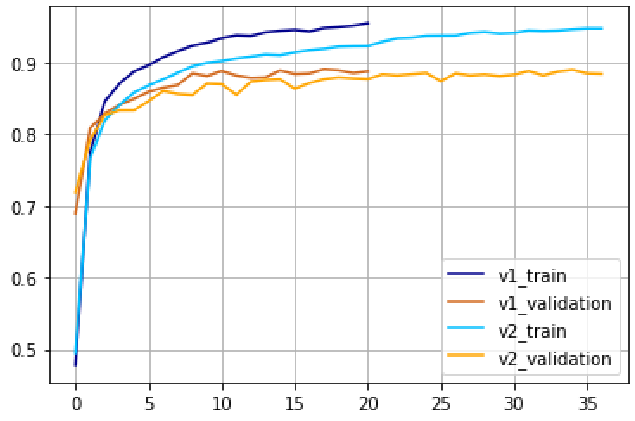

Fig. 4. Training and validation accuracies during each epoch of training in the all-class classification problem.

pre-trained frameworks about MFCC that can help find high-level features efficiently.

TABLE II
ACCURACIES IN THE STATE-OF-THE-ART MODELS

	v1	v2
CNN [13]	84.6%	-
DNN [13]	91.6%	-
Basic LSTM [13]	92.0%	-
Transformer (low-res MFCC)	92.8%	92.1%
Transformer (high-res MFCC)	92.8%	93.0%
DS-CNN [13]	94.4%	-
Attention RNN [14]	95.6%	96.9%
ResNet-15 [13]	95.8%	-
DS-ResNet14 [16]	95.9%	-
DS-ResNet18 [16]	96.7%	-
MHAtt-RNN [17]	97.2%	98.0%
TripletLoss-res15 [18]	**98.5%**	**98.3%**

B. Keyword Spotting with Training and Testing on Different Dataset

In this experiment, I performed simple 8-class classification and compared the results with the baseline in result [9]. The baseline methods was based on Convolutional Neural Networks for Small-footprint Keyword Spotting [5]. I computed 4 test accuracies for both low-res MFCC features and high-res MFCC features:

- training on dataset v1 and test on dataset v1;
- b. training on dataset v1 and test on dataset v2;
- c. training on dataset v2 and test on dataset v1;
- d. training on dataset v2 and test on dataset v2.

The transformer model based on MFCC features improves a lot over the baseline model. The model achieves the greatest accuracies when training on dataset v2 and test on dataset v1, since dataset v2 has more data.

Figures 5 and 6 show the confusion matrix of the best model using low resolution MFCCs and test on dataset v1 and dataset v2.

TABLE III
TEST ACCURACIES ON 8-CLASS CLASSIFICATION

	v1 test	v2 test
v1 traning (baseline)	85.4%	82.7%
v2 training (baseline)	89.7%	88.2%
v1 traning (low-res)	90.8%	90.5%
v2 training (low-res)	92.1%	92.6%
v1 training (high-res)	92.2%	91.7%
v2 training (high-res)	**93.2%**	**93.5%**

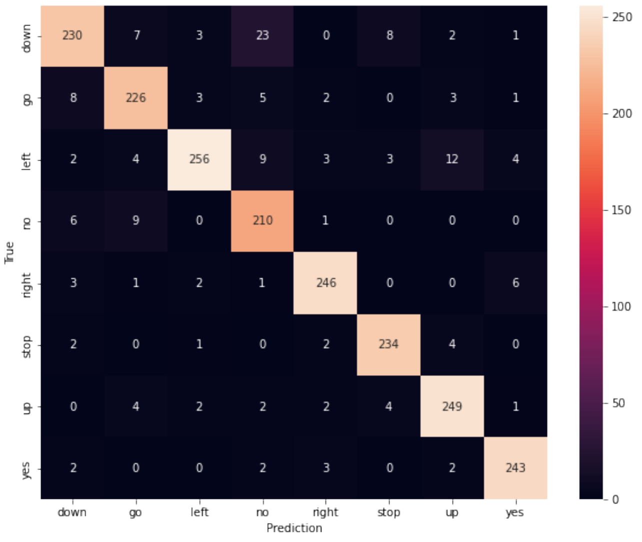

Fig. 5. Confusion matrix on the test set from dataset v1.

