Recognizing predominant musical instruments in music using deep learning techniques

Martynas Vaznonis, Luke de Jeu, Felix Mozes, Michael Filip Toutonghi, Yung-Ru Woo

March 31, 2022

Abstract

Instrument classification is an exciting machine learning problem because it is challenging and can be solved in many ways. This paper focuses on recognizing 11 different predominant instruments in real-world audio. We compare the instrument classification performance of 2 models built upon deep learning techniques and the statistical approaches used in another paper. Our first implementation model uses raw data and a long short-term memory (LSTM); this model gave the same output for each input and therefore requires further adjustments for comparison. Our second implementation uses a dense neural networks with fast Fourier transformation (FFT) data pre-processing and performed approximately 43% worse than the statistical models. Our FFT model had a micro F1 score of 0.29, and the best statistical model achieved a micro F1 score of 0.503.

1 Introduction

A trained human agent can distinguish the instruments used in the creation of a piece of music with relative ease but training people for this reason is expensive and time-consuming. Nonetheless, the classification of predominant instruments would provide useful in many areas such as recommender systems and is worthwhile to automate.

In this research paper, several methods will be studied to label instruments used in a piece of music using different predictive analytic models. We developed two deep learning neural network including one with an Long Short-Term Memory (LSTM) layer. We compare the results of these neural networks with each other as well as with the statistical approaches utilized by Bosch et al. to see if the deep learning techniques improve over the performance of the statistical methods [1]. The data set containing both the training and the testing data is the IRMAS data set provided by Bosch et al. The aforementioned research paper uses The F1 score to evaluate their performance which is a harmonic mean between precision and recall. We also use this score. This way ensure comparison between the models is fair and the performance differences cannot be attributed to difference in evaluation and better or otherwise more representative data.

1.1 Hypothesis

The main focus of this paper is to apply modern deep learning techniques, such as LSTMs, to improve on the results of [1]. As the results from of that research paper were acquired by using older statistical methods and, by current standards, an underwhelming amount of computing power, the following hypothesis was formed:
By using modern methods of machine learning, higher F1 score can be achieved in recognizing predominant instruments than that accomplished by Bosch et al. (2012).

2 Literature Review

2.1 Bosch et al.

The primary benchmark we compare our results to is the aforementioned research paper by Bosch et al. [1], in which the authors attempt to classify the instruments played in polytimbral audio. Their primary research objective was to see how different pre-processing and classification methods combined would affect instrument recognition accuracy, compared to previous attempts. The main difference between Bosch et al. and other examples, and the reason we have chosen them as our benchmark compared to similar research like Heittola et al. [2] or Fuhrmann et al. [3] is their

use of a real world audio data set. As opposed to computer generated audio, or audio comprised of multiple solo instruments artificially joined, real world audio combines harmonically related instruments with effects such as reverb, delays, and noise that make instrument separation and/or identification much more difficult. Furthermore, training an accurate model on real world audio will yield a more relevant result for further application and research.

2.1.1 Models

The core model used as a baseline by Bosch et al. is a support vector machine (SVM) based system that, given an input sound, outputs a probability for each instrument label (estimating the probability that the specified label is being played in the sound). The highest probability labels are then assumed to be predominant. The labels studied in this baseline model, and therefore all models compared to it as well are cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin, and human voice. Each model developed and tested in Bosch et al. is referred to as an experiment.

This baseline is the first experimental model they evaluate. The only pre-processing that this initial model performs on its input audio is a translation from stereo audio to mono. This yields a micro F1 score of 0.378.

The second model they create involves separating the stereo input audio into four different streams, running the original model on each of those streams separately, and aggregating the resulting labels for each stream into a set of output labels. This yields a maximum micro F1 score of 0.451.

The third model separates the input audio into bass, drums, melody and other streams using a 2012 version of The Flexible Audio Source Separation Toolbox [4], and runs the original model on each of those streams. Overall, the micro F1 score for this model is 0.355 due to low precision.

The fourth model is the first to introduce new instrument recognition sub-models to improve overall classification. It consists of using the FASST method to separate the input audio, similarly to the third model, but instead of running the original SVM classifier on each stream, it uses new stream-specific SVM classifiers. This involves creating a unique model for each data stream, but yields an improved maximum overall micro F1 score of 0.446.

