DEEP LEARNING FOR AUDIO

Evan Cameron
University of Victoria
evancam@uvic.ca

ABSTRACT

The purpose of this paper is to test and analyze different machine learning approaches to perform automatic recognition of musical instruments in music. The majority of the analysis effort will be primarily focused on the pre-processing steps of the audio signal, specifically the Short Time Fourier Transform step. There have been many variations of convolutional neural networks used in Music Information Retrieval, and the majority of these follow similar pre-processing steps of obtaining the Mel-STFT. During the transformation of raw audio to a STFT representation, there will be multiple different windowing functions tested. This will be tested on inputs of monophonic, polyphonic and polyphonic MIDI audio. The results will then be evaluated on time required for pre-processing, and the error rate and training speed of the Xception architecture.

1. INTRODUCTION

Music information retrieval (MIR) has been a rapidly advancing field in recent years, particularly with recent developments in deep learning. The MIR field focuses on simply retrieving information from music. This can include track separation, automatic transcription, music generation, various categorization techniques and recommendation systems. There have been efforts to advance the ability to use machine learning to recognize instruments.

Classically, instrument recognition was done using timber analysis using manual crafting of features. These features include MPEG-7 specification based descriptors [20], such as spectrum centroid, spread, flatness, harmonic basis functions, log attack time, fundamental frequency and many others. These techniques make use of both the frequency domain and time domains. Timbre analysis does not necessarily involve identifying the instruments associated with the sound, but merely the qualities of the audio being played.

The goal of using deep learning for MIR is to allow for the deep learning model to learn features on its own. They may overlap with manually crafted features, but we expect the deep learning model to make its own correlations and connections. Current deep learning techniques for MIR leverage currently existing models for image recognition for use in analysis of audio.

© Evan Cameron. Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution:** Evan Cameron. "DEEP LEARNING FOR AUDIO",

2. BACKGROUND

2.1 Deep Learning Networks

Convolutional Neural Networks (CNN's) are currently a very popular deep learning network for MIR. This is due to its success in image and speech recognition, as neural nets are intentionally based on the concepts of neurons, a concept rooted in biology [12]. CNN's still have limitations, as due to the pooling layer CNN's can only evaluate time-invariant features. The pooling layer could be removed, but the ramifications of that would include a drastic increase in the computation time. The advantages of CNN's are mainly for image and video recognition tasks, as the convolutions are good at evaluating local features, such as the edges of objects, or the boundaries between 2 objects.

There are several popular CNN architectures, including AlexNet, VGG, ResNet and Inception. These networks are quite deep, and contain many convolution and pooling layers. The depth of these networks is only possible due to advancements which allow much faster model training on GPU's. Even still, training the network can still take multiple days depending on dataset size. These CNN architectures have been primarily built for image recognition tasks. They can be translated to use for MIR tasks, as Frequency-based representations of audio files can be represented as images.

2.2 Data Representation

There are several different ways of representing data before processing it. In some cases, simply the raw audio data is fed directly into the neural network. The majority of modern MIR analysis is done on data that has been processed in some fashion. Data pre-processing typically happens in MIR to transform the raw audio signal to some sort of frequency-based representation using the Fast Fourier Transform (FFT). FFT is a method of obtaining the Discrete Fourier Transform, typically through divide and conquer methods. FFT results in a time complexity of $\mathcal{O}(n \log n)$ versus a standard DFT of $\mathcal{O}(n^2)$.The data is then further transformed further using the the log representation of Short-Time Fourier Transform (STFT) which is the transformation of the FFT into a 2-D representation based on the magnitude of the FFT. The Mel-STFT then optimizes the STFT even further by compressing the frequency axis.

