Applied Digital Signal Processing Final Project

© 2020 Konstantinos Kallidromitis, Frans Fourie (kk984@cornell.edu, fjf46@cornell.edu)

1 Project Description

The project aims to simulate the process of speech command classification, which is currently used locally in mobile devices to detect triggers such as "Alexa, Siri" that are required to connect with the cloud. A dataset that has passed various quality control methods is used to train a convolutional neural network to determine specific commands from multiple different people and accents. Then the goal of the project is to be able to take completely new samples, pre-process them and check the model using our own voices. To achieve this we will re-sample the signals and use the loudest section extraction method employed.

2 Project Motivation

The purpose of the project is to create a key word recognition program that can detect a set of keywords with high accuracy while being computational and energy efficient. This project is not aimed at competing with server voice recognition but rather to create a program that can run on local devices without being intrusive in terms of performance reduction of the device. The reason a program like this is important is it is unrealistic to send all audio a device receives to a server. The first reason for this is the fact that transferring data over the internet is prone to lag which would make the program feel unresponsive. The second reason is that transferring that amount of data unnecessarily over the internet would likely irritate users in terms of their data usage and lastly there would be major privacy concerns.

Thus, this project scope is constrained to key word recognition the program will be skewed to rather return false negatives than false positives. Thus, sentences or extremely noisy sound can be ignored when trying to recognize a word. The program will also be optimized to recognize a small finite set of words rather than having to be able to detect all words. Due to the program goal being to be used on a personal device it needs to still be able to identify the keywords given different peoples voices, personal device microphones (non studio quality audio), some background noise and people pronouncing the words in a casual manner not focusing on pronunciation [1].

3 Data-set

The "Speech Commands Data Set v0.02" data set used for this project consists of non studio captured audio to ensure they are applicable to real world samples. All 105,829 audio files are collected using crowd-sourcing from Google Brain. The audio samples are all in English and captured using phone or laptop microphones. Audio samples have a maximum length of 1 second and are single words and not part of sentences. There are 35 unique words captured in the dataset with multiple samples from different people in different environments of each word. To ensure all data files were of a high and acceptable standard 2 main tests were done. The first test to eliminate most incorrect audio files were to compress the audio files and look at the size of the compressed audio files. If a compressed audio file were below a certain threshold then it was disregarded as it meant it likely contained little to no actual audio information. The second test was then to listen to all the remaining audio files and if a human was not able to clearly

identify the word being said or if the word was not one of the desired 35 words then the audio sample was disregarded. The remaining audio files were all converted to contain PCM sample data at 16 KHz and were stored as WAV files. This file and sample combination makes further processing the easiest.

4 Preprocessing Methods

Before predictions can be done on our own recorded voice samples we need to perform some preprocesing to get our audio signal in the same format and to match the training dataset as closely as possible. The desired format for output audio files after preprocessing is a wav file with 16k Hz sampling rate and length 1 second. To ensure all input files are converted to this end some signal processing is used such as basic data conversions, denoising, Voice activation detection and audio signal padding.

4.1 Convert data

The first step in preprocessing the data is to make sure all the data are normalized and in the expected format before further processing is done. To this end data is converted to mono (1D) and also the amplitudes are normalized so that the largest amplitude in the signal has an amplitude of 1.

4.2 Denoising

We declare a cut off value that is determined by experimentation. In the current implementation when denoising is used this value is set to 5. In order to do denoising on the input audio file we first need to calculate the Fourier transform of the the input audio signal. Once we have the audio signal in the frequency domain it is easy to set frequencies with amplitudes lower then the cut off value to 0. The inferse Fourier transform of the signal is then calculated in order for the signal to be passed to the next stage of the preprocessing. In Fig. 1 the frequency domain representation of an audio signal before filtering is shown. In Fig. 2 the frequency domain representation of the same audio signal is shown but after denoising where all frequencies smaller than the cut off have been removed.

