12-2009

An Exploration of Monophonic Instrument Classification Using Multi-Threaded Artificial Neural Networks

Marc Joseph Rubin
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a thesis written by Marc Joseph Rubin entitled "An Exploration of Monophonic Instrument Classification Using Multi-Threaded Artificial Neural Networks." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jens Gregor, Major Professor

We have read this thesis and recommend its acceptance:

James Plank, Bruce MacLennan

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Marc Joseph Rubin entitled "An Exploration of Monophonic Instrument Classification Using Multi-Threaded Artificial Neural Networks." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jens Gregor, Major Professor

We have read this thesis
and recommend its acceptance:

James Plank

Bruce MacLennan

Accepted for the Council:

Carolyn R. Hodges
Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

An Exploration of Monophonic Instrument Classification Using Multi-Threaded Artificial Neural Networks

A Thesis Presented for the
Master's of Science Degree of Computer Science
The University of Tennessee, Knoxville

Marc Joseph Rubin
December 2009

Abstract

The use of computers for automated music analysis could benefit several aspects of academia and industry, from psychological and music research, to intelligent music selection and music copyright investigation. In the following thesis, one of the first steps of automated musical analysis, i.e., monophonic instrument recognition, was explored. A multi-threaded artificial neural network was implemented and used as the classifier in order to utilize multi-core technology and allow for faster training. The parallelized batch-mode backpropagation algorithm used provided linear speedup, an improvement to the current literature. For the classification experiments, eleven different sets of instruments were used, starting with perceptively dissimilar instruments (i.e., bass vs. trumpet), moving towards more similar sounding instruments (i.e., violin vs. viola; oboe vs. bassoon; xylophone vs. vibraphone, etc.,). From the 70 original musical features extracted from each audio sample, a sequential forward selection algorithm was employed to select only the most salient features that best differentiate the instruments in question. Using twenty runs for each set of instruments (i.e., 10 sets of a 50/50 cross-validation training paradigm), the test results were promising, with classification rates ranging from a mean of 76% to 96%, with many individual runs reaching a perfect 100% score. The conclusion of this thesis confirms the use of multi-threaded artificial neural networks as a viable classifier in single instrument recognition of perceptively similar sounding instruments.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

About music, the incomparable Kurt Vonnegut put it best: "if I should ever die, God forbid, let this be my epitaph: The only proof he needed for the existence of God was music." (Vonnegut, 2007). As a well-known cynic and atheist, Vonnegut's words emphasize the remarkable power music can have on a human listener. With the prevalence and volume of music in modern culture (e.g., ipods, car stereos, elevator music, commercial jingles, movie scores, birthdays, weddings, etc.), it is of great interest to better understand how and why music affects behavior (e.g., how music drives consumerism). Although such research is primarily the work of psychologists and philosophers (e.g., Janata, 2007), there is the potential for a tremendous amount of computing: from the calculation of individual musical components (e.g., pitch) to fully automated transcription, analysis and classification of the music data (Klapuri & Davy, 2006). In the following chapters, one of the basic building blocks of automated music analysis, monophonic instrument identification, is explored with emphasis on making a neural network approach computationally more feasible through use of multi-core technology.

One of the goals of an automated music analysis tool is to provide precise, objective and generalizable measurements for research purposes. In any psychological study, one of the primary hallmarks of a generalizable experimental design is the elimination of the researcher's subjective bias from the experiment. With regard to research on music and the mind, such subjectivity would include the musical background of the researcher. For example, Western music theory is based on a scale that has twelve tones per octave while Arabic music theory is based on a 24-tones per octave scale (Farmer, 1988). To a researcher more familiar with Western based music, the semitones (e.g., a note halfway between a B and C; or two adjacent

white keys on a piano) of the Arabic scale may sound completely out-of-tune and may be presented as such in an experiment; thus basing a portion of the study on the subjective bias of the researcher.

As another example, knowing that Wolfgang Amadeus Mozart was on his deathbed when he wrote "Requiem Mass in D Minor" (Cormican, 1991) might lead one researcher to believe the piece has a stronger negative emotional component (e.g., sadness) than perhaps a less musically educated investigator may perceive. With such subjective bias, the more musically educated researcher may use Mozart's piece as the quintessential mark of "sad" music, while other researchers may think otherwise. In essence, such subjectivity reduces the ability to generalize the results to larger populations.

