Musical Instrument Identification From a Soundtrack

Roy Steinberg*

Rotem Elimelech*

January 26, 2023

Introduction

In both modern and classical music, very often there is a wide range of musical instruments playing in unison. As musicians ourselves, we feel the need for the correct classification of these musical instruments in real time.

As such, we came up with a neural network that can do just that - identify multiple musical instruments in any composition.

To do so, we built upon the ideas of previous projects[1][2], whose aim was to identify the dominant instrument in a musical piece. Which they did with good results - between 80% and 85% accuracy rate.

Dataset

In order to actualize our idea, we needed a good dataset. Luckily, we found one which completely fits our needs - MusicNet. This dataset includes hundreds of musical pieces (mainly classical), with multiple instruments playing frequently.

The labeling of MusicNet, sadly, was indexed in a way which less suited our needs. The labeling was centered around every single note and instrument, meaning that there were multiple timestamps which corresponded to the same time frame. While it was technically possible to work with this, it would require a lot of processing during the training of the model - something we were hesitant to include. So, we decided some pre-processing was in order.

We loaded our labels, and generated a new labeling system which focuses on each time frame individually, meaning each timestamp is included in the label, with no overlap with other ones, and including all instruments and notes playing in that specific time.

Model

To create our neural network we used a pre-trained ResNet-18 model, with the final layer replaced by a fully connected layer, with the number of outputs matching the number of instruments, and a sigmoid activation function after that.

*These authors contributed equally

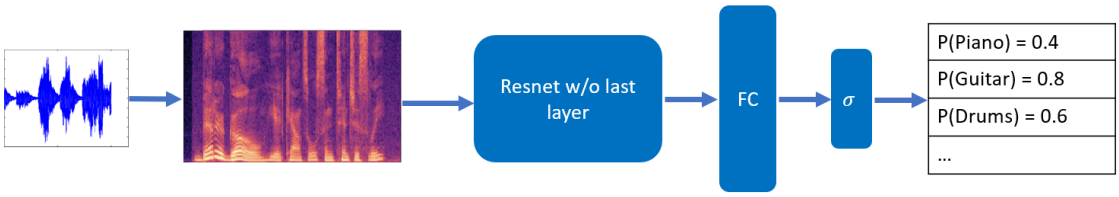


Figure 1: our model

As ResNet was trained on images in the first place, it makes sense to use an image as the input. Figure 1 shows the sequence for our model. A 1-d waveform is given as input, which is transformed into a spectrogram, so that we have an image to work with. This spectrogram is inputted to the pre-trained ResNet model, and we train the final fully connected layer.

But the most important question was how to measure a successful prediction? Also, the sigmoid outputs a vector of probabilities for each instrument, but we still need to choose which instruments are actually playing.

Loss Function

For the prediction, we realized that the well-known MSE loss may not be the best option in our case. After some checks on the validation set, we found that an MSE loss causes the model to predict that no instruments are playing - always. It was quite obvious why.

Our dataset includes 11 instruments, with at most 3 instruments playing at the same time, and the MSE loss counts how many predictions we got wrong. So, assuming that nothing is playing will be correct for 8/11 instruments at minimum. While this is technically correct, it is not useful at all to us.

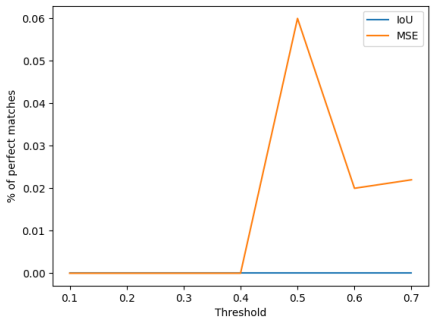
We understood we needed something to mitigate the effects of the false negatives that the MSE loss caused, so we took inspiration from computer vision, specifically image segmentation, and used something similar to the IoU loss.

$$loss = \frac{FN}{N_{playing}} + \frac{FP}{N_{not\ playing}} \tag{1}$$

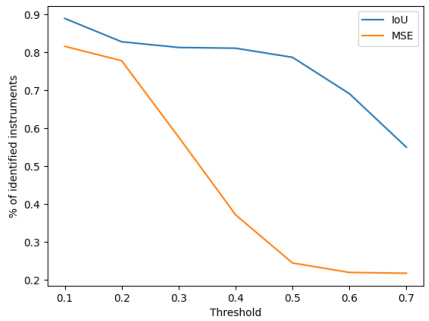
Equation 1 shows our implementation of the IoU loss. We take the false negatives and divide by the number of instruments that are playing in the time frame, and do the same for the false positives, but with the number of instruments not playing instead.

Figure 2 shows us the results on the validation set with the MSE and IoU losses, with the x-axis being the threshold in which we consider the sigmoid output as a prediction. The IoU loss gives a much better prediction no matter the threshold, identifying the correct instruments substantially better. Sadly, said loss also increases the number of false positives on average.

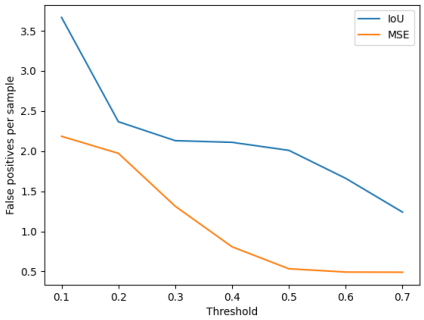
Ultimately, we chose to use the IoU-like loss, as the increase in accuracy ourweighs the increase of false positives.



(a) perfect matches



(b) accuracy



(c) false positives

Figure 2: effects of different losses

Dataset Bias

While checking our hyper-parameters on the validation set, we noticed a clear bias of our model to predict the first four instruments over any of the others. After checking some possible reasons for this, we found the main suspect - the database.