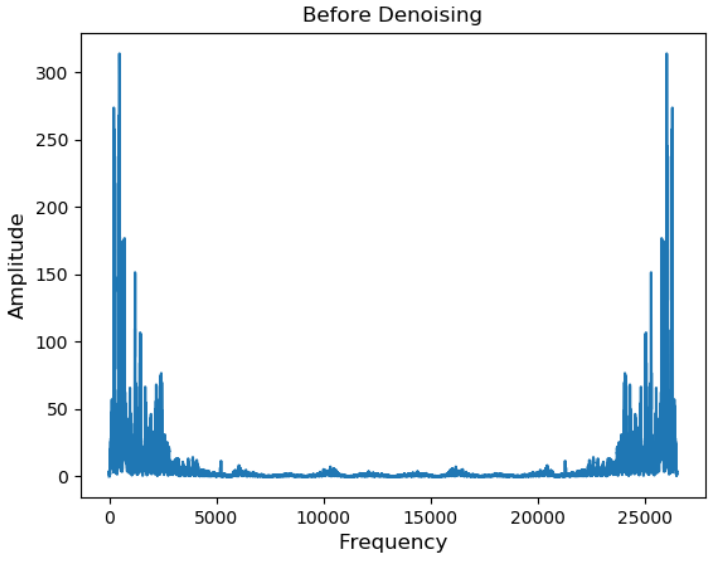

Figure 1: Frequency domain of signal before denoising

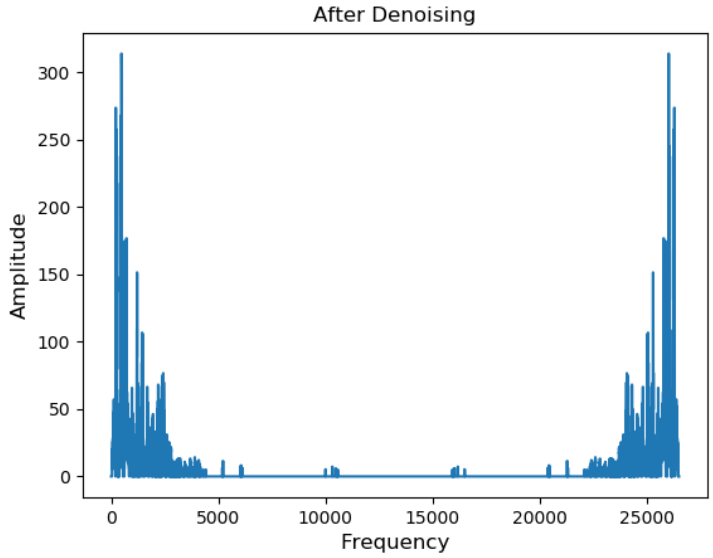

Figure 2: Frequency domain of signal after denoising

4.3 Audio Down Sampling

Since the input samples are of varying sample rate such as (48k Hz) and we want the sampling rate of all inputs to be 16k HZ we have to down sample the inputs. The first step in the down sampling process is to get the decrease ratio which is the current sampling rate divided by the desired sampling rate. Next we create a start and end pad that we add to the beginning and end of the data. Now we make use of the decimate function inside the scipy library to which the signal is given as well as the decrease ratio. The function returns the down sampled signal. The output signal is then extracted and returned.

4.4 Voice Activation Detection

The first step in detecting the start of speech is to create a dynamic threshold. The threshold value is selected as the average energy over the entire input audio signal. By then dividing the signal into windows and looking for the first window whose average energy is larger than that of the threshold we can take that window as the start of the speech. Since all voice snippets have to consist of only one word it is not necessary to identify the end of a word and start of a next word. Because of this it is simple to detect the end of the word by simply checking for the last window that had average energy larger then the threshold and taking that as the end of the word. Now by isolating the part of the audio file from the start window to the end window it is possible to extract only the word part of the audio file.

4.5 Audio Signal Padding

Since it is desired to have all the input audio files of the same length after the word has been isolated it needs to be padded to the necessary length. Based on the word lengths it was decided that all audio files should be 1 second long so since we are using a sampling rte of 16k Hz that means the signal should consist of 16k samples to have a length of 1 second. In an attempt to imitate real world scenarios the word is put at the centre of the 1 second audio file with zero padding before and after the word.

