Voice recognition, combining Neural Network models

Student project in Machine Learning (TMLS20)

Kristoffer Pettersson *Student at Department of Computer Science*
Jönköping University
Jönköping, Sweden
pekr19ct@student.ju.se

***Abstract*—The interest in voice recognition applications has grown over the last years as the technologies surrounding this becomes more frequently used by the community at large. The task at hand is a multiclass classification problem where a voice recognition algorithm will be created. The raw audio wave, and a feature extracted signal of the same input, is paired together. Here two different models are first trained to find as high performance as possible. When this is found, a third model will be created by concatenating the two preceding models. The idea is to show that the third model is stronger in performance than the both preceding two. Thus, leading to the suggestion that the combined model can pick up different features from the two models with high accuracy.**

It is shown that the assumptions are correct, the combined model is performing better than both the other to, both with regard to accuracy and loss. This is also shown with a McNemar hypothesis test.

An interesting idea for the future is to study more in detail how human sound is created and perceived. This to gain more knowledge and choose other approaches in creating well performing models.

***Index Terms*—machine learning, Voice recognition, MFCC, CNN**

I. Introduction

Over previous years, many new technical platforms have gained interest in the field of voice recognition. The applications can be many: voice commands, recognition of people in a professional meeting, transcribing from a professional meeting, to mention a few. When the technology becomes more frequently used by the community at large, the quality and performance of machine learning models that can recognize commands or people becomes more important and interesting.

In this report, I will show a path of creating a voice recognition algorithm. The task is of multiclass classification nature. I will find values and techniques that are suitable and hopefully leads to a well performing model. Inspiration is taken from a Kaggle-competition from 2018 [1] and the dataset used is described below in this report. The data contains 33 classes of different spoken words that are commonly used as voice-commands (e.g., ”up”, ”down”, ”yes”, ”no” etc). The task is to train a model that can predict which word has been spoken out of the 33, given a 1 second audio recording.

Convolutional Neural Networks (CNNs) are commonly used when creating classification algorithms. It is often heard of when image recognition tasks are performed but is also used in voice recognition tasks. [2], [3], [4]

Two models will be trained separately, one on the raw audio wave and one where the data has been passed through a MFCC feature extractor prior to model training and hence has different shape. The models will be produced from the ground up, similar to the way that is done in the Kaggle competition, with as high performance as possible on the dataset given. This part of the training will be referred to as the ”Base models”. After this, the models will be put together and see if they can strengthen each other and perform better than their individual performance. The tools that will be used are tensorflow and librosa library, together with the standard python libraries.

A. Algorithmic choices

Audio recordings can be visualized and presented to a Neural Network in different ways. Either in the format of raw audio waves, or in the format of output from some feature extraction algorithm. Two alternatives together with the raw audio waves has been considered when building the models, spectrogram and Mel-frequency cepstral coefficients (MFCCs).

As mentioned by Muda [2] and Chowdhury [4], cepstral feature extractors are widely used and have reputation of performing well on voice recognition tasks. Anvarjon [5] mentions that spectrograms are frequently used in Speech Emotion Recognition systems. MFCCs are known to be sensitive to noisy environments [4], and the dataset in this project is not perfect. Even so, the quality of the recordings in the dataset are generally good.

Many implementations exist where voice recognition tasks are performed using a CNN. Often in combination with spectral or cepstral feature extraction prior to feeding the network. [6], [2], [4].

For this project I have chosen to use MFCC as feature extraction algorithm in the preprocessing step and to use CNN in the classification task at hand.

II. Dataset and preprocessing

A. Dataset used

The Kaggle competition that is the inspiration of this project uses a public dataset from Tensorflow [7]. I will use the

same dataset, but in its second version. The main difference compared to the original dataset is its size. Instead of 65000 words spoken, it is now 105000 words spoken. The dataset is crowd-sourced which means it is recorded by private persons around the world. Instructions were given of how to perform the recordings, but quality of the recordings can vary between the samples. The words that are spoken are command-like, words and shall have a length of 1 second each and a sampling rate of 22050Hz. When examining the data, not all samples keep the same length though. To be able to handle data in a CNN, all the data samples must have the same shape when the model is trained and used. Therefore, this has to be dealt with in the preprocessing step.

