Speech controlled calculator

A project by Luka Železnik, Aditya Patel and Lars Vonk

January 24th 2024

Introduction

In this project we have researched if it is possible to train a model to classify speech to numbers from 0 to 9 and the basic math operators which are plus, minus, over and times. We will try to train this model, create a basic application to use this model and test this for the results.

This project was created by us since we needed a challenge that would use image or speech recognition and we wanted a project that could have real-world applications but also fall within our field of interest. Since the three of us are all computer scientists, we love math. This is why we came up with this project.

Hypothesis

Since we want to keep our dictionary of words small with only 14 characters, we think it would be very possible to create a model with a high accuracy. Collecting the data will be a challenge since with speech there are a lot of differences between male and female voices, different accents in English, background noise, etc. To classify these correctly we will need a broad set of training and test samples.

Methodology

To complete this project, we have used the programming language Python. Within this language exist many libraries that can aid us with the classification of speech. Since this is also a school project which aims to improve our knowledge of how speech recognition works, we will use the libraries only where needed.

The libraries which we have used for training the model are:

- Pytorch: for creating and training the model
- Librosa: for handling audio files
- SciPy: for help with training creating the training data
- Numpy: for data handling

Some other libraries and frameworks we have used for the prototype are:

- Flask: for the webserver
- Flutter: for creating the client

Aside from these Python libraries we have used the following Kaggle dataset for more training data.

https://www.kaggle.com/datasets/joserzapata/free-spoken-digit-dataset-fsdd

Furthermore, for the training data, we have created our own audio samples and asked some of our female acquittances (girlfriends) to also record some audio samples so we could have a broader set of training and test data with multiple genders and accents. We think our group covers a wide variety of accents since Aditya Patel and his girlfriend are from India, Luka Železnik and his girlfriend are from Slovenia, Lars and his sister are from the Netherlands and Lars' girlfriend is from Poland. Thus, covering three major language groups in accents (Germanic, Slavic, and Indic language groups).

Cumulatively we have collected 3056 samples for the training set, and 540 samples for the testing set. The dataset is not large enough for a truly robust system. We also tried to augment the dataset with mixed results. Because of the size of the dataset, we had some trouble with overfitting, which we managed with undertraining before validation loss would spike. With batch normalization, we also managed to stabilize the learning to get a satisfactory result.

Preprocessing and feature extraction

Firstly, we had to find the part of the audio with speech. We wanted a quick and computationally simple solution for this problem. For that, we used a method using autocorrelation coefficients. During speech, the autocorrelation coefficients form a damped sinusoid. Depending on the person those sinusoids are of different periods. So, we looked at the peaks of those and if those peaks are at the right range and consistent between them (standard deviation of samples between peaks). We used this to create a simple system, that says whenever a single frame (1/80s) has someone speaking or not. We needed to also keep in mind that the given segment must have enough energy (amplitude) to not just dismiss it as noise. Then we added a system, that even if a single frame is determined to not be spoken, the closeness of other spoken frames will mark it as such. We also wanted the end of words to also be encompassed, because mainly of the sound. Below we see an example of autocorrelation coefficients.