Finally, the fifth model mainly concerns optimizing the fourth model through introducing new data streams consisting of combinations of other individual data streams. The maximally optimizing micro F1 score this model yields is 0.503.

2.1.2 Results

Overall, Bosch et al. find that their best model (the fifth model) yields a 32% improvement over the best results achieved in their benchmark. The micro F1 score of this model is therefore our benchmark.

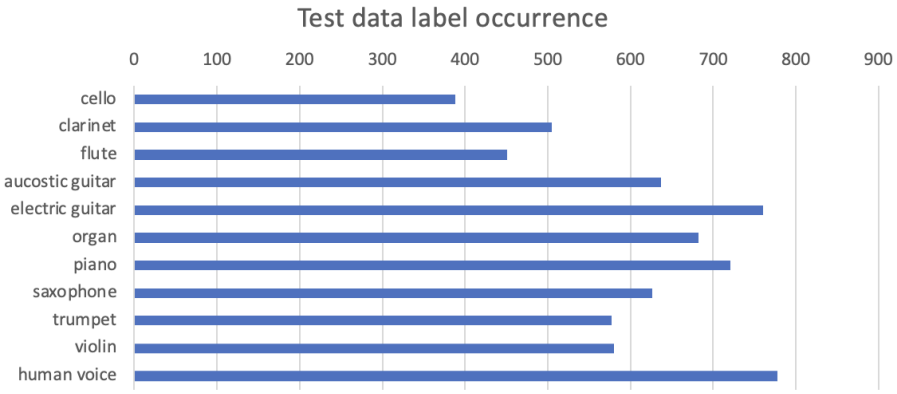
3 Method

3.1 Data Analysis

3.1.1 Data set

The IMRAS Training set is used to train and validate the models. This data set contains 6705 audio files in 16-bit stereo wav format which are all 3 seconds long sampled at 44100Hz. These audio files are fragments form 2000 different recordings. All audio files are already filed by instrument type in their respective folders. The following 11 types of instruments are present in this set: cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin and human singing voice. *Figure 1* shows the occurrence per instrument type in the training data set. The test set consist of 2874 audio files in 16-bit stereo wav format sampled at 44100Hz and are fragments of length 5 to 20 seconds. In contrast to the training set these files can contain multiple predominant instruments and therefore multiple instrument labels. The labels for these audio fragments are found in individual text files with the same name as the snippet.

Figure 1: The occurrences of the labels of the training data



3.2 Data preparation

3.2.1 Padding

The audio fragments in the test database do not all have the same duration. To ensure equal length, padding was added to all audio fragments that did not have the desired duration. This resulted in all testing audio fragments having equal length. The padding added is zeros which is equivalent to silence and it is added before and/or after the true audio fragment. The location of the additional time of silence would be randomly assigned. This mostly results in having the true audio fragment in between unequal episodes of silence. The reason the location of the added silence is randomized is to not directly increase the importance of certain time points of the data, and to force the AI to recognise silence and to understand to ignore this. To be able to learn about this silence some padding was also added to the training data despite it already being of equal length.
The added padding is not necessary when using an LSTM layer as it accepts variable-length inputs but nonetheless, adding the padding allowed us to explore different model configurations and provided freedom as a trade off to computation speed.

3.2.2 Multi-label binarizer

The multi-label binarizer uses one-hot encoding to convert categorical data into numerical data. It converts each category into a value of either 1 or 0 much like regular one-hot encoding but it also supports multi-label encoding such that more than one value can be a 1. This is necessary as the Neural Networks makes numerical computations, and outputs a numerical value as a result of this. Having the output category values be numerical as well allows for the connection of output to category. This is especially useful with the sigmoid activation, which is the function we use to produce final outputs.

3.2.3 Fast Fourier Transformation (FFT)

A Fourier Transformation is a mathematical transformation that is able to split a waveform into its individual sine functions or, in other words, its frequency domain [5]. The fast Fourier transformation is an algorithm which computes this frequency domain in $O(n \log n)$ time. We use this algorithm to convert the audio files into their compounding frequencies and use the result as the training data for one of our learners.