B. Preprocessing

Preprocessing is the process of transforming and forming the data so that a model can be trained and used. In this process there are both steps that are necessary to make the model accept the data at all. But also the process of applying own choices of how to present the data to the CNN model. [3] The chosen dataset is said to be samples of 1 sec long spoken words. When starting to work with the data, it is seen that it is of variable length. 90% of the data in the dataset were exactly 1 second in length. To deal with the last 10%, zero-padding is performed.

Two separate models will be created. The first model, that contains the raw audio wave have been downsampled from 22050Hz to 8000Hz. Even though the reason for this transformation depends on the time spent in model training, 8000Hz is considered enough to find patterns and classify human speech [8]. For the second model, all data are transformed with an MFCC algorithm (python librosa library).

Normalization of a dataset is important when training Neural Networks. This is done for both branches of this dataset. Normalization is done with mean 0 and standard deviation 1 [9].

All data are randomly split into different sets (80% train, 10% validation, 10% test) before they are transformed into tensor arrays. This random split is kept throughout the experiments.

III. Theoretical background

A. Acoustic waves, spectrograms and MFCCs

Acoustic waves are described as frequencies and related energies. When we talk, the energy levels in the surrounding air is changing, and also perceived by the people around us. Acoustic waves are continuous, and when digitally stored it is discretised. [10]

Acoustic waves and audio signals is a field in itself and here only touched briefly. The focus is to take a stored digital signal in raw format (wav) and find a machine learning solution that can predict what word has been spoken.

Sampling rate is the number of digital points stored per second. This means that a recording with sampling rate 22050 Hz has 22050 datapoints per second recorded. A higher sampling rate thus gives a higher precision in the recorded audio file.

A raw digital signal can be processed directly by a Neural Network, however the possibility of preprocessing this signal with feature extraction algorithms can help the Neural Network in finding other feature-patterns in the recording. Moreover, a raw audio signal can reveal some features and other features becomes clearer when processed.

Two types of feature extractions that is often mentioned in the machine learning community are Spectrograms and Mel-frequency cepstrums (MFCCs).

A spectrogram is produced by taking the raw audio signal and splitting it in timeframes. For each of these, a distribution of energy in different frequencies is calculated through Fourier Transforms. The visualization of this is timeframes as x-axis and frequency as y-axis. The intensity or amplitude are visualized through the colour.

One of the most used feature extractor algorithms, in the context of voice recognition, are Mel Frequency Cepstral Coeffients (MFCC) which takes the signal and divides it into small timeframes, typically 20-40 ms in length. Each frame is converted from the domain of time into the domain of frequency in the same manner as a spectrogram. Further, the timeframe is passed through several operations to end up with a number of features extracted from the timeframe. The idea is to produce a representation similar to how humans perceive audio signals, considering the possibility of human perception in different audio spectrums. When using python librosa library, its default output is 20 MFCC features per timeframe.

[2], [4], [6], [11]

What is taken as input to this project is the possibility and opportunity of using these different feature extractions for voice recognition in machine learning. The idea in combining two neural network models with different ”view” of the same data, is that they could strengthen each other, where one model does not find a pattern, the other might.

Fig. 1 shows an audio signal input with a graphical representation of its raw signal, spectrogram and MFCC respectively.

B. Convolutional Neural Networks

Convolutional Neural Networks are neural networks that uses the mathematical operation of convolution instead of general matrix multiplication used in dense, fully connected layers. The input space is traversed by a kernel, seen as a ”window” of random numbers. For each position that the kernel passes, convolves, a convolutional operation is calculated with the kernel. When the whole input space is convolved, the output feature map is returned.