In neuroimaging (i.e., functional Magnetic Resonance Imaging- fMRI) research regarding music and the brain (Janata, 2009), an automated music analysis tool would be helpful for researchers to know that at time T, the fMRI image data was F, and the music was M. Such a system would be very helpful in identifying patterns in how the brain reacts to music, and would provide more insight into how the brain works with regard to memory, emotion and attention.

Beyond the scope of academia, automated music analysis could be useful in industry. Such avenues include copyright investigation and protection, intuitive music database access (via humming or whistling), automatic play-list generation, genre classification, and products for music transcription. Even today musicians are still hired to analyze and classify music by genre in a tedious and error-prone process (as found in a phone interview with Adam Robinson, a former employee of Sound Flavor). The task of music analysis for genre classification (by humans) involves identification of all instruments present plus a harmonic and temporal analysis of the music (e.g., major vs. minor, consonance vs. dissonance, tempo, rhythmic structures, key,

mode, etc.). One could easily imagine writing software to automate this process using computers, but it has been found that automated music analysis is quite a complicated and difficult problem (Klapuri & Davy, 2006).

Despite its complexity, over the years there have been gains made in several avenues of automated music analysis including: genre classification (Tzanetakis, Essl & Cook, 2001), polyphonic instrument recognition (Chafe & Jaffe, 1986; Kitahara, Goto, Komatani, Ogata, & Okuno, 2006) and even mood detection (Lu, Liu, & Zhang, 2006). Although modern automated music analysis is still in its infancy, there have been significant results made in single (monophonic) instrument recognition. Before delving into the details of the previous literature regarding automatic monophonic instrument recognition, the motivated reader is encouraged to review Appendices A and B for details of the typical digital signal processing techniques used and a glossary of common timbral features extracted from an audio music file.

1.2 Summary of Previous Literature: Automatic Classification of Monophonic Instruments

Despite the volume of research conducted and books written about psychoacoustics (e.g., Sethares, 2004) and the existence of an international standard describing the acoustic features responsible for perceptual differences of musical instruments (i.e., Multimedia Content Description Interface or MPEG-7; Salembier, 2002); there does not exist a single, clearly defined algorithm or set of features for automated instrument identification. The use of computers to automatically classify musical instruments has been widely researched using a variety of classification methods (e.g., k-nearest neighbor, neural networks, decision trees, etc.,) on several different combinations of extracted features (e.g., spectral centroid, kurtosis, flux, inharmonicity, etc.,) (Herrera-Boyer, Peeters, & Dubnov, 2003). The following is a brief summary of the

previous literature involving single instrument classification and provides the grounds the experiment conducted.

In a large study conducted by Agostini, Longari and Pollastri (2003), several different classifiers (i.e., quadratic discriminant analysis, canonical discriminant analysis, nearest-neighbors and support vector machines) were tested on 18 features for each of over 1000 audio samples of 27 different instruments. The quadratic discriminant analysis and support vector machines methods performed the best, classifying the single instruments correctly 65% and 70% of the time, respectively. The study also provided a list of the top nine most discriminating features, with mean inharmonicity, spectral centroid mean and standard deviation as the top three. The researchers did not explain how they arrived at the 18 features used, and one is to assume they used a combination of previous literature plus trial and error. This study suggests that there is no single classifying methodology that works best for single instrument classification and that feature selection can be highly researcher dependent.

Eronen and Klapuri (2000) used 43 features and a Gaussian classifier to classify nearly 1500 samples of 30 different monophonic instruments using both hierarchical (e.g., sustained -> strings -> violin) and non-hierarchical methodology. The researchers were able to classify single instruments to 80% accuracy in the non-hierarchical method, performing better than the hierarchical version. The study suggests that a non-hierarchical model of single instrument classification is the better method (with regard to using class hierarchies), requiring less computation and allowing for more flexibility in the instrument voicing (e.g., a violin can sound either pizzicato or sustained, which could confuse a hierarchical classifier).

Zhang and Ras (2007a) used Naïve Bayesian and Decision Tree J48 classifiers to classify over 3200 instruments into four families: percussive, strings, non-pitched and woodwinds. The

authors extracted a set of features in addition to the timbre descriptors listed in the MPEG-7 standard and trained the classifiers with and without the additional features. Their results indicate that the MPEG-7 standard works well, but adding the additional features not listed in the MPEG-7 (i.e., tristimulus, brightness, roll-off, flux, MFCC's, etc.,) made their results more robust, with as much as an 11% increase in percent classified. From this article, it is clear that the MPEG-7 standard is a terrific starting point, but can be improved upon with the addition of other features.

