Audio Classification
Using Deep Learning Architectures for an Application of Binary Instrumental Classification

Zach Halaby, Michael Marzec, Shayan Mukhtar
zhalaby3@gatech.edu, mmarzec3@gatech.edu, smukhtar7@gatech.edu

Abstract

The use of Deep Learning for applications in audio engineering is sparse, if not largely non-existent. The experiments outlined below attempt to begin asking important questions regarding model architecture, data sparsity, data presentation/format, and accuracy. Even after greatly simplifying the objective, deep-learning audio tasks present many issues when considering real-world scenarios. Though we did not produce a production-ready model we believe we addressed (and discovered) some important, foundational questions for future research. ***The code is available on all open branches on github (https://github.com/shayanmukhtar/cs7643_final)***.

1. Introduction/Background/Motivation

Gates are a commonly used tool in the music industry that allow a sound engineer to keep quiet and unwanted sounds from becoming part of the input signal. This is accomplished by setting a particular threshold of loudness for an input signal, and only allowing the signal to pass through if the loudness exceeds the threshold.

In audio engineering, there are a few important available settings: 1) threshold: the level of loudness the input signal must exceed to pass through 2) attack: how quickly the gate opens after the input signal exceeds the threshold 3) hold: how long the gate will stay open after the input signal is below the threshold 4) release: how quickly the gate closes after the input signal is below the threshold (applied after the hold has passed) 5) key: a frequency chosen by the audio engineer that the threshold is applied to 6) range: the amount the input signal is reduced when the gate is closed. If successful, a deep learning network could effectively model parameters 2-5, leaving parameters 1 (threshold) and 6 (range) for the engineer's discretion. See 'Section 6. Conclusion and Future Work' for further discussion of opportunities in future gate modeling.

Our work intends to lay the foundation to a first in the music industry: a gate that learns by example instead of careful tuning from an audio engineer. As such, this foundation entails developing and enhancing methodologies that approach audio classification; specifically, comparing and contrasting approaches that classify audio files via various deep learning network-based approaches.

1.1 Related Work

Audio classification as a task is less researched than its visual counterparts, although it seems many of the same architectures and lessons apply. Currently, the state of the art involves the reuse of highly successful CNN architectures on image datasets for the audio classification task. The following discusses related work on the same dataset that we leverage, which is further discussed in Section 1.2

CNN architectures for large-scale audio classification [1] achieves an accuracy of over 80% on its training data by repurposing CNN architectures such as ResNet and Alexnet. In addition, it uses the same dataset as our efforts and specifically restructures the entire audio clip into a space vector

that is fed into the model, removing the time component entirely. Per Google Scholar, this is the most widely cited paper on audio classification.

Large-Scale Weakly Supervised Audio Classification Using Gated Convolutional Neural Networks [2] again attempts classification but instead preserves the time component within their model. Specifically, they attempt to use convolutional recurrent neural networks, gated RNNs, and ensemble methods and achieve their best precision of 69.7 on their test set using their CNN ensemble method.

Finally, *LEAF: A Learnable Frontend for Audio Classification* [3] is a newly released discussion by Google AI. This work has promised similar performance (76.9% average accuracy) using state-of-the-art CNNs for both audio classification as well as audio generation, but with significantly faster training times. However, the architectural details are yet to be formally published.

1.2 Dataset

To complete our efforts, our team leveraged the 'Audioset' data provided by Google Research. This data consists of manually-annotated audio events – specifically, 632 audio classes broken into 2,084,430 10-second segments for over 5.8k hours of audio all drawn from YouTube videos. These classes include various domains including but not limited to human sounds, animals, sounds of things (e.g., vehicle, bell, alarm), music, and natural sounds.

Each manually annotated example was 10 seconds of audio, where each second of audio underwent dimensionality reduction using principal component analysis into its 128 most significant dimensions. Hence, the data comes in [example] x [10] x [128]. Each audio sample had up to 4 labels associated with it. For example, a single audio sample can consist of speech overlaying guitar, and hence that set would have 2 labels. The data was saved as tfrecords, which were converted to pytorch tensors. See Section 2. Approach for additional information on dataset manipulation as it pertains to modeling architecture and multi-class to binary classification conversion.

2. Approach

The original objective our team wanted to tackle presented us with many novel challenges that greatly influenced our experiment and design choices. First is the format of the presentation of real-world, real-time audio data, typically delivered in buffers of 100 samples and cached for enough time to gain insight from it. We also faced the challenge of labeling data: each buffer of 100 samples (up to 1,920 a second) must be labeled as "open" or "closed", and no such metric has yet been developed that could determine the answer to that. Moreover, the answer is not particularly easy for humans to discern, and would depend on the underlying mathematical assumptions made by the experiment designer. Assuming one has conquered the ability to label data at such a fine detail that it is individually imperceptible to humans, there still remains the question of how to quantify loss based on (best-case) an estimate of a gated signal. To our knowledge, we would have been the first in the world to create and use such data.