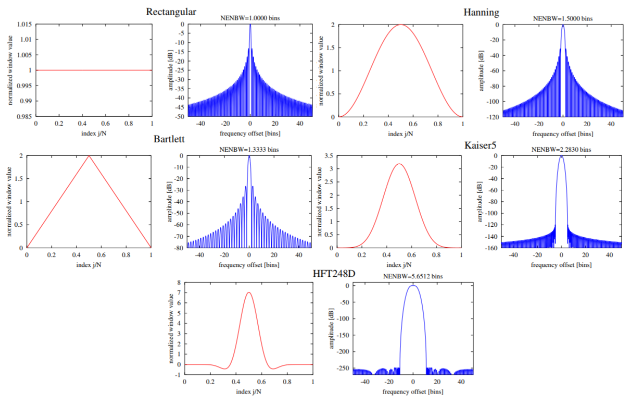

Figure 1. Windowing Functions from Top Left to Bottom: Rectangular, Hanning, Bartlett, Kaiser5, HFT248D [7]

2.3 Windowing

Windowing plays a key role in the pre-processing of data. The process of windowing is what transforms the raw audio data into a form that an FFT can be performed on it. When an FFT converts an audio file to its frequency representation, spectral leakage is added, which creates noise around the the frequencies found in the music, thus making it not a perfect representation of the frequency response of the audio file. A winwoding function can reduce the spectral leakage by using an function to approximate the areas sensitive to spectral leakage and reduce their amplitude. The windowing function itself is often overlooked, as it can significantly affect the STFT. The popularly used Hann window is effective for most cases, but still has spectral leakage below -42.7 dB, and a 3 dB width of 1.3008 bins [7]. There are windowing functions similar to the Hann window with less spectral leakage, such as the Kaiser or Blackman windows. For a smaller 3 dB width, there are the Bartlett and square windows, which allow for a smaller bandwidth at the cost of more spectral leakage. However, perhaps noise will add randomness to a data sample to avoid overfitting, so a simple square windowing function may yield beneficial results as well. The windowing functions can be seen in more detail in Figure 1.

The window size is important to the process as well. For note prediction, it has been shown that larger window sizes, such as 16,384 samples are effective in MIR in some cases [19], however for instrument recognition, a shorter window size is generally better [6]. With the window size, the overlap plays a role as well. Different windowing functions will have varying overlap percentages depending on the shape of the function, as some functions may have a more impulse-like shape which results in a smaller overlap to properly account for each sample of the signal equally.

2.4 Audio Data sources

For recognition of instruments in audio, several types of sources can be analyzed. Monophonic data contains only a single audio source from a single instrument. The RWC Music Database [5] contains monophonic data for instruments, with a wide variety of sources and instrument families.

A more complicated MIR problem is handling polyphonic data. Polyphonic data is much harder to analyze than monophonic data, due to multiple instruments providing both noise from a single source, as well as making it more difficult to properly classify. The MusicNet database [19], provides 330 classical music recordings with labels regarding which instrument plays each individual note.

Another option is Musical Instrument Digital Interface (MIDI) data [14]. MIDI data is more consistent, as it is not a recording of an instrument, but a combination of event messages that specify notation, pitch, velocity, and many other features. MIDI allows for a wide variety of electronic musical instruments. The electronic nature of MIDI allows for a data set to be algorithmically generated and tagged. This gives us the benefit of potentially comparing the recorded polyphonic dataset to an electronically generated one. However, if we are not given the information required for a MIDI file from the dataset, this may become a time-consuming process. Algorithmically generating polyphonic MIDI files allows for cases beyond standard musical composition, including music clips with many more instruments, or an ample number of the same instrument.

3. SYSTEM STRUCTURE

Selecting a system properly tailored to the problem is crucial to its success. Below we propose the optimal structure of frameworks, pre-processing and model and activation function. This structure will have to be flexible enough to handle the multiple different structures of data given to it, as the output classifier will have to be able to differentiate between data with a both single and multiple instrument sources.

3.1 Framework

For creating and training the models, TensowFlow [1] will be used. Tensorflow is chosen due to its ease of building a network model using Python, while still having efficient underlying c++ code. Keras [3] will be used to simplify the creation of the model even further. Keras simplifies the development of machine learning networks by making more readable code, as well as reducing the number of lines required to produce models. It also allows the import of popular pre-trained classification models, but these are trained on the ImageNet, so they will not be useful for this application.

3.2 Pre-processing

The majority of the experiments will be related to the pre-processing step. The performance of the deep learning model on the various types of input data with their respective windowing functions will be key components evaluated in the experiments. The Hann window will be used as a baseline for the system, as the majority of systems have previously used this. The Bartlett window will be used to evaluate the performance of the system when there is a small 3 dB width, with a high sidelobe level of -25 dB.