Deng, Simmermacher, and Cranefield (2008) extracted 44 different features from each of nearly 800 single instrument samples and trained three different classifiers (i.e., k-nearest neighbor, support vector machine and multi-layer operceptron; k-NN, SVM and MLP respectively) to classify based on instrument family (i.e., Brass, Strings, Woodwinds, Piano). Much like the previous article, several additional features were extracted to supplement the MPEG-7 standard timbre descriptors including the zero crossing, and the mean and standard deviation of the: zero-crossing rate, spectral centroid, MFCC's, root mean square, spectral flux, and bandwidth. The classifiers were each trained with all 44 features and then the first five, 10, 20 and 30 principal components (using a principal component analysis or PCA). The results show that one of the best classifiers was the MLP, which was able to classify the instruments up to 95% accuracy with all 44 features. This article provides three key observations: 1) the importance of extracting features beyond those in the MPEG-7 standard, 2) the MLP can be used as an effective classifier and 3) reducing the number of features using the PCA does not significantly deteriorate the results: from 95% accuracy with all 44 features to 86% accuracy with only the top five principal components.

Zhang and Ras (2007b) used a three-level tree-classifying scheme to classify nearly 1600 recordings of 74 different instruments using more than a dozen audio and statistical features in addition to the MPEG-7 standard timbre descriptors. The top level of the tree represented the generalized instrument family (e.g., chordophone, idiophone, aerophone); the second level represented how the instrument was played (e.g., reed, struck, shaken) and the third level was the specific instrument class (e.g., flute, piano, etc.). For each level of the classification tree, Zhang used a logistic regression model with a 99% confidence interval to select the discriminating features and found it significantly improved the classifier performance when compared to using all features. Another interesting result of this study was that the list of salient features changed drastically depending on the level of the tree classifier. Zhang's research highlights three observations: 1) the importance of using additional features not listed in the MPEG-7 standard; 2) the significance of applying a "smart" feature selection algorithm to reduce the dimensionality of the input data; and 3) there is no universal feature set that can discriminate all levels of single instrument classification.

In reviewing the salient literature and two textbooks on music signal analysis, it is apparent that there does not exist a clearly defined, standard set of audio features used to automatically differentiate instruments based on timbre. It is also evident that there are an exponential number of possible combinations of instruments, features and classification methods that could be used for automatic classification of single instruments. In addition to the enormous number of combinations, the result of processing a single audio file could easily yield thousands of data points. For example, if a five second audio file was divided into 50ms windows, and 100 features were extracted from each window, then that would yield 10,000 data points for a single, five second audio file! Due to the sheer volume of input data for a classifier to train with, in the

following experiment an artificial neural network was used the training process can be made substantially faster by utilizing threads and multi-core technology.

1.3 Overview of Remaining Chapters

In the chapters to follow, the details and speedup of a multi-threaded implementation of an artificial neural network are discussed (Chapter 2), followed by the design (Chapter 3), results and analysis (Chapter 4) of a monophonic instrument classification experiment using parallel neural networks. Chapter five contains conclusions that can be drawn from this thesis as well as future directions of such research. Appendix A and B contain information about basic music signal processing and a glossary of common musical features.

Chapter 2: Parallelization of an Artificial Neural Network

2.1 Basics of an artificial neural network

Modeled after the human brain, artificial neural networks have been around for decades and are well known for their ability to classify non-linearly separable data (e.g., XOR in Boolean algebra: 0 + 0 = 0; 0 + 1 = 1; 1 + 0 = 1; 1 + 1 = 0). The following is a brief summary of the structure and implementation of a multi-layer artificial neural network (i.e., ANN) and a more detailed explanation of how parallelization was achieved for the experiment. For a much more detailed explanation of an artificial neural network, including the mathematics involved, the reader is encouraged to consult Pattern Classification by Duda, Hart and Stork (2001).