Convolutional layers inside a CNN model are most often combined with a pooling layer. Pooling also traverses the input space when generating its output feature map. From each window position a value is calculated, most often the max value of the window. In such case, the pooling layer is a ”Max pooling layer”. Max pooling simply takes the highest value in the current ”window” and stores this for its output feature map. This also means that the size of the output feature map is reduced compared to the size of the input feature map, depending on the size of the pooling window. [3]

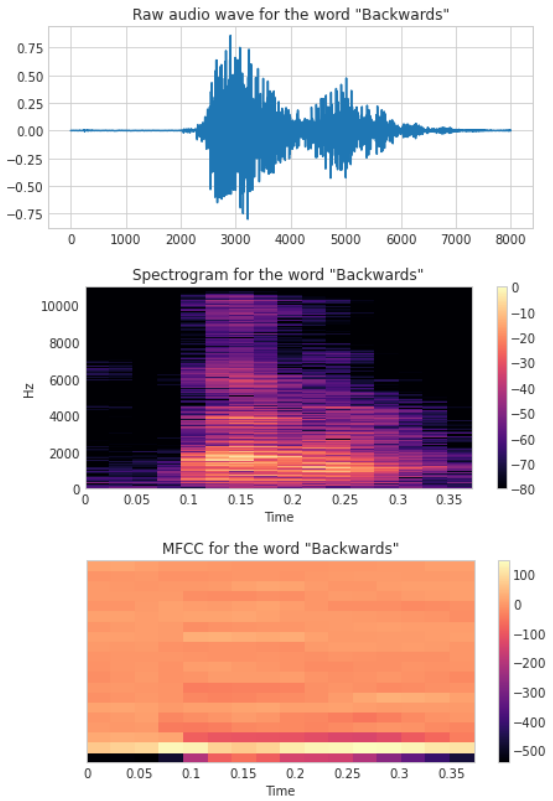

Fig. 1. Audio recording in different shapes

As mentioned in [3], CNNs have two interesting properties:

- The patterns learnt are translation invariant: when a pattern is learnt at one position of the input space, this becomes available to recognize anywhere in further convolutions.
- Convolutional networks can learn hierarchies of patterns: this can be exemplified by an image of a cat. Starting from the low level view, the network recognize shapes, e.g. part of a ear of the cat. In following layers, other parts of the ear is recognized and seen together with the previous known feature. In the end of the convolutional network, the pieces put together mean that the network recognizes that there is a cat in the image, instead of just an unorganized set of shapes.

The above mentioned properties are the reason why CNNs are popular, they facilitate a way of finding patterns, objects in the input space, regardless of where they show up. [12], [3]

IV. Results

Base training is here referred to the process of training two individual models on the same input, one in its raw format and one on the extracted MFCC-features. The training is performed in two steps. First a smaller part of the dataset is used in training and trying to find optimal hyperparameter settings. This is to save time in the elaborative work. A starting set of hyperparameter settings are based on a ”gut-feeling”, and then tried out and tested with several variations and combinations. The first layers in a CNN learns the finest shapes. E.g., edges, corners etc. When an optimal set of hyperparameter settings is found for the smaller dataset, the same settings are used for training and further tuning of the full dataset. The results are shown only for the full dataset training.

In the elaboration of finding the best possible model, the following hyperparameters are tested and evaluated: number of convolutional blocks, number of layers per convolutional block, number of neurons per layer, kernel size, the use of padding in convolutional layers, dropout, and number of epochs.

A. Base training raw audio wave

The raw audio waves are a 1-dimensional input. To classify the input, blocks of 1D convolutional layers in combination with 1D Max pooling layers are used. After the convolutional blocks, the input is flattened before sent to dense layers and finally a dense, output layer with 33 units together with softmax activation function.

Table I shows the layers of the model created. The model is trained with a size of 128 samples per batch and for 15 epochs. The resulting performance is shown in fig. 2 and in table III.

TABLE I
RAW MODEL, LAYERS

Layer	Filters	Kernel/window	Act. function
Conv1D	64	9	relu
Conv1D	64	9	relu
MaxPooling1D		7	
Conv1D	128	9	relu
Conv1D	128	9	relu
MaxPooling1D		7	
Conv1D	256	9	relu
Conv1D	256	9	relu
MaxPooling1D		5	
Flatten			
Dropout		(0.4)	
Dense	128		relu
Dropout		(0.4)	
Dense	33		softmax

Total number of parameters: 2 197 217.