3.2.4 Pipeline

The training data in the IRMAS data set is 3.3GB and the testing data is 7.97GB. Loading all of this into memory at once is very demanding and slow. Instead, a pipeline was used to load the data dynamically at runtime. A python generator was set up to load batches of files at a time which is much faster to update. This in turn allowed for greater quantity of experimentation of data pre-processing and contributed to a general speed up in development.

3.3 Technologies

This section describes the technologies we use and their respective advantages and disadvantages.

3.3.1 Neural network layers

Dense layers often form the bulk of the structure within a Neural Network. Each node in a dense layer is fully connected with every node of the preceding layer. This means each node is connected to every single input node if the input layer is followed by a dense layer. Dense layers allow for the restructuring and changing the dimensionality of the data. On their own, dense layers can only learn linear decision boundaries but combined with activation functions they become much more powerful and can accomplish high accuracy on a wide range of problems.
The Flatten Layer is a useful method for changing the dimensionality of the data without applying any other calculations. It collapses the dimensions of the preceding layers and outputs the same activations but in 1-dimensional form. Flatten Layers are often applied near the end of a model, as the output is usually a 1-dimensional array of classes or outcomes.
Long Short-Term Memory (LSTM) is generally used for classification based on time series data, which is very suitable for the problem discussed in this paper. It is a specific method of deep learning which distinguishes itself from other deep learning methods by using feedback connections [6][7]. This allows for the processing of sequences as opposed to single points. However, due to the complexity of this method, LSTM layers require large amounts of computational power.

3.3.2 Loss function

As the IRMAS testing data set is multi-label, simple loss functions which map how close or how far the classification got to a value are not sufficient. Instead, the categorical cross entropy function is used which calculates the loss based on if the classifier got the answer correctly, how many answers it got right, and how close was the probability distribution to the actual answers such that the loss is smaller if it guesses 0.6 instead of 0.4 where it should have classified 1.

3.3.3 Adam Optimizer (for gradient descent)

Adam is an algorithm for stochastic optimization also used for finding weights for neural networks. Adam uses two different moment vectors for this task, m and v. The first moment vector, m, is the exponentially decaying mean gradient. This vector takes a mean of all the last weights where the latest gradient has more influence than those before. Vector m is updated using the formula $m = \beta_1 * m + (1–\beta_1) * g$, where g is the gradient, and β_1 is an input variable (usually with the value 0.9 and not often changed). The second moment vector, v, is calculated with the squared gradient. $v = \beta_2 * v + (1–\beta_2) * g^2$, where β_2 is an input variable (usually this is 0.999 and is not often changed). Finally, the gradient descent's actual step is denoted by w and calculated with the following formula $w = w - \alpha * \frac{m}{\sqrt{v}+e}$, where e is a constant with the value 10^-8 and α is the learning rate [8]. Adam is very useful because its moment allows the gradient descent to deal with noisy loss landscapes. In addition, Adam works for many machine learning applications whilst still being very computationally efficient and stable.

3.3.4 Activation functions

Rectified Linear Unit (ReLU) is a piecewise linear neuron activation function. If the input to a neuron has a positive value, the neuron will have the same value. If the input value is zero or smaller, the value of the neuron will be zero. In other words, ReLU can be expressed as a simple function $f(x) = \max(0, x)$ where x is the input value to the neuron (the sum of values of the neurons in the last layer that are connected to this specific neuron multiplied by their respective weight) [9].
Leaky ReLU. A problem that ReLU is susceptible to is dead neurons, which are nodes with value of 0 at all times. This happens if during training a neuron is assigned a very negative weight or if the weight is initialized as 0. If the input value were, for example, -1000 the neural value would become 0. Training often does not find a better weight near this very negative or zero value since the neuron in all likelihood will still produce a value of 0. This is a consequence of training methods usually looking for nearby values of weights to examine what provides better performance, but if the value is extremely negative then a slight shift towards positivity means that the node remains 0. This means that the trainer can not use any type of gradient descent to look for improvements because the loss landscape in the large negative area will be flat. Getting dead neurons severely

limits the model's capabilities because the network becomes smaller than designed. A solution to this problem is the leaky ReLU as the activation function. Instead of taking a value of 0 when the input is negative, it multiplies the negative input value with a small number between 0 and 1. [10] The formula for this is $f(x) = \max(\mu x, x)$ where x is the input just like in regular ReLU and μ is some constant smaller than 1 and larger than 0, usually around 0.01. A large negative value with leaky ReLU will affect the end result more than with ReLU as ReLU transforms it to a 0 while leaky ReLU has passes on a fraction of the negative value to the next neuron. This means that during training the learning algorithm will have to take into account that neurons with very negative values will need to be changed to get accurate results. In other words, leaky ReLU creates a non zero gradient which can be descended. [9].

