Convolutional Neural Networks using Pytorch for Speech Classification

The objective of this project was to use machine learning techniques to classify spoken words. As this was a multiclass classification task, we decided a convolutional neural network would be most appropriate. The steps we took to implement this are briefly described below.

Computational experiments:

First we started exploring and visualizing our data. We are given 105835 audio samples in the wav format, of approximately a second long. There are 35 classes, with commands like left, two, yes. Sound waves are digitised by sampling them at discrete intervals known as sampling rate. The sampling rate of the files is 16kHz, meaning samples are taken 16000 times per second. The librosa package is used to analyse the audio data. Librosa.display and matplotlib can be used to display the individual waveform of the file. We can look for patterns between the different classes. From a visual inspection we could see that it is quite hard to visualize the difference between some classes. We have also checked the class distribution, this is pretty even. Approximately 2.9% per class. From the wav files as an experiment we have calculated and plotted the signals, fft, Filter bank coefficients and Mel-frequency cepstral coefficients.

We decided to work with the MFCC features provided in feat.npy. MFCC summarises the frequency distribution across a window size, so it is possible to analyse both the frequency and time characteristics of the sound. These audio representations will allow us to identify features for classification. In order to understand the features that we were going to work with, we started off with exploring the data to determine how to best process it. The main steps we took were: (1) match the paths of the files, and MFCC features with their corresponding labels, (2) ensure that the size of all the features were the same (99,13) and if they weren't, pad them (in the model that we submit, we did the padding within the model), (3) divide the data set to train and test set so that we could train and test our model on different subsets of the data (4) map each class of labels to a number, and convert the training labels to their mapped number. Later on, we mapped these numbered values back to their corresponding labels.

Learning algorithms and its parameters

We explored two ways to implement CNNs in python: using Pytorch, and Keras and Tensorflow backend. We were able to get a working model using both of these approaches. However, in this report, we will focus on the methods we used in the Pytorch as that is what we use for our submission. Since neither of us had prior experience with machine learning algorithms, we followed the approach taken by Chris Lovett (2018).We built the model from scratch, defining the functions that we need within a class. Initially we started with defining a basic multi layer perceptron and then refined it to use CNN. We started with a sequential model and built the model layer by layer. First we had the convolution layer. Here, we experimented with several features, in particular we tried using the rectified linear unit (ReLU) activation function vs the leaky rectified linear unit (Leaky ReLU) layer. The leaky ReLu function led to better learning rates so we used that. After convolutions, we applied the pooling layer. Specifically, we used MaxPooling 2x2, which performs a downsampling operation reducing the size of the input with max(). In order to control overfitting, we used the Dropout function, a regularization technique which

aims to reduce the complexity of the model. Finally, we applied a linear transformation to the data using nn.Linear.

Regarding the optimizer, we used the Adam's optimizer and experimented with different learning rates ranging from 0.001 to 0.01. We also experimented with different batch sizes, and epoch sizes. To measure the performance of our model, we used the cross entropy loss function. This function calculated the loss and also ensured that the final probabilities sum up to one.

The model required heavy computational power to run, so we were not able to run it many times. Additionally, we spent a lot of time on pre processing, so we did not have much time to test our hyper parameters. Our final accuracy was 68%. We think we would have been able to improve our accuracy by doing more runs and adjusting the hyper parameters.

1. Work done by group members

- Everyone: Every team member was involved in creating a plan for the project, figuring out how to preprocess the data, run the models, and determine how to change hyper parameters and writing and editing the report. In addition the specific methods and functions that the members contributed to are detailed below
- Camiel Leenders: Explored the data and possibility of creating our own features, defined the function to pad the arrays, did the pre processing and degugged to make it work for the keras model, researched how to build CNN model using Keras and Tensorflow backend and experimented with the functions, debugged the final model.
- Dennis Hokke: Defined functions to convert labels to numbers and the numbers back to labels, figured out how to create a dictionary to map the paths to their corresponding training labels. Researched how to build the model using Keras and Tensorflow backend and experimented with the functions, debugged the preprocessing code and the final pytorch model.
- Arati Sharma: Researched how to build and built the model from scratch in Pytorch. Debugged the pre processing code

Codalab

Account name: FlyingTigers, c.leenders@tilburguniversity.edu

References

Chollet, F. (2018). *Deep Learning with Python*. United States of America: Manning Publications Company.

Lovett, C. (2018). Training an audio keyword spotter with PyTorch. Retrieved 27 November 2019, from https://microsoft.github.io/ELL/tutorials/Training-audio-keyword-spotter-with-pytorch/

Lovett, C., Chuck, J. (2019, August 2). microsoft/ELL. Retrieved November 28, 2019, from https://github.com/microsoft/ELL/blob/master/tools/utilities/pythonlibs/audio/training/train_classifier.py