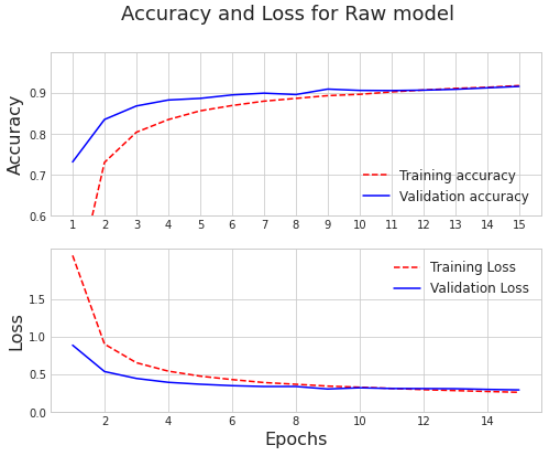

Fig. 2. Raw model, accuracy and loss

B. Base training MFCC feature extracted

The MFCC feature extractor returns a 2-dimensional input, much like an image. Blocks of 2D convolutional layers in combination with 2D Max pooling layers are used. After this, the input is flattened and sent to dense layers in combination with dropout, ending with a final dense layer with 33 units and softmax activation function.

Table II shows the layers of the model created. The model is trained with a size of 32 samples per batch and for 15 epochs. The resulting performance is shown in fig. 3 and in table III.

TABLE II
MFCC MODEL, LAYERS

Layer	Filters	Kernel/window	Act. function
Conv2D	64	(3,3)	relu
Conv2D	64	(3,3)	relu
Conv2D	64	(3,3)	relu
MaxPooling2D		(2,2)	
Conv2D	128	(3,3)	relu
Conv2D	128	(3,3)	relu
MaxPooling2D		(2,2)	
Flatten			
Dropout		(0.25)	
Dense	128		relu
Dropout		(0.25)	
Dense	128		relu
Dropout		(0.2)	
Dense	33		softmax

Total number of parameters: 431 521.

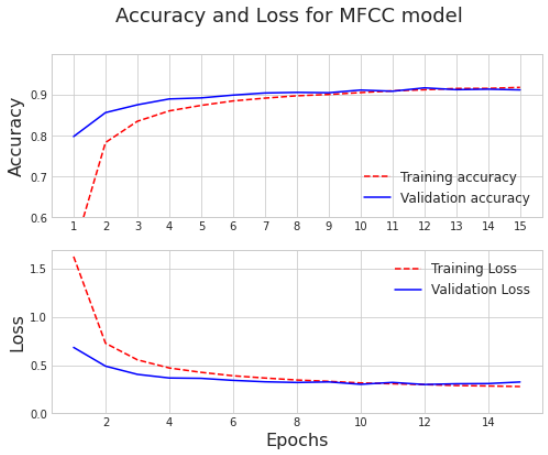

Fig. 3. MFCC model, accuracy and loss

C. The combined model

As is shown in table III, the results from model training with raw and MFCC model separately gave similar accuracy on the held out test set, 91.13% and 90.41% with a corresponding loss at 0.32 and 0.34.

The combined model is built and trained with the same hyperparameter settings as the individually trained models. The two branches are concatenated and trained with a size of 128 samples per batch and for 15 epochs. A final dense layer with 33 units and softmax activation function is also added to make the classification. The idea is to keep the settings as similar as in the individual training, even though other hyperparameter settings is tried out.

The combined model performs better both in accuracy and loss. Accuracy and loss for the held out test set is 92.82%/0.26 for the combined model. Compared to the Raw model and MFCC model, this is an improvement by 1.85%/-18.8% and 2.67%/-23.5% with respect to the individually trained models.

The resulting performance for all the three models is shown in table III and the training performance for the combined model also in fig 4.

TABLE III
PERFORMANCE OF MODELS

	Raw model	MFCC model	Combined model
Train accuracy	91.66%	91.60%	**94.07%**
Val. accuracy	91.13%	91.19%	**92.71%**
Test accuracy	91.13%	90.41%	**92.82%**
Loss	0.27	0.28	**0.19**
Val. loss	0.31	0.31	**0.25**
Test loss	0.32	0.34	**0.26**

