A music detector for hearing aids

Audio explorers - Software challenge
Herring Oats Team
`https://github.com/indianatoms/AudioExplorers`

Piotr Samir Saffarini Tomasz Mariusz Koziak Aleksander Nagaj

Abstract

In this project, we are trying to make the most efficient music binary classifier to hearing aids devices. Many solutions were evaluated in terms of their accuracy, memory usage and computational time. During the experiments, we discovered AI algorithms perform better than the conventional machine learning ones. We implemented the Convolutional Neural Network (CNN) that gave 98% accuracy on the validation data. We also examined the possibility of making the input data more compact by extracting different features based on log-mel spectograms as well as performing Principal Component Analysis (PCA). Finally, we presented the ideas of improving the model.

1 Introduction

Hearing aids help people around the world. Providing patients with the most advanced and reliable technology is one of the most important goals of this field. Noise muting with respect to relevant sounds like music or conversations still proves to be a current research topic [7]. In this project we focused on recognizing if music is present in an audio snippet. That classifier could be useful to later amplify this sound or to attenuate the noise when music is not played 1. Approach that we decided to take in this project was to classify two seconds samples in a form of 79 log-mel spectrograms. For that purpose we benachmarked many existing state of the art algorithms with the idea of finding the most appropriate one. Additionally, we took an effort of optimizing the classification scheme to fit the memory constraints of hearing aids.

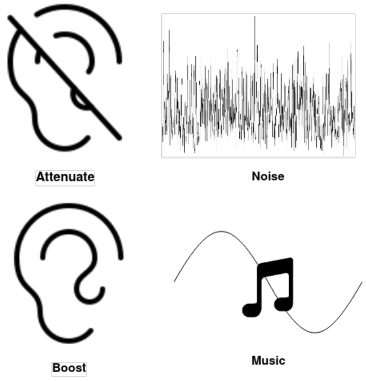


Figure 1: Overview of the desired solution

1.1 Problem Overview

The goal of the project was to develop a binary classifier that is able to recognize music to enhance the user experience of hearing aid devices and to make them more autonomous. Low computational power as well as limited memory of the device are the critical aspects that we had to take into consideration while doing this project.

Data was already converted from 2s audio samples to 79 log-mel spectograms (each consist of 25ms). The array was as followed: 10 500 samples x 30 frequencies bands x 79 time frames.

1.2 Project constraints

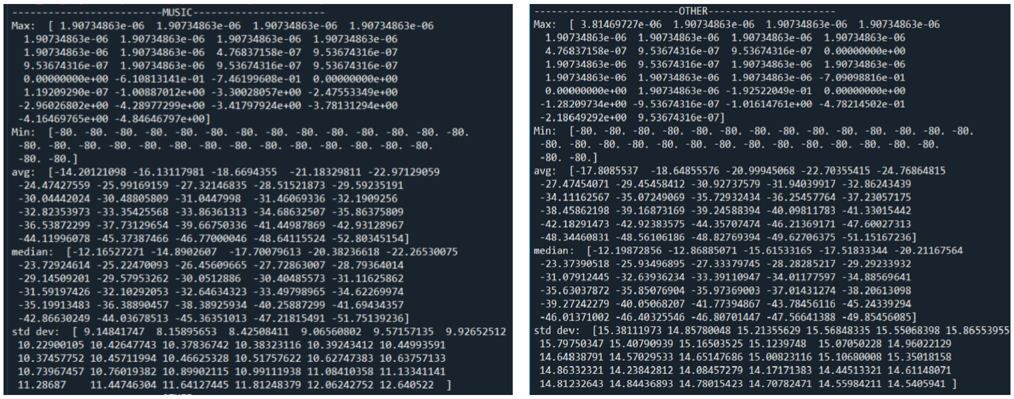
Memory: According to instructions, the device cannot store 2sec of audio, so more data efficient way is necessary. One 2sec sample is the size of 79x30. We assumed two variants: First is that it is feasable to store whole 2 sec sample with 79 log-mel spectograms, and second that more compact way of storing is necessary. We considered a real time computing variant which is not necessarily frame-by frame (1x30). Grouping e.g. 6 frames with PCA component reduction to 5 would give the same size (6x5 = 1x30) as frame-by-frame but with more time variation to the classification input.