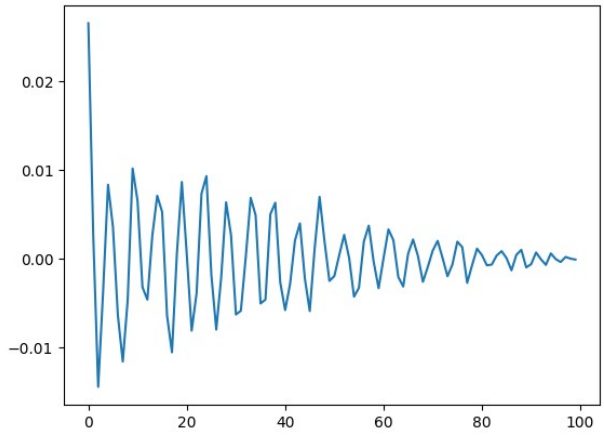


*Figure SEQ Figure * ARABIC 1: Autocorrelation coefficients during speech*

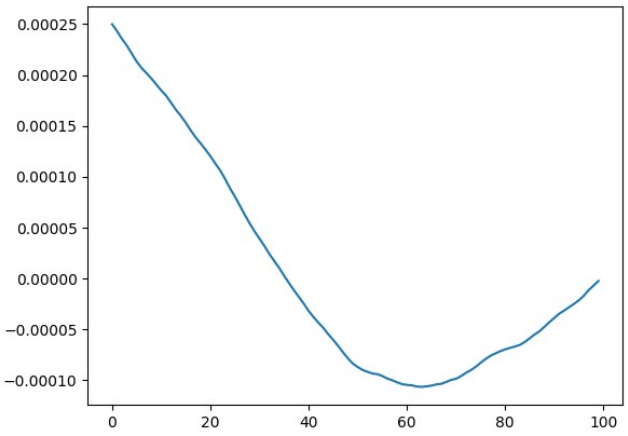


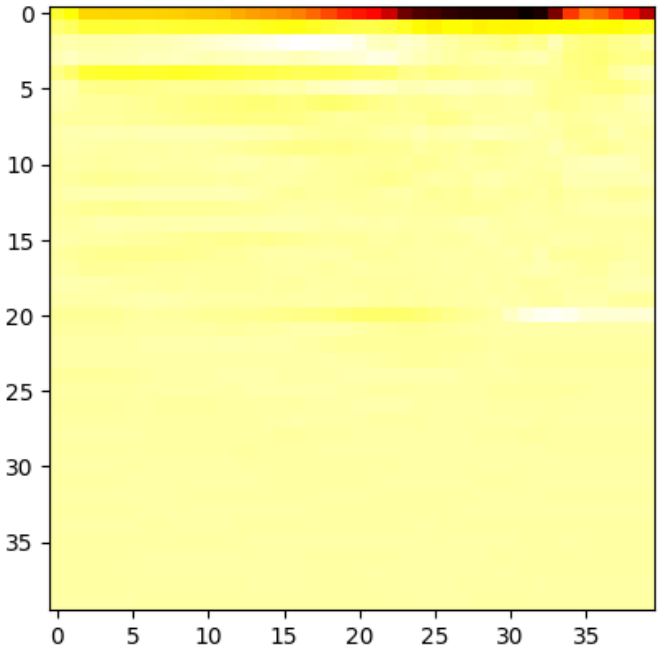
Figure : Example autocorrelation coeffiecients not during speech

We also resampled the sample to 8000Hz (we wanted a lightweight system, we don't need high harmonics for this task, and given data has this sampling rate), normalized the audio, and resampled it to 1 channel. Now that we get our short clip of a spoken word we can begin extracting features. We also applied a preemphasis filter, which reinforces higher frequencies and suppresses low-frequency noise.

Our main features are MFCC or Mel frequency cepstral coefficients. We calculated them using the librosa library. They are calculated by first transforming the audio signal to the frequency domain using the Fourier transform then changing it from linear to the Mel scale, which simulates human hearing, and finally calculating cepstral coefficients, which are the standard way of simulating human hearing.

We produced a single 2d array of 20 features per frame for 40 frames, regardless of the audio length. We then stacked this 40x20 grayscale image with its delta variant

(differences of coefficients over time) to get a nice 40x40 input image. We then put this into a convolutional network feature extractor to find our weight.