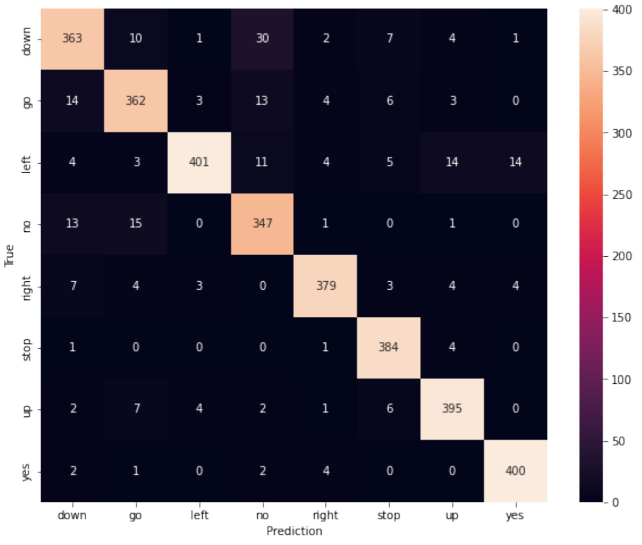

Fig. 6. Confusion matrix on the test set from dataset v2.

The high resolution MFCC performs better in this task. Figure 5 and 6 show the confusion matrix of the best model in each test set. The model achieves very high accuracy on each command. Commands “down”, “go” and “no” seem to be a little more difficult to predict, and the most frequent error is misclassifying command “down” to command “no”.

C. *Model Complexity Analysis*

In this section, I analyzed the model complexity using the number of trainable parameters and multiply–accumulate operations (MAC). The number of parameters can affect the time for model training and the MAC operations will affect the time for training and making predictions. For the sake of simplicity, I only calculated the operations for the linear layers, multi-head attention layers as well as their activation functions. Although there are other operations that take time, their calculations make very small proportion, and can be ignored.

Model parameters The number of trainable parameters is one of the most important factors of model training time. During each iterations, these parameter values get updated through backpropagation. Typically, the more parameters a model has, the longer it takes to train. Adding dropout layers can reduce the number of parameters by randomly setting the weights connecting the nodes to be 0. For simplicity, I do not take the effect of dropout layers into account when calculating the number of model parameters.

Since the my transformer model does not have any pre-trained layers, trainable parameters are all the parameters in the model. As mentioned in 3.3, the model contains a dense layer, 1 transformer encoder and another dense layer. Inside the transformer encoder, there are 4 encoder layers, each with a self-attention and 2 dense layers. Tables IV and V show the parameters of each component.

TABLE IV
PARAMETERS OF EACH LAYER IN THE MODEL

	Parameters (Percentage)	
	low resolution	high resolution
Embedding	448 (0.527%)	2.62K (1.290%)
Encoder Layer 1	20.9K (24.601%)	49.7K (24.456%)
Encoder Layer 2	20.9K (24.601%)	49.7K (24.456%)
Encoder Layer 3	20.9K (24.601%)	49.7K (24.456%)
Encoder Layer 4	20.9K (24.601%)	49.7K (24.456%)
Linear	396 (0.466%)	780 (0.384%)

TABLE V
PARAMETERS OF EACH LAYER IN THE ENCODER LAYER

	Parameters (Percentage)	
	low resolution	high resolution
Self-attention	4.22K (4.973%)	16.64K (8.183%)
Feedforward 1	8.45K (9.946%)	16.64K (8.183%)
Feedforward 2	8.22K (9.682%)	16.45K (8.809%)

MAC operations I used the number of MAC operations to measure the computational cost of the model. An MAC operation is a step of accumulating the product of two numbers, which is very similar to the operations that are performed in linear layers. In addition, it can also be used for the approximation of division and square root [15], which means it can perform self-attention calculations.

Tables VI and VII show the MAC operations of each component in the transformer model. The transformer layers make up the most proportions of calculations; and inside the transformer, the two dense layers account for the most calculations. .

TABLE VI
NUMBER OF MAC OPERATIONS OF EACH LAYER IN THE MODEL

	MAC Operations (Percentage)	
	low resolution	high resolution
Embedding	40.8K (0.374%)	250.94K (1.009%)
Encoder Layer 1	2.71M (24.906%)	6.15M (24.747%)
Encoder Layer 2	2.71M (24.906%)	6.15M (24.747%)
Encoder Layer 3	2.71M (24.906%)	6.15M (24.747%)
Encoder Layer 4	2.71M (24.906%)	6.15M (24.747%)
Linear	396 (0.004%)	780 (0.003%)

TABLE VII
NUMBER OF MAC OPERATIONS OF EACH LAYER IN THE ENCODER LAYER

	MAC Operations (Percentage)	
	low resolution	high resolution
Self-attention	1.11M ((10.171%)	2.94M (11.834%)
Feedforward 1	803.07K (7.368%)	1.61M (6.457%)
Feedforward 2	802.85K (7.366%)	1.61M (6.457%)

Comparisons I compared the model complexity with state-of-the-art models. Table VIII is the result.