Computational Power: On top of memory constraint, the device cannot perform complicated tasks due to CPU and battery limitations. Frame-by-frame computing is in contrary to this constraint, so we decided to also group frames into 5-10 and classify them all at once. This is where first variant (classifying 2 sec samples) has the advantage over the frame-by-frame computing.

2 Investigated Solutions

2.1 Getting to know the data

Before experimenting with the potential solutions to the problem, we run statistical analysis of the data in the format as one continuous stream of frames (samples*timeframes x frequencies). When comparing music data with other data, basic statistical features of each frequency bands were very close to each other (Figure 6. This indicated the usual frame-by-frame music recognition may not be feasible, especially in conventional machine learning algorithms. Bearing that in mind, instead of pure frame-by frame (1x30) we decided to investigate the whole spectrograms (79x30) or part of it, as it seemed that the detection of music samples is more possible with that approach.



```
-------------------------MUSIC-----------------------
Max:  [ 1.90734863e-06  1.90734863e-06  1.90734863e-06  1.90734863e-06
  1.90734863e-06  1.90734863e-06  1.90734863e-06  1.90734863e-06
  1.90734863e-06  1.90734863e-06  4.76837158e-07  9.53674316e-07
  9.53674316e-07  1.90734863e-06  9.53674316e-07  9.53674316e-07
  0.00000000e+00 -6.10813141e-01 -7.46199608e-01  0.00000000e+00
  1.19209290e-07 -1.00887012e+00 -3.30028057e+00 -2.47553349e+00
 -2.96026802e+00 -4.28977299e+00 -3.41797924e+00 -3.78131294e+00
 -4.16469765e+00 -4.84646797e+00]
Min:  [-80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80.
 -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80.
 -80. -80.]
avg:  [-14.20121098 -16.13117981 -18.6694355  -21.18329811 -22.97129059
 -24.47427559 -25.99169159 -27.32146835 -28.51521873 -29.59235191
 -30.04442024 -30.48805809 -31.0447998  -31.46069336 -32.1909256
 -32.82353973 -33.35425568 -33.86361313 -34.68632507 -35.86375809
 -36.53872299 -37.73129654 -39.66750336 -41.44987869 -42.93128967
 -44.11996078 -45.37387466 -46.77000046 -48.64115524 -52.80345154]
median:  [-12.16527271 -14.8902607  -17.70079613 -20.38236618 -22.26530075
 -23.72924614 -25.22470093 -26.45609665 -27.72863007 -28.79364014
 -29.14509201 -29.57953262 -30.0512886  -30.40485573 -31.11625862
 -31.59197426 -32.10292053 -32.64634323 -33.49798965 -34.62269974
 -35.19913483 -36.38890457 -38.38925934 -40.25887299 -41.69434357
 -42.86630249 -44.03678513 -45.36351013 -47.21815491 -51.75139236]
std dev:  [ 9.14841747  8.15895653  8.42508411  9.06560802  9.57157135  9.92652512
 10.22900105 10.42647743 10.37836742 10.38323116 10.39243412 10.44993591
 10.37457752 10.45711994 10.46625328 10.51757622 10.62747383 10.63757133
 10.73967457 10.76019382 10.89902115 10.99111938 11.08410358 11.13341141
 11.28687    11.44746304 11.64127445 11.81248379 12.06242752 12.640522  ]
```

```
-------------------------OTHER-----------------------
Max:  [ 3.81469727e-06  1.90734863e-06  1.90734863e-06  1.90734863e-06
  1.90734863e-06  1.90734863e-06  1.90734863e-06  1.90734863e-06
  4.76837158e-07  9.53674316e-07  9.53674316e-07  0.00000000e+00
  1.90734863e-06  9.53674316e-07  1.90734863e-06  1.90734863e-06
  1.90734863e-06  1.90734863e-06  1.90734863e-06 -7.09098816e-01
  0.00000000e+00  1.90734863e-06 -1.92522049e-01  0.00000000e+00
 -1.28209734e+00 -9.53674316e-07 -1.01614761e+00 -4.78214502e-01
 -2.18649292e+00  9.53674316e-07]
Min:  [-80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80.
 -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80. -80.
 -80. -80.]
avg:  [-17.8085537  -18.64855576 -20.99945068 -22.70355415 -24.76864815
 -27.47454071 -29.45458412 -30.92737579 -31.94039917 -32.86243439
 -34.11162567 -35.07249069 -35.72932434 -36.25457764 -37.23057175
 -38.45862198 -39.16873169 -39.24588394 -40.09811783 -41.33015442
 -42.18291473 -42.92383575 -44.35707474 -46.21369171 -47.60027313
 -48.34460831 -48.56106186 -48.82769394 -49.62706375 -51.15167236]
median:  [-12.19872856 -12.86885071 -15.61533165 -17.51833344 -20.21167564
 -23.37390518 -25.93496895 -27.33379745 -28.28285217 -29.29233932
 -31.07912445 -32.63936234 -33.39110947 -34.01177597 -34.88569641
 -35.63037872 -35.85076904 -35.97369003 -37.01431274 -38.20613098
 -39.27242279 -40.05068207 -41.77394867 -43.78456116 -45.24339294
 -46.01371002 -46.40325546 -46.80701447 -47.56641388 -49.85456085]
std dev:  [15.38111973 14.85780048 15.21355629 15.56848335 15.55068398 15.86553955
 15.79750347 15.40790939 15.16503525 15.1239748  15.07050228 14.96022129
 14.64838791 14.57029533 14.65147686 15.00823116 15.10680008 15.35018158
 14.86332321 14.23842812 14.08457279 14.17171383 14.44513321 14.61148071
 14.81232643 14.84436893 14.78015423 14.70782471 14.55984211 14.5405941 ]
```