4.6 Preprocessing Results

Fig. 3 shows the audio signal when it was loaded in and Fig. 12 shows the signal after preprocessing have been done. Fig. 3 is the signal that is saved to a wav file that will be used for predictions.

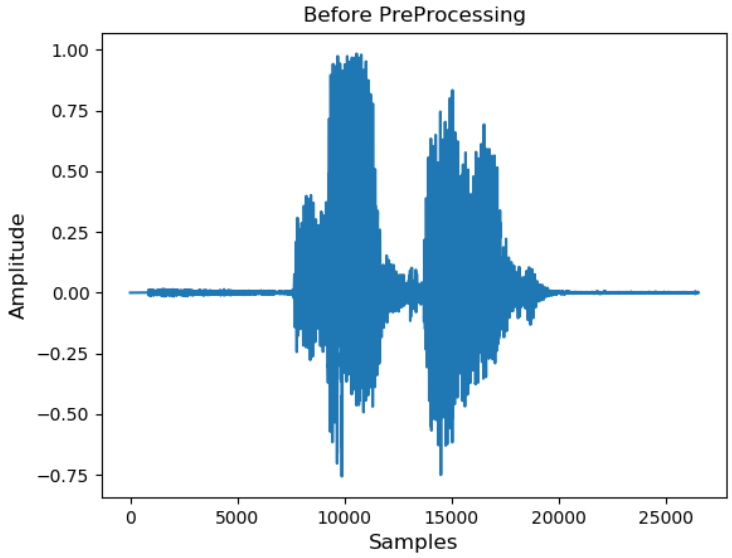

Figure 3: Audio signal before preprocessing

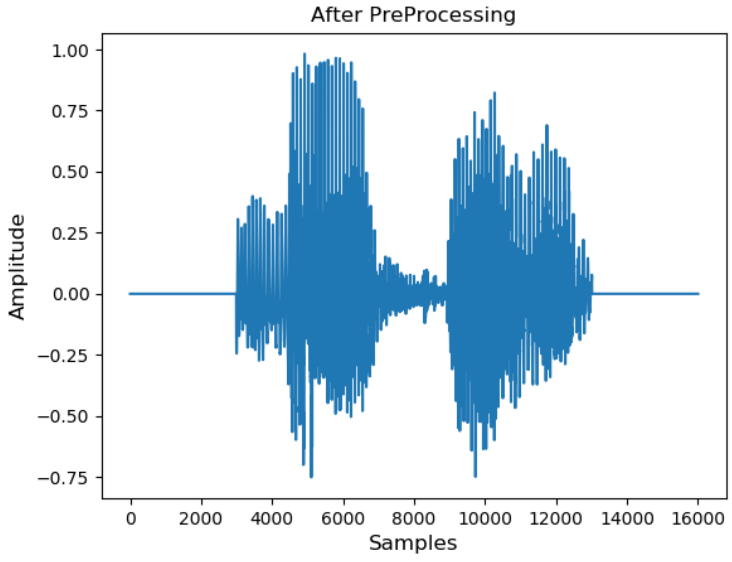

Figure 4: Audio signal after preprocessing

5 Convolutional Neural Networks

A neural network is trained to make predictions.The input is a vector of 1×16000 samples and the output is a 35 dimensional vector of pseudo-probabilities for each class. A CNN (Convolutional Neural Network) is used since it excels in detecting patterns. Figure 5 shows the architecture of the model. The first five layers are 1D convolutional layers with max pooling in between to reduce the covariance shift and speed up the training [2]. CNNs are able to extract feature maps by applying filters to the input. Each successive convolution the filters become more nonlinear and sophisticated to each class. [3] The there are 3 FC (Fully Connected layers) that use the filter output to make predictions and identify which word it is. Dropout is also an excellent method that is used in every layer. It randomly shuts down specific neurons to avoid over-fitting.