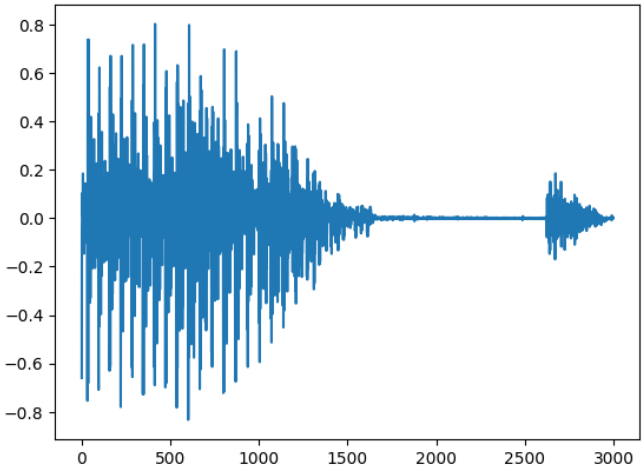


*Figure SEQ Figure * ARABIC 3: Cut*

*Figure SEQ Figure * ARABIC 4: MFCC*

We also calculated some secondary features, which were also used for classification. Those were the order polynomial coefficients of the sample in spectral domain, zero-cross rate and tone vs noise ratio. We calculated those for 15 frames per sample. We then added them up in the linear fully connected layer for an additional 45 features.