Sigmoid activation. The sigmoid function maps input values to a value between 0 and 1 with an S-shaped curve and can be used as yet another activation function. The sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. For this function, if inputs are around 0, they make a large difference, but the more or less the input value changes, the less of a difference it makes towards the output.[9] e.g. the output values from inputs 10 and 11 differ less than 1 and 2. One problem that occurs with using sigmoid is that at either the far negative and positive numbers the slope becomes almost 0. This means that gradient descent or back-propagation will no longer work as effectively and the neural network will stop training. Using the sigmoid function is very popular for multi-label classification problems because if the final output value is larger than 0.5 it can be rounded up to 1 and if it is smaller then it can be rounder down to 0.

There are furthermore some activation functions used implicitly in our project. Those include the sigmoid and the tanh functions found in the LSTM layer. The tanh is similar to the sigmoid function but it maps every input between -1 and 1 instead of between 0 and 1.

4 Models

4.1 Raw Data Based LSTM network

4.1.1 Motivation

As mentioned in 3.3.1, LSTM has the unique capability of being able to process sequences of data. This contrasts with most other techniques, where only data of fixed length can be processed and the temporal ordering is hard to express. The two main advantages of the technique used by an LSTM for instrument recognition are learning from patterns in sequences and understanding temporal data. By looking at the pattern of sequences, the hypothesis is that the LSTM model can learn to distinguish between more continuous sounds (e.g. violin) and more sporadic sounds (e.g. piano). As well as that, the LSTM model will be able to learn from sound sequence patterns like repeating chords or frequented notes that may follow each other. Additionally, speed and tempo can be extracted more effectively. If the LSTM layer can extract this data, it is not unreasonable that the model would learn to classify instruments based on it.

4.1.2 Architecture Overview

The input to the LSTM model consists of an input layer, followed by an unaltered LSTM layer. The input layer feeds the LSTM layer audio fragment of varying duration. Where other neural network types require the same number of data points to make a valuable comparison, an LSTM does not. Therefore, the discrepancy in size between the training data and testing data has no negative impact. At each step, the input layer feeds the consequent LSTM layer a single data point, the value of the sound wave at that point in time. After the LSTM layer, the data passes through three separate dense layers with leaky ReLU activation. These layers are meant to process the output of the LSTM layer and to separate the information which the LSTM was able to extract, into distinct classes. The last of these three dense layers feeds the output to a final dense output layer. This output layer uses the sigmoid activation to allow for multi-label classification. The output layer has an output size of 11, corresponding to the 11 possible instruments. The size of the first dimension of each of these layers is dependent on the batch size. This is a hyperparameter which can be set and adjusted before creating the model. The batch size must be equal for each layer. The batch size can be altered to change processing efficiency.

This is an overview of the layers and sizes, in order, of this model:

- Input with shape equal to (Batch size, None, 1)
- LSTM with output shape (Batch size, 22)

- Dense layer with shape (Batch size, 2048) and leaky ReLU activation
- Dense layer with shape (Batch size, 32) and leaky ReLU activation
- Dense layer with shape (Batch size, 1024) and leaky ReLU activation
- Dense output layer with shape (Batch size, 11) and sigmoid activation

4.1.3 Issues

Due to the very long back-propagation in the LSTM layer, it will inherently require significantly more processing power than other layer types. While LSTM is likely very suitable for instrument classification, it may be impossible to tune it appropriately because of insufficient computing power. As our research is very limited in time and processing resources, this method may very well not lead to groundbreaking results despite its apparent fitness.