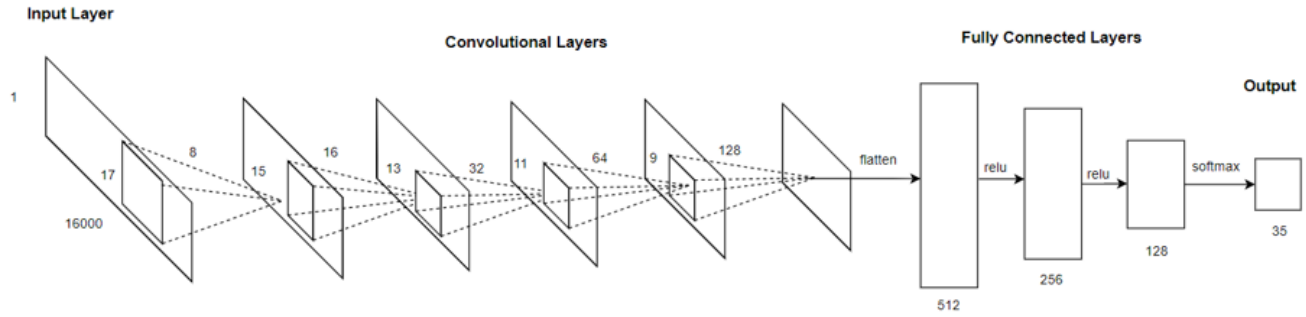

Figure 5: Neural Network Architecture

A cross entropy loss function is used with an Adam. Cross entropy is always the first option when performing a classification task and Adam is selected since it is known for being an extremely robust optimizer. The ReLU (Rectified Linear Unit) output is 0 when the input is smaller than 0 and linear afterwards. A ReLU activation function is usually the best choice because it creates nonlinear relationships between the nodes.

6 Results and Analysis

6.1 Existing Dataset

Training Accuracy = 0.8018
Validation Accuracy = 0.8356
Testing Accuracy = 0.8081

The final testing accuracy for each class is shown in Figure 7. The validation and testing accuracy are higher than the training because of dropout which allows the model to generalize well to new cases. Comparing Figures 6 and 7 there is a correlation between accuracy and number of samples. The higher the number of samples the better the accuracy for that specific class tends to be, for example: ”tree”, ”forward”. On the other hand, words such as ”backward” and ”zero” have a very distinct sound and are easy to distinguish so they have a high accuracy without the number of samples affecting the model. Similarly, words such as ”tree” which sound very similar to other like ”three” have a much lower accuracy.