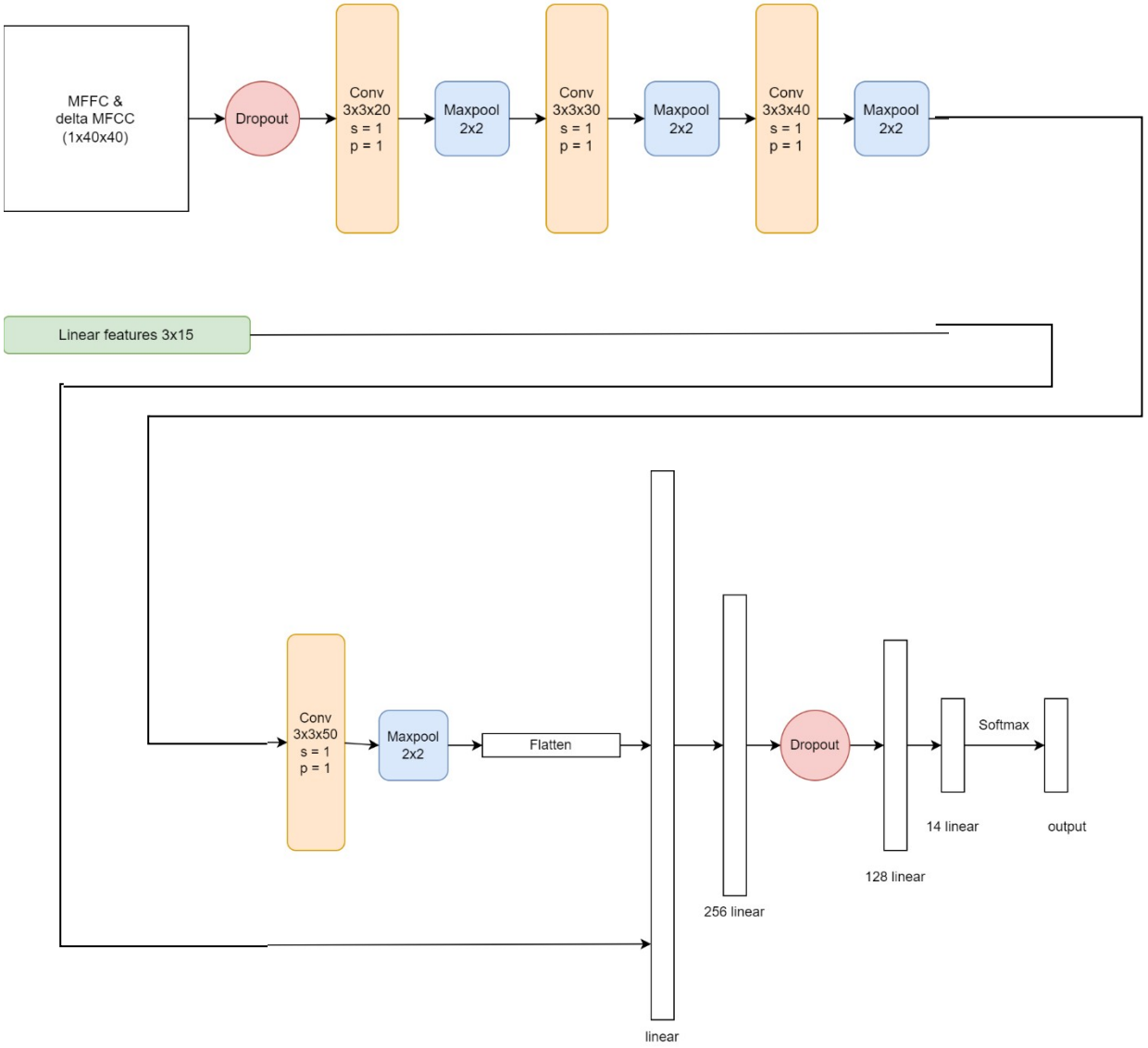
Neural network and training

For classification, we used a standard convolutional network. The main difference is that we introduced some precomputed features after the convolutional feature extraction part of the network. The first part of the network is comprised of 4 3x3 convolutional layers with padding and stride = 1. After each layer, we do Max pooling with a 2x2 kernel.

We used a ReLU activation function. Our learning rate was a constant 0.001, with MSE loss function. We trained with batch size 16 for 20 EPOCHS to not overtrain the network. After each EPOCH we shuffled the training data. We also used some dropout layers with 10% dropout to further prevent overfitting – first at input, and later in the fully connected part of the network. We slightly undertrained the network to also prevent overfitting.

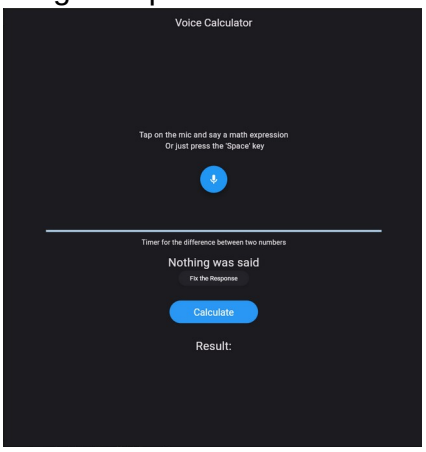
After that, we flattened our found features and added precomputed ones to have 3 fully connected layers with 256, 128 and 14 neurons respectively. The activation function is ReLU with the last neuron using softmax.

On the next page, we can see the full structure of the neural network.

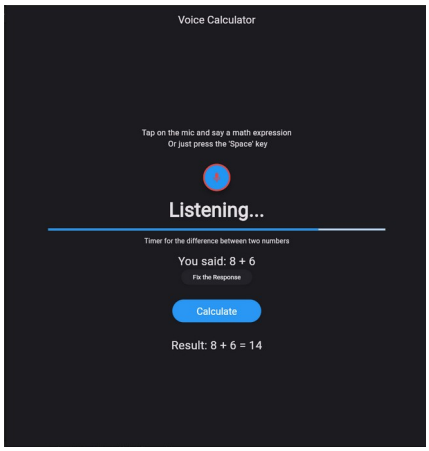

MFFC &
delta MFCC
(1x40x40)
Dropout
Conv
3x3x20
s = 1
p = 1
Maxpool
2x2
Conv
3x3x30
s = 1
p = 1
Maxpool
2x2
Conv
3x3x40
s = 1
p = 1
Maxpool
2x2
Linear features 3x15
Conv
3x3x50
s = 1
p = 1
Maxpool
2x2
Flatten
linear
256 linear
Dropout
128 linear
14 linear
Softmax
output

Prototype application

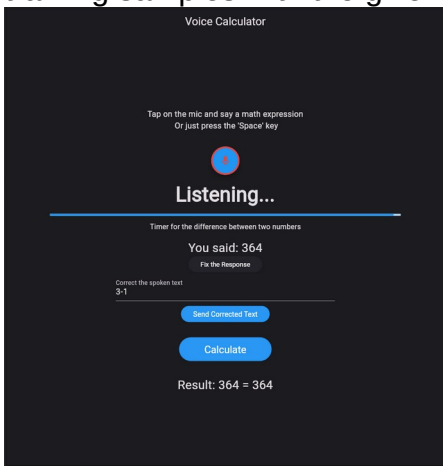
For our prototype application we have combined the above model with a small usable application with a frontend built in Flutter and a backend built in Flask. Since this is not part of the research this paragraph will only give a short overview of what was made.
For the frontend we made a simple application which can be run and used to input audio using the space bar or the microphone button.