Given these primary objectives, the timeline of the project, and the expertise of our team, we revised our objective to focus on a preliminary task to all of the above: given a complete audio signal over time that has been pre-processed, can we (effectively) classify a human voice with minimal training data. This addressed many of the challenges one would face in an idealized laboratory setting before attempting to overcome the real-world challenges when working with audio in real-time. Namely, working with extremely clean data, working with well-defined and established loss metrics, but still overcoming the problem of real-world data sparsity.

Although the dataset and the related papers deal with multi-label classification, our initial problem statement involved isolating a single audio source from surrounding noise. Therefore, our approach was to convert the multilabel dataset to a binary classification problem. This was accomplished by converting the labels of each example to either positive or negative, depending on whether it contained the target class. Furthermore, because of the relatively small number of training examples (~24000 total samples in the training set) we also experimented with various forms of data

augmentation. Taking into account the dataset and the results of the state-of-the-art, our group decided to further investigate and compare various architectural performances. Note that all models are developed using the PyTorch library and all results can be found in the following 3. Experiments & Results section.

2.1 Linear

First, we analyzed a basic linear model in an attempt to baseline our results. We used a basic sequential model with a Linear – ReLu – Linear framework. While we hypothesized this result would provide relatively unstable results, with basic hyper-parameter tuning (e.g., batch size, learning rate, hidden layer size) it provided slightly better than expected results to be further discussed.

2.2 1-D Convolutional

Our team proceeded to test various combinations of a multi-layered convolutional network. With a 1-D convolution, the idea was to stride a 1-D kernel across the 10 seconds of audio all flattened together. We implemented an eight-layer network, which we proceeded to iteratively test by layer in addition to hyper-parameter tuning. The layered network leverages the following architecture: Conv1d – ReLu – Conv1d – Dropout – MaxPool1d – Conv1d – Dropout – Linear. As such, we first tested the Conv1d framework, then the Cov1d – ReLu framework and so on. We took this approach to both investigate the effects of various layers in addition to fine-tuning hyperparameters such as batch size, learning rate, kernel size and dropout rate.

2.3 2-D Convolution

In addition to assembling all 10 seconds of the audio features together, we experimented with stacking the seconds of audio on top of each other to form a 2-D data pattern. We then used visual convolutional models followed by linear layers to classify the audio samples. The hope here was that the model might "see" into the future and be able to leverage some information from the next timestep. The following design was used: Conv2d – ReLu – Conv2d – MaxPool2d – Conv2d – Flatten – BatchNorm – Linear – Dropout – Linear. Hypertuning was performed on the batch size, learning rate, kernel size, channel size, and dropout rate.

2.4 Recurrent

Our recurrent model involved the use of an LSTM network of variable length. The dataset was changed to add special <SOS> and <EOS> tokens (-1 and -2 respectively, which do not occur in the normal data) around each 10 second clip. The output of the LSTM network was fed into a linear classifier (the hidden state was thrown away). The input was 1 second of audio, and the clip was fed through the network with the prediction only being taken on the last second of audio.

2.5 Transformer

The transformer architecture reused the architecture from the class assignment with appropriate changes in parameters/dimensions including one critical difference: as we were working with pre-processed audio data, there was no way (or need) to use an embedding layer, so the embedding layer was ignored and the input was treated as its own "embedding".

3. Experiments & Results

All experiments were done using the training section of the dataset, with ⅘ of the set for training, and the other ⅕ reserved for cross-validation. The final accuracy metrics were calculated via the evaluation portion of the dataset, which contained entirely novel samples that were never in the training or validation sets.

3.1 Linear

In tuning the hyper-parameters for our linear classification model, we noted performance that was more consistent than expected across our various hyper-parameter combinations. See Table 3 within the appendix for detailed results from hyper-parameter tuning. In addition, see Table 1 in

the appendix for results from hyper-parameter testing across other schedulers (if not otherwise noted, all models leveraged an 'Adam' scheduler).