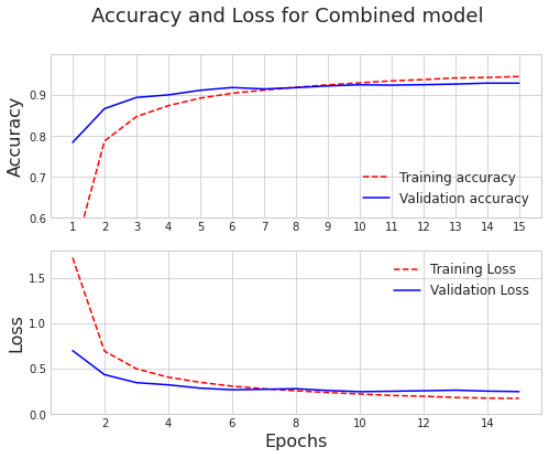

Fig. 4. Combined model, accuracy and loss

D. McNemar hypothesis test

To verify the result, McNemar hypothesis test is used. Two contingency tables are produced from the results of the model predictions on the test set. One where the combined model is compared to the raw model, and one where the combined model is compared to the MFCC model.

The null-hypothesis, ”the compared models perform the same with regard to proportion of errors” can be rejected in both cases. The p-value for the McNemar test results in 0.000012 and 0.00000012 respectively, which is far from a typical threshold p-value of p=0.05. The contingency tables and results from the McNemar test are shown in table IV and table V.

[13]

E. Log of time spent for this project

The distribution of time spent for different tasks in this project is shown in table VI.

TABLE IV
MCNEMAR TEST, CONTINGENCY TABLES

	Raw model correct	**Raw model incorrect**
Combined model correct	8977	429
Combined model incorrect	310	451
	MFCC model correct	**MFCC model incorrect**
Combined model correct	8987	419
Combined model incorrect	279	482

TABLE V
MCNEMAR TEST, P-VALUES AND SCORE

Combined model compared to	**p-value**	**Statistic (Chi-score)**
Raw model	0.000012	19.16
MFCC model	0.00000012	28.08

V. CONCLUSIONS

In this project three models have been presented. Two models with different view of the same input. One model kept its raw audio format, and one applied a MFCC feature extraction algorithm prior to CNN model training. The third model that had access to both the preceding models' features is clearly stronger in performance. This is also shown with the McNemars test.

The field of voice recognition is interesting and to produce even more accurate models, further experiments should comprise of adding more models together. Models with different knowledge about features other than given by MFCC algorithms. A thorough study of what features to capture would need prior studies closer to the field of human production and human perception, of audio signals. This would be very interesting to go deeper into.

TABLE VI
DISTRIBUTION OF TIME IN THE PROJECT

Task	**Hours**
Research previous work/getting ideas in addressing the task (articles/books/web). Choice of algorithms, preprocessing etc.	50
Programming / Coding	20
Lab / testing configurations	35
Write report	30
Total	**135**

REFERENCES

[1] "Tensorflow speech recognition challenge - kaggle," https://www.kaggle.com/c/tensorflow-speech-recognition-challenge/overview, 2018.

[2] L. Muda, M. Begam, and I. Elamvazuthi, "Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques," *arXiv preprint arXiv:1003.4083*, 2010.

[3] F. Chollet *et al.*, *Deep learning with Python*. Manning New York, 2018, vol. 361.

[4] A. Chowdhury and A. Ross, "Fusing mfcc and lpc features using 1d triplet cnn for speaker recognition in severely degraded audio signals," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1616–1629, 2019.

[5] T. Anvarjon, S. Kwon *et al.*, "Deep-net: A lightweight cnn-based speech emotion recognition system using deep frequency features," *Sensors*, vol. 20, no. 18, 2020.

[6] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[7] "Google ai blog: Launching the speech commands dataset," https://ai.googleblog.com/2017/08/launching-speech-commands-dataset.html, 2017.

[8] F. Camastra and A. Vinciarelli, *Machine learning for audio, image and video analysis: theory and applications*. Springer, 2015.

[9] F. Chollet *et al.*, *Deep learning with Python*, vol. 361.

[10] F. Camastra and A. Vinciarelli, *Machine learning for audio, image and video analysis: theory and applications*. Springer, 2015.

[11] H. Shinde, "Audio processing and speech classification using deep learning - part 1," https://medium.com/@harshalshinde_35237/audio-processing-and-speech-classification-using-deep-learning-part-1-434171b3c508, 2019.

[12] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[13] "Statistical significance tests for comparing machine learning algorithms," https://machinelearningmastery.com/statistical-significance-tests-for-comparing-machine-learning-algorithms/, 2019.