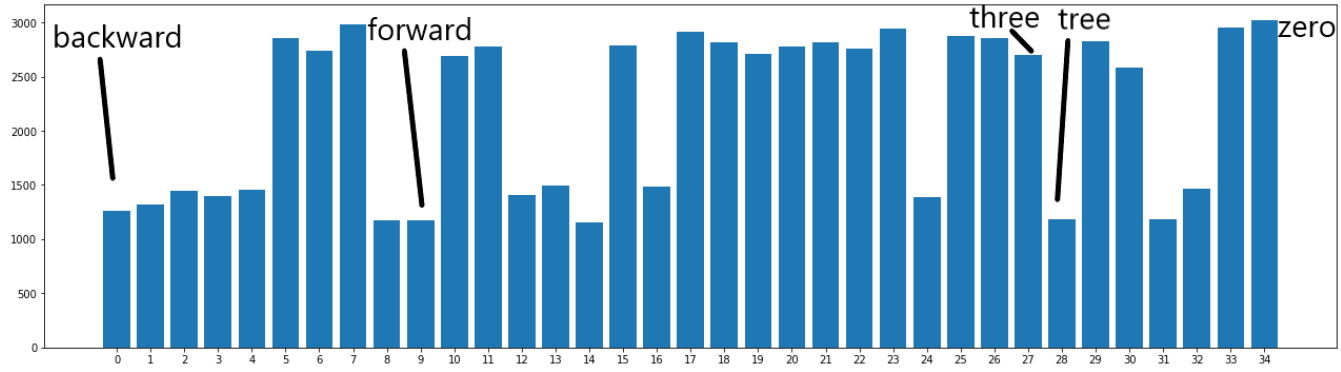

Figure 6: Number of Samples per class

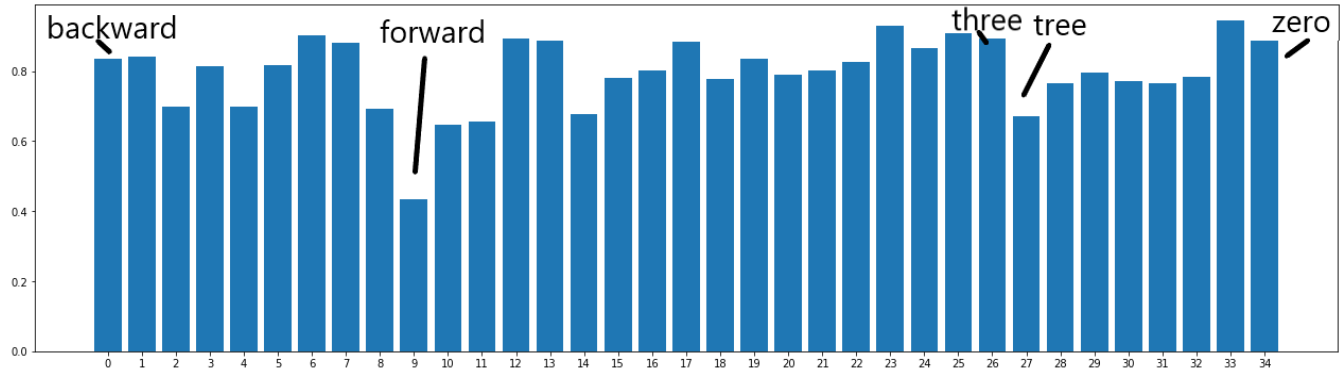

Figure 7: Accuracy per class

6.2 Personal Data

A personal dataset was also created with a total of 70 recordings, two per class, using our voices. The final accuracy was 0.8571 which is the result of good digital signal prepossessing techniques and better data quality. It is also worth mentioning that from the 10 false predictions, 8 of those had the correct prediction in the top 5 as seen in Figure 8, where tree and three are very similar words and difficult to distinguish.

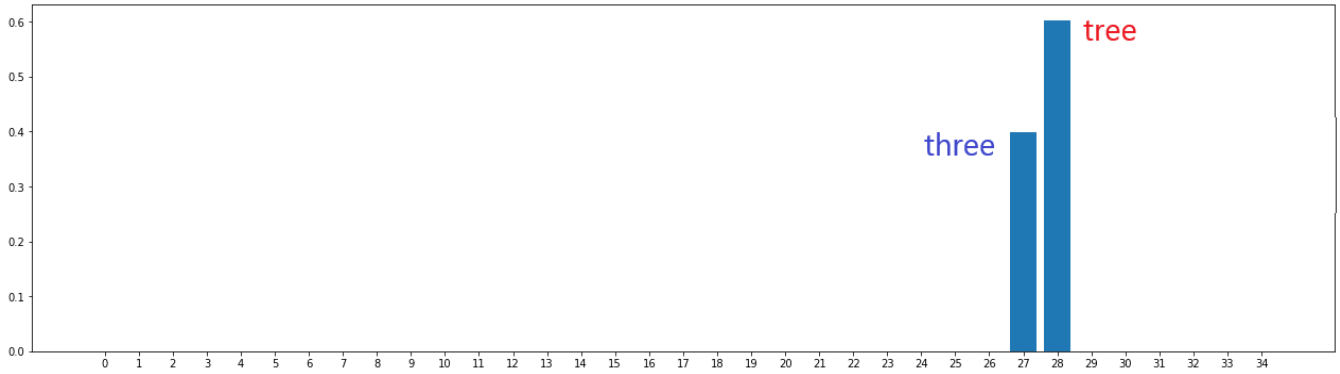

Figure 8: Predictions for word three

A more interesting example, is to investigate the worst prediction which came from the recording ”up1.wav”. From Figure 9 it can be seen that there is almost no confidence in the correct label and the model seems confused since multiple classes seem to have a high prediction value. In general the pronunciation can be divided into two sounds ”u” and ”p”. This is obvious by looking at a typical graph of the word shown in Figure 10. Figure 11 shows the sample that was miss-classified. The signal in Figure 11 has significantly more noise and worse microphone quality. The result is a waveform which doesn't resemble a typical ”up” word.

