TensorFlow Speech Recognition

Petra Brčić
Applied Mathematics
petrabrcic94@gmail.com

Ivan Čeh
Computer Science
ivan.ceh1234@gmail.com

Sandro Lovnički
Computer Science
lovnicki.sandro@gmail.com

June 27, 2018

Abstract: We are witnessing the significant development of the blooming relationship between science and technology. Humanity is striving on making everyday life easier and more controllable by the integration of the intelligent machines. We want to control those machine easier, such as by spoken commands. This is where this project comes in and answers that question.

Keywords: tensorflow, speech recognition, audio recognition, feature extraction, MFCC, convolutional neural network.

I. Introduction

This project was given by Kaggle as Tensorflow challenge[1], where the task is to correctly classify the audio recording of a spoken words. In this paper, we discuss audio transformations needed to extract shared features to feed the convolutional neural network classifier and present results we obtained on couple of examples. All of the code as well as usage instructions can be found at our GitHub repository[2].

II. Data

Dataset was given by Tensorflow as one second audio clips of different words spoken by different people. Dataset can automaticly be downloaded when starting the train part of machine learning solution to the problem. The goal is to have a model that tries to classify a one second audio clip as either silence, an unknown word, "yes", "no", "up", "down", "left", "right", "on", "off", "stop" and "go", even though more words can be found in the dataset and included in the training process. Each of the words is separated into files named as the recordings of the word they present. In order to make these clips as realistic as they would be in real life, the background noise is added in preprocess part of the code. There is a file of few background noises within the dataset that is downloaded. Each audio file is encoded with the id of the person who recorded the word, followed by nohash and a number from 0, specifing the time the same person recorded the word. Apart from the audio files that take up the majority of the data file, testing and validation lists can be found too. Each of those text file contains a list of the audio clips so that dataset can be patitioned in training, validation and test set.

i. Preprocessing

In order to feed our neural network with the data, we first need to process it. Neural network is fed by the image, so we need to tranform the audio file to a spectrogram. Spectrogram is a visual representation of the spectrum of frequencies of sound or other signal as they vary in time. Spectrograms can be generated by an optical spectrometer, a bank of band-pass filters or by Fourier transorm. First, let us observe what the data looks like after reading the audio file.

[1] https://www.kaggle.com/c/tensorflow-speech-recognition-challenge
[2] https://github.com/Qkvad/SpeechRecognition

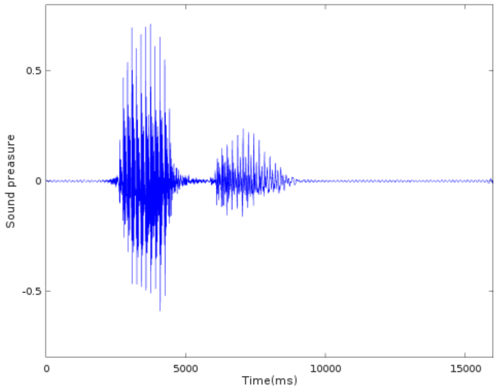


Figure 1: *Waveform for audio 'happy'*

Creating a spectrogram using FFT is a digital process. Digitally sampled data (which ours is), in time domain, is broken up into chunks, which usually overlap, and Fourier transformed to calculate the magnitude of the frequency spectrum for each chunk. Each chunk then corresponds to a vertical line in the image; a measurement of magnitude versus frequency for a specific moment in time (the midpoint of the chunk). These spectrums or time plots are then laid side by side to form the image or a three-dimensional surface, or slightly overlapped in various ways, i.e. windowing. This process essentially corresponds to computing the squared magnitude of the short-time Fourier transform (STFT) of the signal x(t) - that is, for a window ω,

$$spectrogram(t,\omega) = |STFT(t,\omega)|^2,$$

where

$$\begin{aligned} STFT\{x[n]\}(m,\omega) &\equiv X(m,\omega) \\ &= \sum_{n=-\infty}^{\infty} x[n]\omega[n-m]e^{-j\omega n}. \end{aligned}$$

The STFT is performed on a computer so it uses the FFT (Fast Fourier Transform), so all the variables are discrete and quantized.
After running our code in Octave to generate spectrogram yield the figure 2