4.2 FFT Based network

The fast Fourier transform is an efficient algorithm for calculating the discrete Fourier transform of a given input signal. Raw signal data is taken as input, and a frequency domain is produced as output (3.2.3). In the context of machine learning, the FFT is a data pre-processing method that allows a model to train on processed frequency signatures, as opposed to noisy raw audio data.

4.2.1 Motivation

Pre-processing our raw audio sample data into the frequency domain facilitates easier computation of correlations between frequency signatures, and is a relatively time-efficient operation. In addition to this, instruments often have certain repeated frequency patterns, which are easier to identify in the frequency domain than in raw data. By using the FFT as a data pre-processing method we make the assumption that a mathematical frequency spectrum of an audio sample carries more relevancy with regards to instrument recognition than the raw audio sample itself, and therefore, the transformation improves the signal to noise ratio.

4.2.2 Architecture Overview

The general architecture of our FFT based model is neural network consisting of four dense hidden layers, with a sigmoid activation function on the output layer, and the FFT derived frequency domain as input data. The input shape of this model is made up of the batch size hyperparameter and 442000 features (the total number of points on each frequency domain, after padding is applied to the input sample to be over 20 seconds long, divided by two, taking advantage of FFT symmetry). All hidden layers use the leaky ReLU activation function and are shaped (batch size, 64), (batch size, 100000), (batch size, 32) and (batch size, 50000) respectively. The output layer again has shape (batch size, 11), one output for each instrument label.

4.2.3 Issues

Pre-processing our raw audio data into the frequency domain prunes time data. This diminishes the value of certain data-length agnostic classification methods like LSTMs, requiring the addition of padding to each piece of input data such that every input is the length of the longest excerpt. Fortunately, the frequency domain remains almost entirely unchanged with the addition of silent padding in the time domain.

4.3 Other networks

Besides the two networks described in detail above, we experimented with a handful of other options, predominantly with convolutional layers. Alas, this yielded no better results than the simpler networks so we elected to omit them from this paper. It is likely that better outcomes can be accomplished with CNNs but we had to prioritize which networks to explore more in-depth and we chose the above-mentioned options instead.

5 Results

5.1 LSTM method

5.1.1 Performance Details

As expected, the LSTM model was challenging to train and test. It required large amounts of processing power to compute, which resulted in a maximum batch size of 2 and a reduction in the number of training iterations in order to train it within the time frame of this research project. The overall results of this model were inadequate for instrument classification. The micro precision = 0.25, recall = 0.75 and F1 = 0.38. On the surface, these results may seem to indicate that the model has indeed learned something and is not just randomly guessing. Upon closer inspection though, the relatively high recall and lower precision indicates a model which prioritizes classifying more positives than negatives which leads to many false positives and few false negatives.

Further analysis of the outputs still, reveals the nature of the results. The electric guitar, organ, piano, saxophone and human voice show a recall of 1. All the other instruments show a recall of 0, or very close to 0. This indicates that it is very likely that the model had the same output for nearly all inputs. The results indicate that this output would have been [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1], where the values 1 correspond to each instrument with a recall of 1. The acoustic guitar has a precision of 0.25 and a recall of 0.04, making it the only instrument with a higher precision than recall. However, these values still do not look promising. The extremely low recall indicates that the model only was able to successfully identify a very low number of acoustic guitars, and the relatively high precision then indicates that this successful classification was one of the few positive classifications of this instrument.

5.1.2 Hyperparameters

During the testing phase, it became obvious some hyperparameters must be reduced due to hardware limitations. However, it is crucial to try to reduce as little as possible and to have the results be minimally affected. Therefore, the number of epochs was greatly reduced. While training, the most significant changes seemed to happen in within the first 5 epochs. After these, the improvements became extremely marginal. Because of this, the total number of epochs was set to 15. The training and testing validation were nearly identical and did not show any signs of overfitting. This may also be a sign that the model was hardly learning anything. The model was trained on a batch size of 2, but as the change in batch size from 1 to 2 did not seem to have any significant effect on model performance. A larger batch size was not possible due to memory limitations. The size of the LSTM layer was set to 22. This was the least preferred hyperparameter to limit, as it was deemed to be the most likely to cripple the performance. However, reducing the size of the LSTM layer also have had the most significant effect on run time. Therefore, necessary reductions were made. Finally, the dense layers were structured to have a small dense layer in between large dense layers. Due to how dense layers function, being connected from each node in the previous layer to each node from the next layer, having a large dense layer connected to another dense layer in sequence would have astronomically increased the number of trainable parameters/connections. Therefore, in order to still maintain at least two large dense layers, a smaller dense layer was necessary in between these two larger layers, to maintain a manageable number of parameters.