The Kaiser5 window will be used to evaluate a reasonable improvement over the baseline, with a 3 dB width of 2.15 bins, and a sidelobe level of -119.8 dB. To exaggerate the effect even more, the HFT248D window will be used to highlight the effects of a lowered sidelobe level of -248.4 dB, with a larger central bandwidth of 5.5567 bins. The square wave will be used to observe how the system responds under high noise, as the sidlobe level is very high at -13dB. The main goal behind the selection of the windowing functions is to give sufficient variety in their structure to present a sufficiently broad range of available options of windowing functions.

From [6], a window size of 1024 frames will be used, with varying hop sizes depending on the windowing function being used. See Table 1 for the windowing functions and their respective hop sizes.

For input size, the popular duration for many databases is 30 second audio clips, so the Mel-STFT will be normalized around this duration. The data sets could be downsampled with respect to the highest frequencies used by the instruments in the dataset, but the processing involved to determine this would not be worth it when comparing to sampling at the standard rate of 44100 Hz.

3.3 Deep Learning Model

The ConvNets [17] have been used very effectively in large scale image recognition. Since the data is in essence being represented as a 2-D image, we can effectively use networks that built for that purpose. It has been shown that a slightly modified Inception V3 had the lowest error rate when compared to other popular CNN architectures [8]. Xception, a modification of the Inception architecture, was tested on ImageNet, a standard image recognition database and was found to have a lower error rate than Inception V3 [4]. Xception will be used as the deep learning model, due to its low error rate and ease of integration with Keras as there is already an Xception model built into the framework.

Dropout will be used in the deep learning model. Dropout provides a simple way to avoid overfitting by randomly dropping units and connections from the network during training [18]. Dropout has shown success in CNN architectures over conventional methods [16]. In our architecture, we will use dropout after the fully connected layer at a rate of .5 due to its tendency of overfitting [6].

3.4 Activation Function

Since the 2010's ReLU has been used in virtually all deep learning tasks. For polyphonic instrument recognition in deep convolutional networks, it has been found that very leaky ReLU (LReLU) is more effective than standard ReLU or normal leaky ReLU [16].

$$y_i = \begin{cases} z_i & z_i \geq 0 \\ \alpha_i z_i & z_i < 0 \end{cases} \tag{1}$$

The activation function used will be standard ReLU. Even though LReLU performed better in previous instrument recognition networks, it was specifically trained on polyphonic music. This may be a problem when testing on monophonic music, or computer generated polyphonic music. To ensure the activation function does not create an undesired bias in the system, standard ReLU will used.

	Overlap%	Hop (Frames)	Preproc. (s)
Hann	50	512	1173.02
Bartlett	50	512	1171.71
Square	50	512	1152.26
Kaiser5	70.5	302	1291.17
HFT248D	84.1	163	1532.32

Table 1. Hop Sizes of Windowing Functions by Percentage and Frames, and Pre-processing time

4. EXPERIMENTS

4.1 Setup and Data

The experiments will be performed on a Nvidia GTX 1080 with 8 GB of memory and 2560 Cuda cores which will be primarily used for training and testing the model. The CPU is an Intel 6700K, which will be used for doing the preprocessing of the audio samples. The tests were performed on a Windows 10 machine running Python 3.6. In each experiment, a different windowing function will be tested on the various types of data sets. The windowing functions tested in each experiment will be the Hann window, Square Window, Bartlet Window, Kaiser Window and HFT248D window. Since the Hann window is used most frequently, it will be used as the baseline. The preprocessing speed of each of the windowing functions will be evaluated, as well as the training speed and accuracy of the network itself.

For data, the IRMAS [2] dataset was used. It contains 6705 samples from 2000 unique pieces of music at a duration of 3 seconds long. The clips are sampled at 44100 Hz, and span various decades and genres. The instruments that will be classified from the samples are the cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin, and human singing voice.

4.2 Model and data generation

For the generation of data for input into the modified Xception model, the images after being preprocessed were put into individual pickle files for loading. Solely the IRMAS training data was used for both training and validation due to the ease of partitioning the data. The testing data provided in the IRMAS set also included multi-label data which was undesired for validation.

The Adam optimizer was used with the default Keras parameters (Learning rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0) [10]. The reasoning for choosing it was the computational efficiency and low memory requirements. The Adam classifier also highlights that is useful for noisy applications which is appealing, as beyond the noise created by the spectral leakage from the windowing functions, there is noise in the dataset itself as the samples are not purely clean samples of the individual instruments.