As can be seen in figure 3, the first 4 instruments constitute a large majority of the instruments in the database, leading our model to prefer these predictions over others.

To check if our model is as biased as the dataset, we inputted white noise into the model. Were our model unbiased, we expect to see the prediction of noise distributed evenly over all instruments. But the results were obvious - our model is clearly predicting the first four instruments with a much higher likelihood than the others. It was overfitting on the dataset.

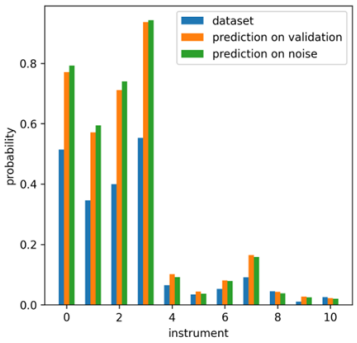
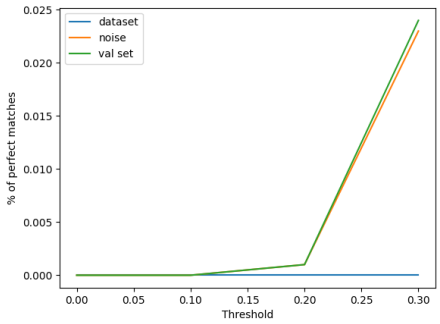


Figure 3: dataset bias and effects on our model

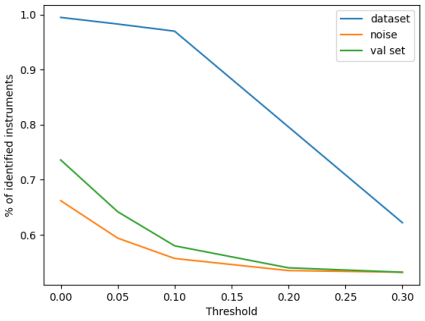
To mitigate these effects, we came up with two possible solutions.

Normalization

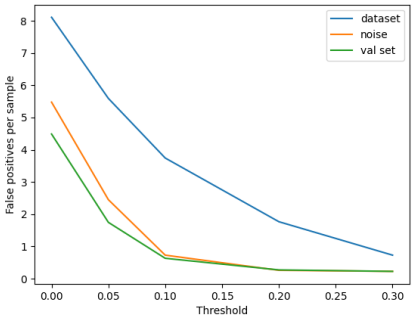
The first option we chose was to simply normalize our data by subtracting the empirical expectancy. This also lead to us adding class-specific thresholding.



(a) perfect matches



(b) accuracy



(c) false positives

Figure 4: effects of normalization

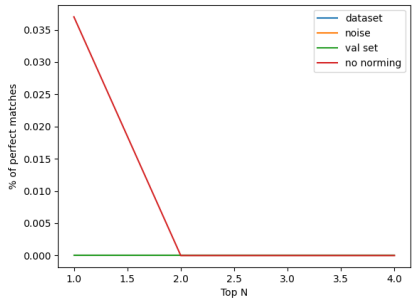
Figure 4 shows that ultimately, the improvement in this case is quite negligible, if not worse than the original, un-normalized dataset. Although we get less false positives, our model is now quite bad at predicting the correct instruments.

Top N Classes

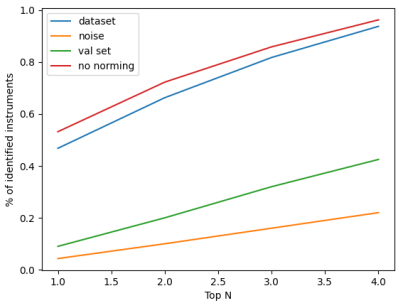
Instead of attempting to fine-tune the threshold for each class, we tried a more brute-force approach. After our final activation layer, we simply take the top N instruments, where N is a hyper-parameter.

We tried this with a few options, including the normalization of the previous section. But we found that the regular dataset, with no normalization at all gives us the best results, as can be seen in Figure 5.

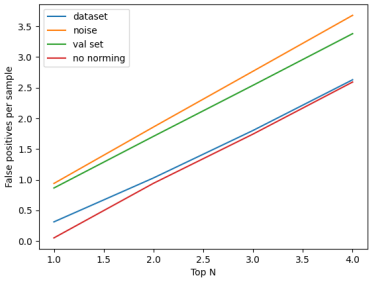
We found that taking the top 3 classes gives us the best combination of accuracy and false positives, and as such decided that this will be our choice.



(a) perfect matches



(b) accuracy



(c) false positives

Figure 5: effects of choosing the top n classes

Results

After tuning our hyper-parameters, we ran our model on our test set.

Doing so gave us an **accuracy of 76%**, meaning the model correctly identified 76% of all instruments playing. On the other hand we received, on average, **1.6 false positives**, meaning for every time frame the model added 1.6 instruments which weren't playing.

Ultimately, we are happy with the results. As mentioned, previous works in the subject identified the instrument playing with an accuracy of 80%-85%. But, as this was only one instrument, we do not think that our results are worse in any way. The false positives aren't great, but are an obvious side effect of choosing the top 3 instruments no matter what. Perhaps a mix between top-n and class-specific thresholding can give better results.

Going Forward

We feel that this project can be improved, and not just in the sense of better accuracy. If we manage to add note detection, and add that to the instrument that is being played, it would be quite easy to create a transposition of the musical piece being played.

This would simplify many jobs, where once a skilled musician had to labor to transpose a piece, having a neural network do it, and in only seconds, would be a huge improvement.

Links

- Youtube presentation
- Git repository

References

[1] https://github.com/micmarty/Instronizer

[2] https://github.com/biboamy/instrument-prediction