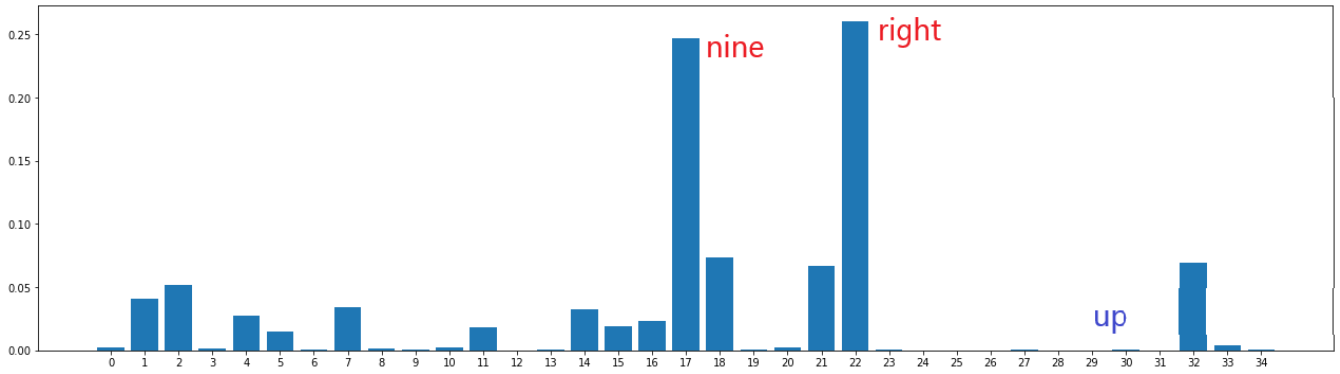

Figure 9: Predictions for word up

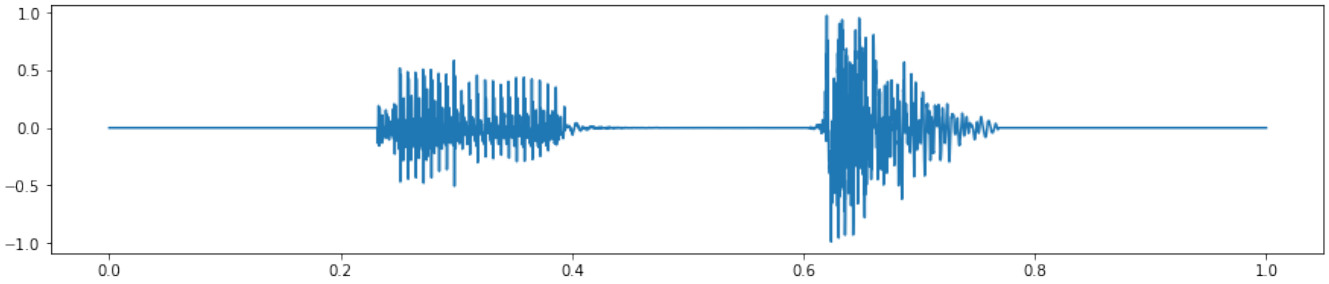

Figure 10: up.wav waveform showing distinct ”u” and ”p” sounds

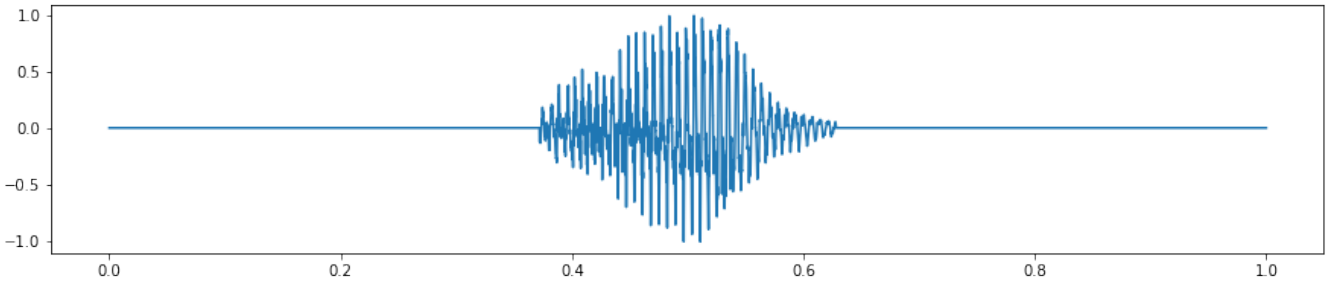

Figure 11: up1.wav waveform that was missclassified

Denoising was not used to get the final 0.8571 accuracy since it actually worsens the overall quality of the model. This is likely due to two main reasons. The first one is that the training data already has some noise and that means there is a form of denoising already existing in the network. Moreover, adding denoising might make it harder to distinguish between details in the patterns of the filters and thus exacerbate the separation of the classes. However, given sufficient noise such as this sample, using denoising is able to significantly increase the probability of the correct ”up” label in the top 3 (Figure 12). Using threshold of 10 for denoising the overall accuracy of our recordings was reduced to 0.7714.