Figure 2: Basic statistical features of each frequency band every frame

2.2 Baseline models

We tried implementing many various baseline classification models with different parameters to get the feeling how well they perform on our data. We discovered that in some machine learing models PCA components reduction gave better end results in terms of accuracy and/or size of prediction model.

Kmeans We used Kmeans for classifying the data with 2 centroids which would be 'music' and 'other'. Frame-by frame classification accuracy was like a coin-toss probability and PCA dimension reduction did not produce better results. We did not include the results of classifying whole sample as it was too big to compute on our computers, so definitely is not suitable for small hearing aids devices.

Logistic Regression Although it performed similar to Kmeans, logistic regression model proved to be very lightweight and provided 67% accuracy using sample by sample approach.

Random Forest Among all conventional machine learning models (excluding AI) we have tested [5] Random Forest gave the best results. Unfortunately, the memory size is too big to implement in hearing aids industry. Even after compression the model was over 300MB.

CNN Convolutional Neural Network gave the best preliminary results in testing. As there are many different AI solutions besides CNN, we decided to further explore this path.

Classification Model	**frame-by-frame**	**sample-by-sample**
KMeans	50% accuracy 6.3MB model	Model too big
Logistic Regression	49% accuracy 1KB model	67% accuracy 3KB model
Random Forest	88% accuracy 2GB model	Model too big
CNN	95% 10 frames with PCA total 37 features 1MB model	98% accuracy 3MB model

Table 1: Baseline models overview

3 Final Model

In the final solution we decided to implement the convolutional neural network (CNN) (Figure 3) as it achieved the best values in terms of accuracy (98%) when being ran on all 79 log-mel spectrograms in each sample. The network was trained on 13440 samples and validated on 3360 samples with a *batch size* of 32. It consists of two *convolutional* layers with kernels of size 3 × 3; each is followed by 2 × 2 *max-pooling* and 0.2 *dropout* layers. The *fully connected* layer is a 1 × 3456 vector which is followed by 64 and 32 *dense* layers which produce the output of the last layer, which uses *sigmoid* activation for binary classification. All other hidden layers use *ReLU* as an activator.

Bearing in mind that hearing aids are memory limited, we have also implemented CNN for just 10 time-frames with PCA component reduction, where we achieved the accuracy of 95%. In this approach, we are waiting for ten frames to be accumulated in the buffer and then the recognition will be performed. Thanks to that, we will be able to implement near real time recognition (5FPS) instead of taking 79 frames at once (0.5 FPS). At the same time, thanks to PCA, the input to CNN would be almost the size of one time-frame (37 features instead of 10x30). Thus, hearing aid device would not need to store additional data, therefore the memory constraint is satisfied.