For the weight of the model, it was initialized to random weights. This was partially due to the results found from [15]. Another aspect of this decision was due to attempted training with 'Imagenet' weights, which are for a large image database, provided for Xception from Keras. The initial results of this attempted training were significantly lower classification rates with the initial few epochs when both allowing and disallowing training on the inner layers of the model. It may have been the case where with enough training the random and 'Imagenet' weights may have converged to similar results, but this was not investigated.

Based on the size of the batch sizes used in [13], a batch size of 21 will be optimal for training when calculated from the difference in hardware specifications and image sizes. In reality when this batch size was attempted, only 6 GB of memory could be allocated from the graphics card, which resulted in memory errors and TensorFlow crashing. A batch size of 12 was used instead, which was discovered through trial and error, and even then memory warnings still occurred, but were not critical enough to stop the training.

In typical image classification, data augmentation will occur to increase the number of samples available without requiring more initial data. This is not particularly useful in this case as most of the standard augmentations involve using skew or linear transforms, which is not useful for out input. Our Input is standardized as a square image with variable patterns based on the various preprocessing windowing functions. There may be a way to augment data but not through traditional image processing methods.

4.3 Preprocessing Results

The preprocessing results for the time can be seen in Table 1. The column of the table labeled Prepoc. (s) is the amount of time taken for the preprocessing algorithm to run on all the samples. For the windowing function, the numpy and scipy [9] libraries were used. From the results, we can see there is significant variation in time required to complete preprocessing. The square wave took the shortest amount of time to complete preprocessing which makes sense, as this is essentially performing no windowing function upon the bins, resulting in no change to the amplitude. All other windowing functions require vector multiplication of each bin multiplied by the discrete amplitude of the windowing function for each frame.

Overall we can see a trend that as the overlap percentage increases, the amount of time required to complete preprocessing increases. The Bartlett and Hann windows took a very similar amount of time to complete, within 0.17% of each other, both with 50% overlap. What is interesting is the amount of time the Kaiser5 and HFT248D windows took to complete, as they appear to be the two anomalies. In theory the HFT248D has a higher overlap percentage than the Kaiser5 function, but the difference could be due to how each function is handled by the underlying libraries,

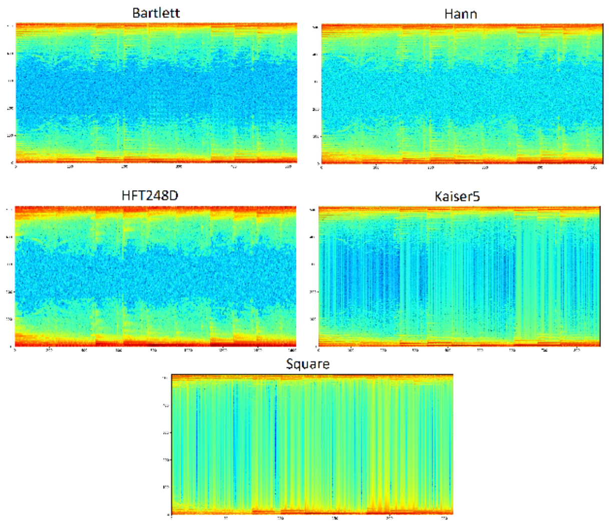

Figure 2. Sample output from preprocessing step, from Top Left to Bottom: Bartlett, Hann, HFT248D, Kaiser5, Square

	Accuracy (%)	Avg Epoch time (s)
Hann	70.83	232.9
Bartlett	74.17	240
Square	61.67	239.7
Kaiser5	72.5	227.7
HFT248D	70.83	249.8

Table 2. Best Accuracy and Average time per Epoch of various windowing functions

as the HFT248D made use of scipy's windowing functions while all other windowing functions used the numpy library.

From the appearance of the preprocessed samples obtained, the Bartlett, Hann and HFT248D windows have similar appearance in terms of noise values, with variation in pixel intensity. The Kaiser5 window appears to have a significant amount of noise on the middle of the stft image, which is surprising as the kaiser window with an alpha value of five should be similar in structure to the Hann window. As expected, the square window has a very significant amount of noise in the image.