5.2 FFT method

5.2.1 Performance Details

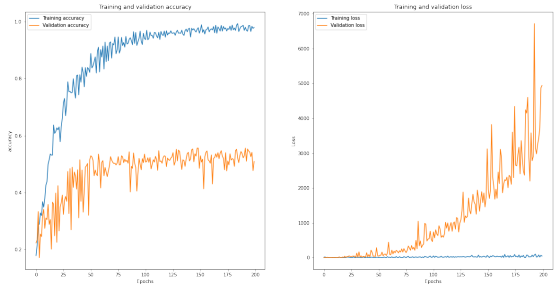
Overall, using FFT pre-processing combined with a densely layered network has been our highest performing method, with a micro F1 = 0.29. Notably, it has a very high micro average recall of 0.68 across all instruments, and thus performs better in this aspect than any model tested in Bosch et al. Unfortunately, its overall precision is very low, at a micro average of 0.18, bringing down the F1 score to below that of the worst performing models in the original paper.

If we analyse the performance of this model per instrument, we find that, inline with SVM models explored in prior research, voice is the easiest to recognise correctly, with a precision of 0.48, recall of 0.80 and F1 score of 0.60. Also consistent with prior models is the fact that clarinet is the most challenging, with precision of 0.02, recall of 0.74 and an F1 score of 0.05. This is likely due to how voice and clarinet are represented in the training data, and could benefit from further investigation on different data-sets.

5.2.2 Hyperparameters

As shown in the figure 2, both training and validation accuracy begin to plateau after around 75 epochs, whereas validation loss continues exponentially diverge as epochs increase, potentially indicating overfitting with a large number of epochs. On the other hand, when trained on only 75 epochs, the results of classification on the testing data were worse (micro F1 = 0.16). Therefore, the diverging loss may not be exactly indicative of overfitting.

Figure 2: The effect of increasing epochs on accuracy and loss



For batch size, we find that 4 yields good results before returns become diminishing, and memory costs outweigh increases in accuracy. The number of neurons in the dense layers, and the number of dense layers (as described in 4.2.2) were found through a process of trial and error, though further experimentation would likely lead to beneficial results.

For learning rate, we find that large values cause the network to unlearn potentially useful information when it encounters outliers. Due to this, we chose a relatively small value of 0.0001, mostly through trial and error.

6 Conclusion

The hypothesis stated the following: *By using modern methods of machine learning, higher F1 score can be achieved in recognizing predominant instruments than that accomplished by Bosch et al. (2012).* The values from the results prove this hypotheses to be false. The worst performing model in [1] has a micro F1 of 0.378, and their best performing model has a micro F1 of 0.503. The models in this research produced a micro F1 of 0.29 and 0.38 for the FFT and LSTM model respectively.

While the LSTM model seems to perform as well or even slightly better than the worst model in [1], this is actually not the case. The F1 value of the LSTM is not representative to the classification capabilities of of the model. The results show that the model has the same output for almost all instances, regardless of the input. The instruments which are almost always classified as true are the electric guitar, organ, piano, saxophone and human voice. As seen in *Figure 1*, these instruments make up 5 out of the 6 most occurring instruments. This makes it clear that the LSTM model did not just under perform, but it did not perform as a dynamic classification model at all. By always classifying the most common instances of the database as true, LSTM model found a method to have the largest probability to be right, without having to recognise instruments. Therefore, the F1 value does not hold any meaning, as it was achieved without making any attempt to recognise instruments.

Although the FFT model's results are more varied, and therefore more indicative of an instrument recognition system than the LSTM model, it still underperforms our baseline. The FFT model was shown to have a better micro average recall than any model in our baseline, yet a much lower micro average precision, and therefore overall a lower performing micro average F1 score. This implies that the likelihood that the model will mark a given instrument as predominant for a given input is too high, and it should be less lenient when making classifications. Fortunately, this provides a promising avenue to continue exploring in search of F1 score improvement, such as increasing model complexity, and adding more advanced pre-processing methods.