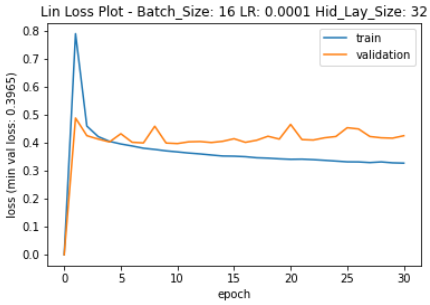

Figure 1. Linear Loss Plot

3.2 1-D Convolutional

Experiments here dealt with changing the starting kernel size as well as the other typical hyperparameters. Generally the smaller kernel sizes (in the range of 2-5) performed better than larger sizes (10-16). For the final set a kernel size of 3 was used. See Table 5 (Appendix) for detailed hyper-parameter results and Table 4 (Appendix) for detailed results from the aforementioned 'activated layer' experiments. In addition, the Adagrad and RMSProp schedulers were also tested, which can be seen in Table 2 (Appendix).

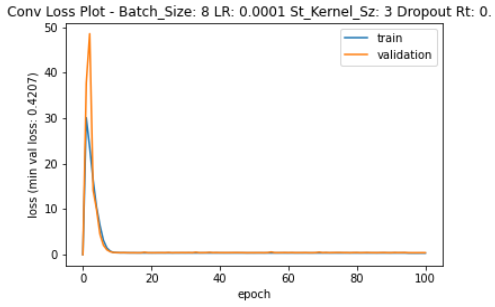

Figure 2. Convolutional Loss Plot (2 Active Layers)

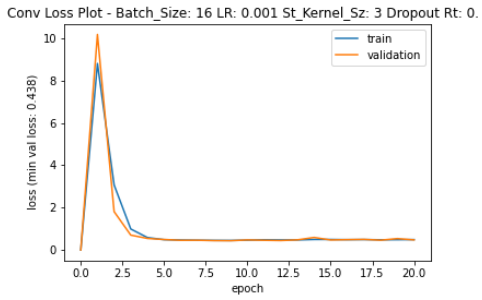

Figure 3. Convolutional Loss Plot (8 Active Layers)

3.3 2-D Convolutional

Experiments with the 2D Convolutional network saw generally the same results as the 1-D network. Smaller kernel sizes performed better (expected with the small size of the input) and an increase in hidden-layer channels past 4 led to extreme overfitting. The "best" model used a kernel size of 2 and a hidden-layer channel size of 4.

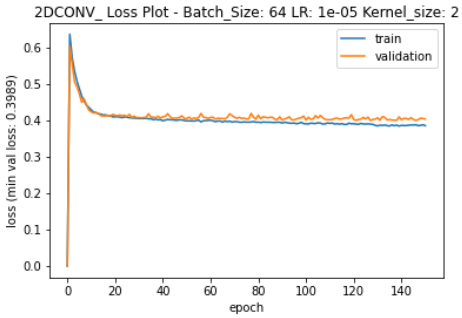

Figure 4. 2-D Convolution Loss Plot

3.4 Recurrent

Of all pursued models, our recurrent model performed the worst. Hyperparameter tuning made little difference from the stock parameters that were used. To our surprise, stacking LSTM layers had a detrimental effect after 4 layers.

3.5 Transformer

Tuning the transformer was unsuccessful as well, as it performed consistently given any reasonable hyperparameters. Like the others, it could overfit or underfit, but never exceeded the performance of the others.

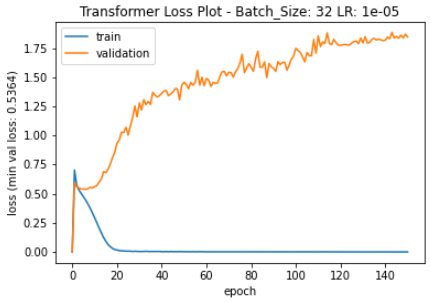

Figure 5. Transformer Loss Plot (Binary-CE)

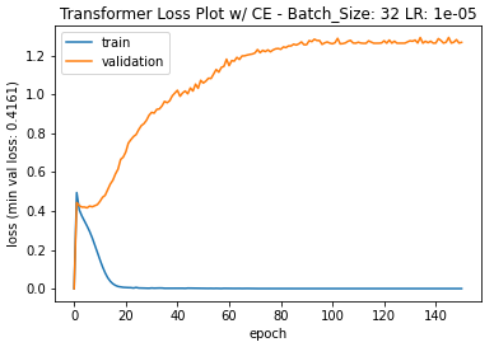

Figure 6. Transformer Loss Plot (CE)

4. Discussion

We set out to create an automatic gating mechanism to replace the hand-engineered electronic gates that exist today in the audio industry. Our models, however, did not perform as well as they would need to in order to be effective at this task. The state-of-the-art performance on this task is approximately 80% on the original multiclass label problem. Our best model achieves just a bit over that on the binary version of the problem.