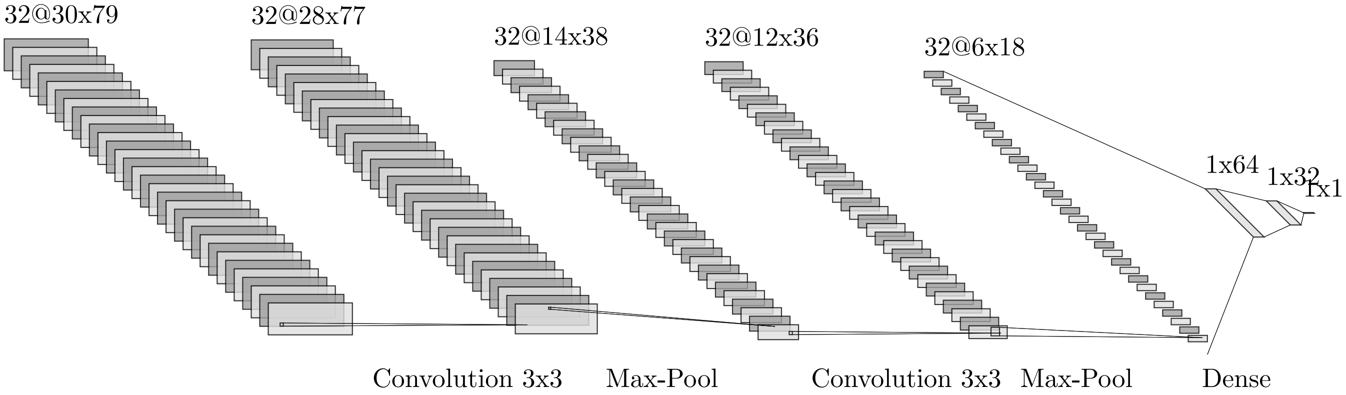


Figure 3: CNN architecture

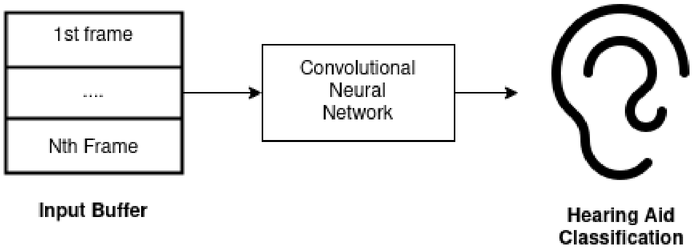


Figure 4: Final Model Detection

After fine tuning our models, we achieved the following results:

Classification Model	**Accuracy**	**Precision**	**F1 score**	**Recall**
CNN sample-by-sample	0.983	0.997	0.983	0.969
CNN 10 frames with PCA	0.948	0.981	0.947	0.915

Last but not least, in order to make the model more lightweight we decided to test it using integers insted of floats. With this approach we achieved the accuracy of 94%. Thanks to that conversion and assuming that the production model will be implemented in C we will be able to save 6 bytes of memory per variable.

4 Project Feasibility

The main trade-off that we tried to solve is efficiency vs. memory and power. To increase the accuracy, we took the whole sample at once. As the solution has to be more memory efficient, we tweaked the data with PCA that 10 frames consuming similar amount of memory as one original frame. Moreover, the buffer allows to trigger the classifier less frequently, thus saving the power 10 times compared to frame-by-frame classifier.

This solution may sacrifice a little bit of user experience, as the processing would be computed at 5FPS. This is where the power constraint comes into play - analyzing whole sample would save more energy, but sacrificing the UX.

It was also possible to compute PCA component reduction frame-by-frame, but then computations are 10x more often than doing it once after 10 frames. Both options are equally good and possible with our application, it just depends on the priority. We decided that we can sacrifice a little bit more memory for the sake of saving the CPU power and battery.

On top of taking into consideration model sizes 1 and input for the models we described throughout the 3, we measured the computational time of our classification model. We carried out the tests on Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz. Computing PCA for each sample took 0.28ms in total. CNN classifier takes 0.16ms. The computational times of CNN classifier are very similar when taking the whole sample as well as 10 frames as an input (0.15 ms) We are aware that this will take significantly more time on small hearing aid device, but also the code could be greatly optimized - coding with C instead of python and make operations that are prepared for the device's hardware.

Moreover, we can clearly see, that although PCA may result in saving the memory, it will consume almost 2x more power than the classification itself. Maybe extracting different features described in 5.2 is a path to explore.

5 Conclusions and Future Work

5.1 Conclusions

Two CNN models were developed, but as the one that takes all 79 frames at once perfoms better (98% accuracy) versus the one with PCA and 10 frames (95%). We are aware that the first model might not be memory-efficient, but as the data was given in this form, we decided to compute the final test results with this neural network. However, with PCA solution we developed, the accuracy is only slightly worse. In a long run, when only shorter samples are available and with the buffer we implemented or a sliding window, this would be the optimal solution.