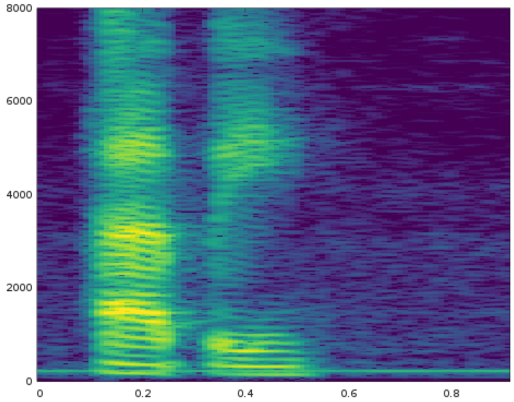


Figure 2: *Spectogram for audio 'happy'*

Our code works slightly different, first we scale the volume, then we shift it in time, add background noise and then calculate spectrogram. But for all the features of spectrogram to be easily visible on the paper form, we added the octave function to obtain the standard version of the spectrogram.
There is one more thing we have to dig into, that is getting from spectrogram to the image that is going to feed the machine learning algorithm. Final output of the preprocess is mel-frequency cepstrum [3] coefficients (MFCCs), the coefficients that make up an MFC. MFC is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. This frequency wraping can allow for better representation of sound in audio compression.
MFCCs are derived as follows:

1. Take the Fourier transform of (a window excerpt of) a signal
2. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows
3. Take the logs of the powers at each of the mel frequencies
4. Take the discrete cosine transform of the list of mel log powers, as if it were a signal

[3] Cepstrum (the name derived by reversing the first four letters of "spectrum", whereas operations on cepstra are labeled quefrency analysis, liftering or cepstral analysis) is the result of IFT of the logarithm of the estimated spectrum of a signal

5. The MFCCs are the amplitued of the resulting spectrum.

III. Machine Learning

In order to classify processed audio data into known classes ("up","down",...), we use a neural network model, specifically - a convolutional neural network (CNN). It is important to notice that audio samples corresponding to the same word need not to "look" the same. For example, one speaker may had began speaking later than the other. CNNs have had the most success in dealing with classifying such data and we say that CNN model is thus *spatially invariant*.

Specifically, as is reported in [1], CNNs outperform Deep Neural Networks which outperform Hidden Markov Model system, which is a commonly used technique for keyword spotting.
To put it simply and stress this fact's importance, we can write

$$CNN > DNN > HMM$$

where $>= \{(x,y) \ : \ x,y \text{ are classifiers}\}$ and $(x,y) \in >$ if and only if x performs better that y on a given dataset.

i. Convolutional Neural Network

We use a convolutional neural network model described in [1], with two layers or convolution. The task of convolution layers is to extract features in a spatially invariant manner and it is their output that finally goes into fully connected hidden layers such as in ordinary neural network.

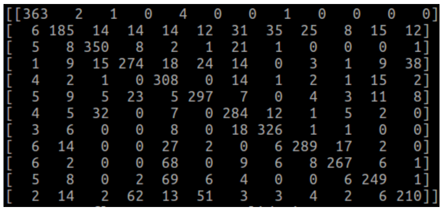
ii. Training

To be honest, training a convolutional neural network with our dataset of over 100000 audio samples was not a walk in the park. On a fairly strong laptop, it took us about 19 hours for 15000 training steps with learning rate of 0.001 and 3000 training steps with learning rate 0.0001 for finner descent.

Some of the important parameters for training the model are: `wanted_words`, `batch_size`, `learning_rate`, `how_many_training_steps`, `clip_duration_ms`, `data_dir`, ...

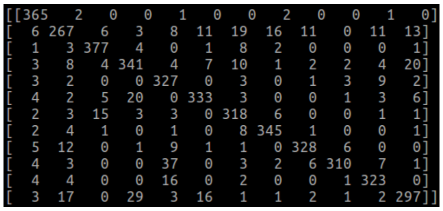
Most important parameters that control the preprocessing are: `background_volume`, `silence_percentage`, `unknown_percentage`, `time_shift_ms`, `window_size_ms`, `window_stride_ms`, `dct_coefficient_count`, ...