Our expectation was that the linear model would perform the worst. However, the linear model actually performed remarkably well, especially considering its F1 score was so close to the much more complicated convolutional model (0.8 for the best linear model, 0.81 for the best 1-D convolutional). Our hypothesis for this was the limited receptive field of the convolutional model compared to the linear model. For the linear model, the entire 10 seconds of data was flattened into one tensor which was then fed through the network. In this way, the entirety of the dataset is visible to the network all at once. Though this approach is intractable on an image, the relatively small size of the inputs make it possible. This also points to the fact that the convolutional model was not able to pick up on features within the dataset effectively. Even though the last kernel had 128 channels, it seemed unable to learn an effective semantic representation of the data to correctly classify speech amongst the other sounds.

The 2-D convolutional model performed almost identically to the 1-D model. It was prone to extreme overfitting when the number of channels exceeded 4, and tended to underfit as the size of the kernel increased.

The recurrent model performed rather poorly, with an F1 score of just 0.75 after hyperparameter tuning. Whereas some papers trained on the raw dataset and simply threw out predictions made during the transitional phase between samples [4], we tried adding in <SOS> and <EOS> tokens surrounding the 10 second clip. Results from both approaches were similar, with the extra insertions marginally improving results. Since audio processing is dependent on temporal relationships, we assumed that adding in a recurrent layer would help with the classification task.

The transformer performed rather unremarkably and, similar to the linear model, was able to overfit/underfit based on the capacity we allowed it. Similar to the linear layer and the knowledge we gained from class, we hoped the ability to focus on certain frequencies in the timeline might help it outperform the others. It performed just as well as the other models, but no better.

Given the relatively limited nature of the dataset, we experimented with data augmentation as well. The data was augmented in two ways. Positive samples (samples containing the target class of speech) were replicated by adding and subtracting a constant from their audio. The upper and lower bounds of [0,255] were respected in this process, and samples were capped or floored at these levels. Furthermore, new samples were created by linearly combining other positive samples (0.5 of each sample, capped or floored at [0,255]). The resulting networks displayed better recall with the positive examples in the evaluation dataset, but were worse at classifying the negative samples. The overall F1 score was actually on-par with or less than when we used the data without augmentation.

We also performed a simple experiment where we trained the models on the evaluation data (and then tested against the evaluation data). Given all the best hyperparameters for each model, the transformer performed the best with no other modifications. The others performed almost exactly the same and, though we believed they could have performed better, it did lead us to some questions regarding the training and evaluation sets since we had proved the models were capable of learning what they needed for the evaluation task.

To explain the relatively poor performance of our classifiers for the task we had set out to accomplish, we have several theories. First, other researchers never converted the multi-class labels of the dataset to binary. This presents the network with a relatively difficult task of learning a target class in the presence of a variable number of other classes, without knowing those other classes, or even how many there are. Put simply, one example could be speech alongside a guitar, while another example could be speech alongside car traffic, car horns, and a crying baby. The original concept of creating an automatic audio gate was to train the network on clean examples of the target audio, and evaluate it with mixed samples to see if the target class could be caught. However, the dataset did not allow this. Furthermore, data augmentation may not have been done correctly, or in a way that would benefit the network. Since the audio samples were dimensionally reduced, the samples don't represent the raw amplitude of the sound wave. Adding or subtracting a constant may result in a sample that holds no semantic meaning for the target class anymore. There should, however, be a correct way to augment the data so that it retains the features of speech. For example, perhaps logarithmically transforming the data would improve performance. In addition, if we had access to the raw sound files for this dataset, we acknowledge that there may be an opportunity for a more direct and effective approach.

5. Work Division

All members contributed equally to the project. Shayan set up the initial jupyter notebook and the code for running the training loops and evaluations. Shayan also contributed to the linear model, the convolutional model, and the recurrent model. Michael greatly improved the jupyter notebook, made it compatible with Google Colab, and moved training to the GPU for greater speed. Michael also performed the majority of the hyperparameter tuning. Zach contributed the convolutional model, 2-D convolutional model and Transformer. He also was extensively involved in model hyperparameter tuning, loss-function experiments, and experiments to explain/validate causes for poor performance. All members were present during meetings and all contributed equally to the final report.

6. Conclusion and Future Work

To continue this research and find a deep learning model that can accomplish our task of real world audio gating, we would like to redo our experiments with the raw sounds files instead of the dimensionally reduced dataset made public by Google. We have several theories we'd like to try out, including training on the raw FFT of the data alongside the time domain version, and better leveraging recurrent networks on the raw data. It may also provide an easier means for data augmentation.