After performing PCA on one frame, we discovered that first 6 principal components gives 95.2% of variance. We repeated the experiment on 10 frames and results are similar as seen on Figure 5. The fact that only 12% (or 20% in one frame case) of input data holds over 95% of information make us wonder that there might be more compact audio metrics that we could utilize. This is described in Section 5.2

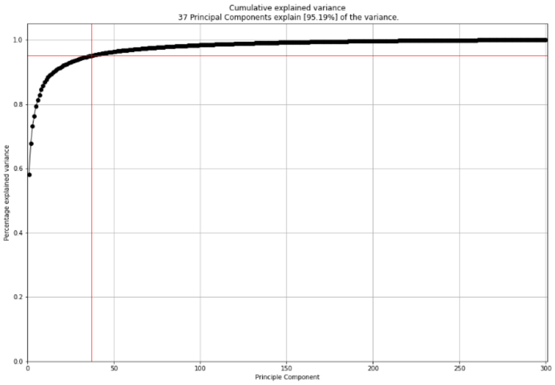


Figure 5: Principal component analysis of 10 frames[6]

5.2 Future Work

Transformer Neural Networks [2] could be a way to obtain even better results than CNN. They are better suited for sequential data such as text or music, as it makes the connections with the previous inputs. This is an improved version of LSTM [4] network. If we were given more time, this would be the main focus of our work to get even better results.

Besides given log-mel spectograms, we discovered **other metrics** that could be derived from audio signal or based on the spectogram itself. Some features are useful to process speech and singing[6]. Spectral envelope and detail, formants, MFCC [1]) are one of many [3] that could be explored more in depth. During the project we explored some with no improved results, but if raw audio files and more time, we believe this approach may bring valuable results.

On top of our CNN classifier solution, we could also optimize its performance on the whole samples by introducing **PID controller** on top of the buffer/sliding window. If 'music' was detected for a longer period, it would take more outputs classified as 'other' to fully 'convince' the model that music is no longer played. It could be useful if music had some parts that are only with lyrics, without the instruments. We believe that it would greatly improve the user experience. As it was not the essence of this project to implement it, we focus on other aspects of the problem instead.

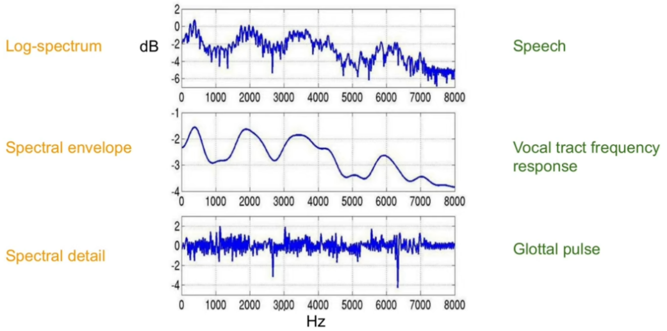


Figure 6: Speech features[6]

References

[1] Amit Meghanani, Anoop C. S., A.G.R.: An exploration of log-mel spectrogram and mfcc features for alzheimer's dementia recognition from spontaneous speech https://ieeexplore.ieee.org/document/9383491

[2] Ankit, U.: Transformer neural network https://towardsdatascience.com/transformer-neural-network-step-by-step-breakdown-of-the-beast-b3e096dc857f

[3] Feinberg, D.R.: Voicelab https://voice-lab.github.io/VoiceLab/

[4] Olah, C.: Understanding lstm networks http://colah.github.io/posts/2015-08-Understanding-LSTMs/?fbclid=IwAR0St715-AtEjh6eVlAP7DeMqkCtvHCyUd32-1iynsndQZQp6pdHN5tw39g

[5] Ortner, A.: Top 10 binary classification algorithm https://medium.com/thinkport/top-10-binary-classification-algorithms-a-beginners-guide-feeacbd7a3e2

[6] Velardo, V.: Mel-frequency cepstral coefficients https://www.youtube.com/watch?t=1294&v=4_SH2nfbQZ8&feature=youtu.be

[7] Venkatesh, S., Moffat, D., Miranda, E.R.: You only hear once: A yolo-like algorithm for audio segmentation and sound event detection. Applied Sciences **12**(7) (2022), https://www.mdpi.com/2076-3417/12/7/3293