TABLE VIII
MODEL COMPLEXITY OF THE STATE-OF-THE-ART MODELS

	Parameters	MAC Operations
Att-RNN	202K	22.3M
ResNet-15	238K	894M
MHAtt-RNN	743K	22.7M
DS-ResNet14	**15.2K**	15.7M
DS-ResNet18	72K	285M
Transformer (low-res MFCC)	84.94K	**10.90M**
Transformer (high-res MFCC)	203.34K	24.87M

The low resolution MFCC transformer performs very good in terms of complexity. Although the DS-ResNet14 and DS-ResNet18 have fewer model parameters, they have more MAC operations. On the other hand, the high resolution MFCC transformer is much more complex. It has higher input dimension and the embedding size also needs to be increased so that it can learn the data.

V. CONCLUSION AND PROPOSED FUTURE WORK

A. Conclusion

In this paper, I introduced the transformer model based on MFCC features for Keyword Spotting. I analyzed the model accuracy on predicting different number of classes, as well as training and testing on different datasets. In addition, I evaluated the model complexity using the number of parameters and MAC operations. The model produces high accuracies on these prediction tasks, although it does not reach the state-of-the-art level result. When it comes to complexity, the low resolution MFCC transformer performs very good. This model sacrifices some accuracy in order to gain more efficiency.

B. Future Work

The model still needs improvement in accuracy. One of the proposed works is to use pretrained models to discover more features useful to the prediction on MFCC (i.e., learning the i-vectors and adding to the MFCC). This model does not have used any data augmentation techniques. The proposed work is to augment the data by adding noise, changing pitch or speaking rate, shifting time, etc. They are useful to improving the model accuracy.

In addition, there is one limitation of this model: it can only recognize commands in a certain set. The proposed work is to make the model suitable for open set prediction, which is more useful and practical in the real world.

REFERENCES

[1] T. Bluche, M. Primet, T. Gisselbrecht, "Small-Footprint Open-Vocabulary Keyword Spotting with Quantized LSTM Networks" arXiv preprint arXiv: 2002.10851, 2020.

[2] B. Yan, R. Guo, X. Zhu and B. Zhang, "An approach of keyword spotting based on HMM," Proceedings of the 3rd World Congress on Intelligent Control and Automation (Cat. No.00EX393), 2000, pp. 2757-2759 vol.4, doi: 10.1109/WCICA.2000.862561.

[3] G. Chen, C. Parada and G. Heigold, "Small-footprint keyword spotting using deep neural networks," 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 4087-4091, doi: 10.1109/ICASSP.2014.6854370.

[4] S. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, A. Coates, "Convolutional Recurrent Neural Networks for Small-Footprint Keyword Spotting" arXiv preprint arXiv: 1703.05390, 2017.

[5] M. Sun, A. Raju, G. Tucker, S. Panchapagesan, G. Fu, A. Mandal, S. Matsoukas, N. Strom, S. Vitaladevuni, "Max-Pooling Loss Training of Long Short-Term Memory Networks for Small-Footprint Keyword Spotting" arXiv preprint arXiv: 1705.02411, 2017.

[6] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," in Interspeech, 2020.

[7] B. Kim, S. Chang, J. Lee, D. Sung, "Broadcasted Residual Learning for Efficient Keyword Spotting" arXiv preprint arXiv: 2106.04140, 2021.

[8] Y. Gong, Y. Chung, J. Glass, "AST: Audio Spectrogram Transformer" arXiv preprint arXiv: 2104.01778, 2021.

[9] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition" arXiv preprint arXiv: 1804.03209, 2018.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pretraining of deep bidirectional transformers for language understanding," in NAACL-HLT, 2019.

[11] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, R. Soricut, "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations" arXiv preprint arXiv: 1909.11942, 2020.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in NIPS, 2017.

[13] Zhang, Yundong, et al. Hello Edge: Keyword Spotting on Microcontrollers. arXiv preprint arXiv: 1711.07128, 2018.

[14] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," CoRR, vol. abs/1808.08929, 2018.

[15] R. Tang and J. Lin, "Deep residual learning for small-footprint keyword spotting," in ICASSP. IEEE, 2018, pp. 5484–5488.

[16] M. Xu and X. Zhang, "Depthwise separable convolutional resnet with squeeze-and-excitation blocks for small-footprint keyword spotting," in INTERSPEECH. ISCA, 2020, pp. 2547–2551.

[17] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," in INTERSPEECH. ISCA, 2020, pp. 2277–2281.

[18] Vygon, Roman, and Nikolay Mikhaylovskiy. Learning Efficient Representations for Keyword Spotting with Triplet Loss. 2021, pp. 773–85. arXiv.org.