The task of creating an audio classification system that can discern a target audio class in the presence of variable noise is an important problem that can improve the quality of various fields, including audio engineering. Our experiments on publicly available datasets were not sufficient to create a model that can be used in the field, and future research is still required. However, we hope our work begins to highlight some of the challenges, questions, and research needs for the future of Deep Learning in audio engineering applications.

References

[1] S. Hershey et al., "CNN architectures for large-scale audio classification," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 131-135, doi: 10.1109/ICASSP.2017.7952132.

[2] Y. Xu, Q. Kong, W. Wang and M. D. Plumbley, "Large-Scale Weakly Supervised Audio Classification Using Gated Convolutional Neural Network," 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018, pp. 121-125, doi: 10.1109/ICASSP.2018.8461975.

[3] "LEAF: A Learnable Frontend for Audio Classification," Google AI Blog, Mar. 12, 2021. https://ai.googleblog.com/2021/03/leaf-learnable-frontend-for-audio.html (accessed Oct. 24, 2021).

[4] P. Gimeno, I. Viñals, A. Ortega, A. Miguel, and E. Lleida, "Multiclass audio segmentation based on recurrent

neural networks for broadcast domain data," EURASIP Journal on Audio, Speech, and Music Processing, vol. 2020, no. 1, Mar. 2020, doi: 10.1186/s13636-020-00172

Appendix

A. Model Performance Tables

Scheduler	Batch Size	Learning Rate	Hidden Layer Size	Best Validation Loss	F1
RMSProp	32	0.001	128	0.4372	0.738
RMSProp	16	0.0001	32	0.3982	0.7986
Adagrad	32	0.001	128	0.5015	0.7888
Adagrad	16	0.0001	32	0.5297	0.7442

Table 1. Linear Model Scheduler Experiments

Scheduler	Batch Size	Learning Rate	Dropout	Best Validation Loss	F1
RMSProp	16	0.001	0.3	0.4705	0.7268
Adagrad	16	0.001	0.3	0.5454	0.7743

Table 2. 1-D Convolutional Model Scheduler Experiments

Batch Size	Learning Rate	Hidden Layer Size	Best Validation Loss
16	0.0001	32	0.3965
32	0.001	128	0.3969
64	0.001	128	0.3971
32	0.001	64	0.3972
16	0.005	32	0.398
32	0.005	64	0.3987
64	0.001	64	0.3999
32	0.005	32	0.4002
32	0.0001	64	0.4025
32	0.0001	128	0.4112
64	0.0001	64	0.4118
16	0.00001	64	0.4218
32	0.00001	64	0.4576
16	0.00001	32	0.4672
16	0.005	64	0.4754
64	0.0001	128	0.4978
64	0.01	128	0.5651
32	0.01	64	0.5671
8	0.005	16	0.5866
32	0.00001	128	0.7456

Table 3. Linear Model Hyper-Tuning Results

Activated Layers	Batch Size	Learning Rate	Kernel Size	Dropout	Best Validation Loss
2	8	0.0001	3	0.2	0.4207
2	8	0.01	3	0.2	6.3515
2	64	0.001	3	0.2	0.4307
2	16	0.001	3	0.2	0.4337
2	32	0.0001	3	0.2	0.4353
2	8	0.001	3	0.2	0.4784
2	32	0.005	3	0.2	0.5171
2	32	0.001	3	0.2	0.4226
3	32	0.001	3	0.2	0.7558
4	32	0.001	3	0.2	4.4892
5	32	0.001	3	0.2	0.7767
6	32	0.001	3	0.2	0.5792
7	32	0.001	3	0.2	0.4604
8	32	0.001	3	0.2	0.4705

Table 4. 1-D Convolutional Activated Layer Experiments

Activated Layers	Batch Size	Learning Rate	Kernel Size	Dropout	Best Validation Loss
8	8	0.0001	3	0.2	0.8425
8	32	0.001	3	0.2	0.6498
8	64	0.001	3	0.2	0.8554
8	16	0.001	3	0.2	0.466
8	32	0.0001	3	0.2	0.5764
8	8	0.001	3	0.2	0.4735
8	32	0.005	3	0.2	0.5745
8	8	0.01	3	0.2	0.5687
8	8	0.001	3	0.1	0.4692
8	8	0.001	3	0.3	0.4631
8	8	0.001	3	0.01	0.4468
8	8	0.001	3	0.5	0.4518
8	16	0.001	3	0.01	0.4582
8	16	0.001	3	0.1	0.4711
8	16	0.001	3	0.3	0.438
8	16	0.001	3	0.5	0.473

Table 5. 1-D Convolutional Model Hyper-Tuning Results