7 Conclusion

In conclusion, the CNN was able to work well both with the existing dataset and the new recordings. Converting, Down Sampling, Padding and using a VAD was able to achieve an accuracy of approximately 0.85 on completely new data that was recorded with different techniques and format. Denoising was not sued in general since the prediction accuracy was worse, however it ended up being useful in extreme cases where the network itself is having trouble identifying the correct class.

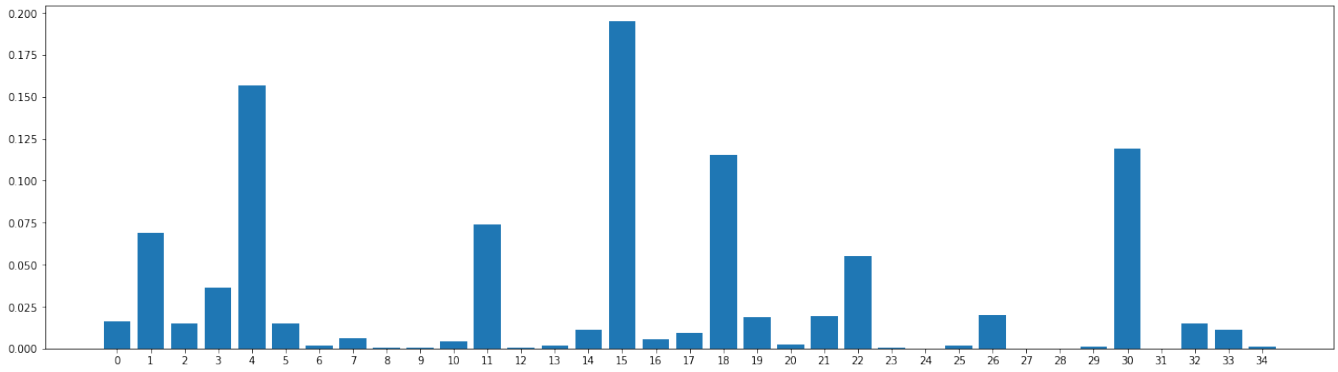

Figure 12: New predictions of up1 using denoising with amplitude 10

References

[1] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," arXiv preprint arXiv:1804.03209, 2018

[2] C. M. Bishop, Pattern Recognition and Machine Learning. Springer.

[3] A. Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. OReilly, 2017.