In summary, our best model (in terms of both performance and the ability to represent instrument recognition) underperforms our baseline by 42%, and further underperforms the worst model tested by Bosch et al. by 23%.

7 Discussion

There are three potential reasons why the LSTM did not just underperform, but misperformed as well. The first reason could be a wrong model structure. The model structure could be altered by chaining the number of layers, the order of the layers, or the layer types. Each of these changes to the model structure must create a significant change, in order to change the way the LSTM model classifies. The second reason is that the model requires significantly larger layer sizes. Chaining the model size only slightly will not change how the model classifies, but will only slightly change the efficiency. However, greatly increasing the layer size will possibly allow for completely new complex network structures, which can make proper classifications. The third reason, is that the model was potentially trained on a much too small of a data set. A larger data set allows for different methods of data prepossessing and restructuring. This could eliminate class imbalances and make outputting the the same classes regardless of the input much more expensive.

Despite not surpassing the methods employed by Bosch et al., it is very likely that instruments can be classified using deep learning techniques. The FFT based network showed clear signs of improvement and further model exploration and additional pre-processing techniques could lead to vastly greater performance.

One obvious detriment to the development of the models and subsequently their performance is the lack of time and computational power. Improvements could have been made and a higher expected performance achieved if the training time and computational power were not as constricted. This would have allowed for greater hyperparameter selection, model exploration, and data manipulation. One data pre-processing method which may lead to better results is separating the data into different streams using the FASST method like done by Bosch et al., potentially then applying the FFT to each stream as well. Another method worth considering of implementation is conversion of data to spectrograms and training an image classification model to translate them into instrument labels. This could work in conjunction with the other models and the final set of labels could be taken by combining the predicted results.

References

[1] J. J. Bosch, "A comparison of sound segregation techniques ... - ismir 2012," 2012. [Online]. Available: https://ismir2012.ismir.net/event/papers/559_ISMIR_2012.pdf

[2] T. Heittola, A. Klapuri, and T. Virtanen, "Musical instrument recognition in polyphonic audio using source-filter model for sound separation," 2009. [Online]. Available: https://www.researchgate.net/publication/220723588_Musical_Instrument_Recognition_in_Polyphonic_Audio_Using_Source-Filter_Model_for_Sound_Separation

[3] F. Fuhrmann, M. Haro, and P. Herrera, "Scalability, generality and temporal aspects in automatic recognition of predominant musical instruments in polyphonic music." 2009. [Online]. Available: https://www.researchgate.net/publication/220723441_Scalability_Generality_and_Temporal_Aspects_in_Automatic_Recognition_of_Predominant_Musical_Instruments_in_Polyphonic_Music

[4] Y. Salaün, E. Vincent, N. Bertin, N. Souviraà-Labastie, X. Jaureguiberry, D. Tran, and F. Bimbot, "The flexible audio source separation toolbox version 2.0," 2014. [Online]. Available: https://hal.inria.fr/hal-00957412/document

[5] M. Heideman, D. Johnson, and C. Burrus, "Gauss and the history of the fast fourier transform," *Archive for History of Exact Sciences*, vol. 34, pp. 265–277, 01 1985. doi: 10.1007/BF00348431. [Online]. Available: https://www.cis.rit.edu/class/simg716/Gauss_History_FFT.pdf

[6] M. Liwicki, A. Graves, H. Bunke, and J. Schmidhuber, "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks," *ICDAR*, 2007. doi: 10.1.1.139.5852. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=4261F9335B602D2931B907FD5DAEC261?doi=10.1.1.139.5852&rep=rep1&type=pdf

[7] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition," 2014. doi: 10.48550/ARXIV.1410.4281. [Online]. Available: https://arxiv.org/abs/1410.4281

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *3rd International Conference for Learning Representations*, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[9] J. Feng and S. Lu, "Performance analysis of various activation functions in artificial neural networks," 2019. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1237/2/022030/meta

[10] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1. Citeseer, 2013, p. 3. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.693.1422&rep=rep1&type=pdf