4.4 Training Results

The results from the training can be found in Table 2. We can see that the Bartlett window achieved the best results, at 74.17% classification rate. This seemed rather unexpected, as the Bartlett window was rather neutral in terms of windows, without having very low or high noise values, or the smallest 3dB width. The Hann window, which was used somewhat as a baseline was found to be tied for third the best windowing function with the HFT248D window at a classification rate of 70.83%. As was suspected, the square window performed the wort at a classification rate of 61.67%, likely due to the high noise of the processed images being difficult to classify.

To contextualize these results, Han et al. achieved an F1 classification of 59.1% [6], but this was using the IRMAS polyphonic test data and classifying to multiple labels. Using the RWC database and classification of 12 individual instruments, Krishna and Sreenvias used Gaussian Mixture Models and maximum-likelihood classification to obtain an accuracy of 84% on solo phrases [11]. It should be noted that the data used in IRMAS is based on predominant instruments for classification, but may still contain other instruments or percussion.

As expected, the average epoch time is roughly equivalent since all data was run through the same model steps. After each windowing function had completed the 30 epochs, it did not appear that any of them had converged to optimal results, with the exception of possibly the square function, as it had been showing worsening results for a number of epochs.

5. CONCLUSION AND FUTURE WORK

Based on the results of the best accuracies in Table 2, the popular Hann window may not be the best option for classification of musical instruments. Based on the results, it was found that The Bartlett window was the best at 3.34% higher accuracy, and the Kaiser5 window had a 1.67% better classification rate over the Hann window. There are some limitations to these results, as the validation was only run on 120 samples, which results in a reasonable amount of uncertainty.

One obvious improvement would be to make proper use of the IRMAS test data. Since the test data was used for both training and validation, similar samples may have been used in such a way to have skewed results from overfitting. The model would have to change to account for multi-label testing data, though, or the data should be sanitized to only include single-label data. The number of validation samples should be increased as well to yield more accurate results. Beyond that, it would be good to evaluate the performance of the setup on different datasets as well.

There is likely much potential for further investigation of the parameters used in preprocessing. One key change would be the use of MFCCs with a properly chosen constant instead of just log scaled STFTs. Parameters such as window size and sampling frequency could be investigated as the different windowing functions may perform better with various tweaks to the parameters.

6. REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association.

[2] Juan J. Bosch, Ferdinand Fuhrmann, and Perfecto Herrera. IRMAS: a dataset for instrument recognition in musical audio signals, September 2014.

[3] François Chollet et al. Keras. `https://keras.io`, 2015.

[4] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[5] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. Rwc music database: Popular, classical, and jazz music databases. In *In Proc. 3rd International Conference on Music Information Retrieval*, pages 287–288, 2002.

[6] Yoonchang Han, Jae-Hun Kim, and Kyogu Lee. Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *CoRR*, abs/1605.09507, 2016.

[7] Roland Heinzel, Gerhard; Rdiger Albrecht; Schilling. Spectrum and spectral density estimation by the discrete fourier transform (dft), including a comprehensive list of window functions and some new at-top windows. *Max Planck Society*, 2002.

[8] Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, R. Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron J. Weiss, and Kevin W. Wilson. CNN architectures for large-scale audio classification. *CoRR*, abs/1609.09430, 2016.

[9] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed Oct 20th].

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[11] A. G. Krishna and T. V. Sreenivas. Music instrument recognition: from isolated notes to solo phrases. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages iv–iv, May 2004.

[12] Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[13] Ardi Loot. Training xception model for kaggle competition cdiscount s image classification challenge . 2018.

[14] Robert A. Moog. Midi: Musical instrument digital interface. *J. Audio Eng. Soc*, 34(5):394–404, 1986.

[15] Jordi Pons and Xavier Serra. Randomly weighted cnns for (music) audio classification. *CoRR*, abs/1805.00237, 2018.

[16] S. Sigtia and S. Dixon. Improved music feature learning with deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6959–6963, 2014.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[19] John Thickstun, Zaid Harchaoui, and Sham M. Kakade. Learning features of music from scratch. In *International Conference on Learning Representations (ICLR)*, 2017.

[20] X. Zhang and W. R. Zbigniew. Analysis of sound features for music timbre recognition. pages 3–8, April 2007.