Let us observe the confusion matrices and accuracy of our model after 4000 steps and at the end of training. Classes are listed in order: silence, unknown, yes, no, up, down, left, right, on, off, stop, go.



```
[[363   2   1   0   4   0   0   1   0   0   0   0]
 [  6 185  14  14  14  12  31  35  25   8  15  12]
 [  5   8 350   8   2   1  21   1   0   0   0   1]
 [  1   9  15 274  18  24  14   0   3   1   9  38]
 [  4   2   1   0 308   0  14   1   2   1  15   2]
 [  5   9   5  23   5 297   7   0   4   3  11   8]
 [  4   5  32   0   7   0 284  12   1   5   2   0]
 [  3   6   0   0   8   0  18 326   1   1   0   0]
 [  6  14   0   0  27   2   0   6 289  17   2   0]
 [  6   2   0   0  68   0   9   6   8 267   6   1]
 [  5   8   0   2  69   6   4   0   0   6 249   1]
 [  2  14   2  62  13  51   3   3   4   2   6 210]]
```

Figure 3: *Confusion matrix after 4000 train steps*



```
[[365   2   0   0   1   0   0   2   0   0   1   0]
 [  6 267   6   3   8  11  19  16  11   0  11  13]
 [  1   3 377   4   0   1   8   2   0   0   0   1]
 [  3   8   4 341   4   7  10   1   2   2   4  20]
 [  3   2   0   0 327   0   3   0   1   3   9   2]
 [  4   2   5  20   0 333   3   0   0   1   3   6]
 [  2   3  15   3   3   0 318   6   0   0   1   1]
 [  2   4   1   0   1   0   8 345   1   0   0   1]
 [  5  12   0   1   9   1   1   0 328   6   0   0]
 [  4   3   0   0  37   0   3   2   6 310   7   1]
 [  4   4   0   0  16   0   2   0   0   1 323   0]
 [  3  17   0  29   3  16   1   1   2   1   2 297]]
```

Figure 4: *Confusion matrix after 18000 train steps*

We obtained total accuracy of $88,4\%$. As expected, the two classes hardest to distinguish are "no" and "go".

iii. Application

Now, to see this trained model in action, we give it some easy and also some hard audio files to classify to demonstrate it's power.

First, the easy part, let's predict a word "yes" without much background noise. It's read audio file and corresponding MFCC that our model obtains are given in figure 5

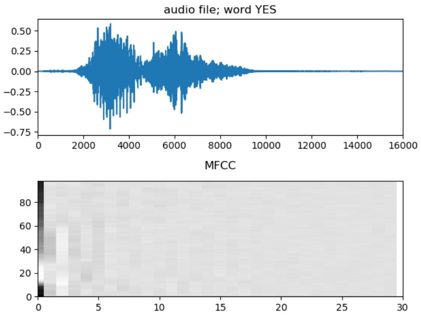


Figure 5: *Audio recording of a word "yes"*

The model prediction is as follows:

- yes (score = 0.99882)
- _unknown_ (score = 0.00062)
- left (score = 0.00052)

Now we feed our model with extremely noised and shifted audio data for "yes".

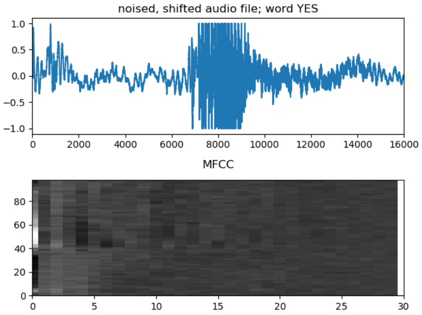


Figure 6: *Extreme audio recording of a word "yes"*

Although this is extremely distorted and noised, it seems that our model is pretty confident predicting the following:

- yes (score = 0.85262)
- left (score = 0.07611)
- _unknown_ (score = 0.02981)

IV. Different Approaches

References

[1] Tara N. Sainath, Carolina Parada. Convolutional Neural Networks for Small-footprint Keyword Spotting

[2] Shikha Gupta, Jafreezal Jaafar, Wan Fatimah wan Ahmad, Arpit Bansal. Feature Extraction Using MFCC

[3] Namrata Dave. Feature Extraction Methods LPC, PLP and MFCC In Speech Recognition

[4] Urmila Shrawankar, Vilas Thakare. Techniques for Feature Extraction in Speech Recognition System: A Comparative Study