After recording and pressing the “calculate” button the audio is send to the backend which cuts the audio in 3 second samples and sends them to the model to be classified. After this it returns the results in a string with the evaluated result of the calculation.

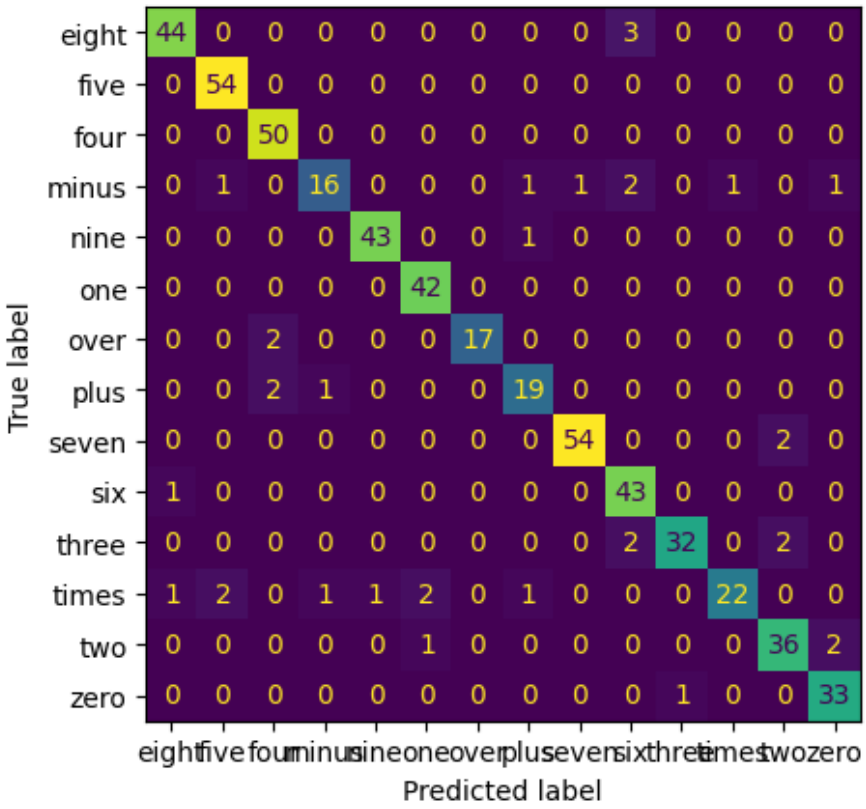


If the result is not good text can be send to the backend to correct the output, the input will then be saved for future training data by taking the split audio samples and saving them as training samples with the given input from the client.



Results

We managed to train the model to 93,5% accuracy on the test set. Below we see the confusion matrix of the model:



On the confusion matrix, we see that times and minus are misclassified to be something else. This is because we have a smaller dataset for those signs. We supplemented it, and made a system that lets us get training data while we use the app.

Future work

Since there was a time constraint, we had time to create a minimal viable product with some extras for our prototype. But there are some other nice-to-have additions that we did not have time for which are mentioned below:

- Create a stream of audio that is served to the backend and classified by the model so the output is shown live in the client while the user is speaking.
- Add extra operators so more advanced mathematical equations can be input
- Add a start and stop word to the dictionary so the calculator can truly be used hands-free.
- The classification quality can be increased by training the model on more samples. Also, some different methods that we have not tried yet could be experimented with to increase this accuracy.
- Greatly increase the size of the training set
- Rules that symbols can be only after numbers. If not true use the Markov model to find the best replacement.

Conclusion

After working on this product we have concluded that it is possible to create a model that can classify speech to a limited dictionary. However, it becomes hard to classify in different environments since the background noise and the acoustics make it hard to get audio that resembles the audio that the model was trained on. Also, a different tone of voice creates some problems even the voice of the same person in the morning and evening can differ. We can conclude that training a robust speech recognition model is difficult because of these differing circumstances.

This can be overcome by gathering a lot of training data in a lot of separate environments with a large number of different voice actors with diverse backgrounds and recording audio samples during multiple times of the day.