The basic building block of an ANN, much like the human brain, is a neuron or node. Each of these neurons contains an activation function (e.g., a logistic sigmoid transfer function, Figure 1 or hyperbolic tangent sigmoid transfer function, Figure 2) that produces a real number as output and the input is a linear combination of items to be discussed. These neurons are organized into layers with an input layer, hidden layers and an output layer. The input layer has the same number of neurons as features in the input data (e.g., two in the classic XOR example listed above) and the output layer contains one or more neurons, depending on the classification label (e.g., a binary label requires three nodes for three classes). Although the input and output layers are clearly defined, the hidden layer(s) allow for much more flexibility. It is generally up to the experimenter to decide on how many hidden layers and neurons in each hidden layer for a particular classification task.

The layers of an ANN are connected such that each neuron in a layer is connected to all neurons in the subsequent layer. Each of these interconnections has a weight associated with them, which gets included in the linear combination that serves as input to the node's activation

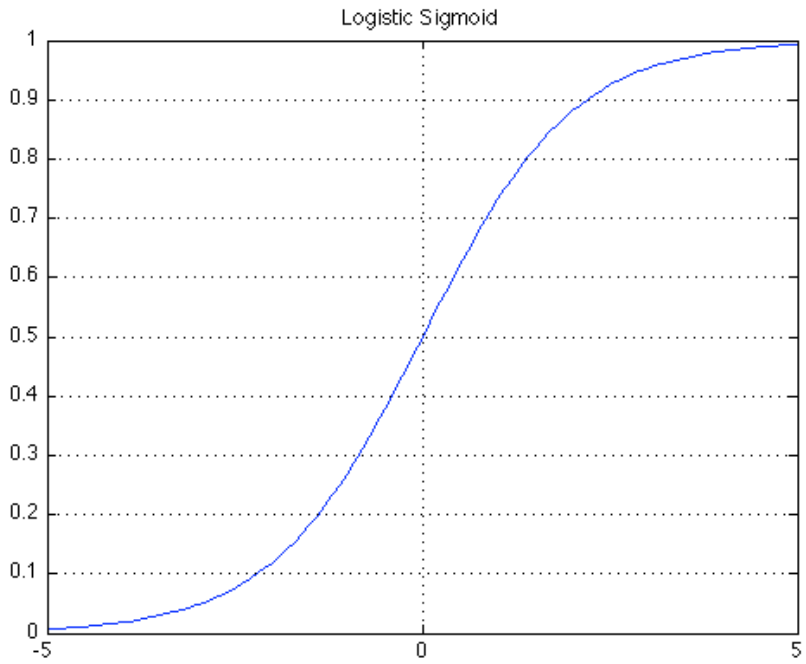

Figure 1: Logistic sigmoid transfer function is commonly used as the activation function in neurons

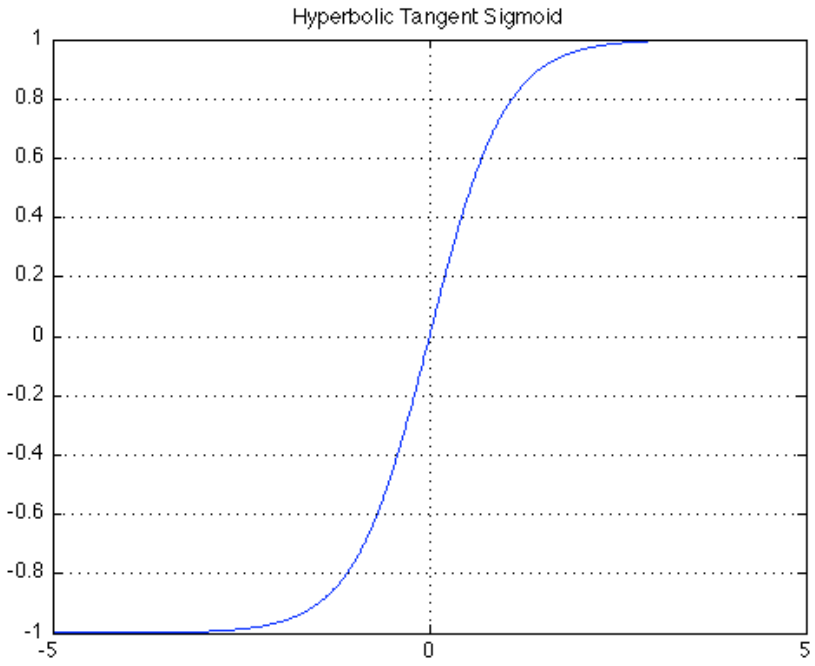

Figure 2: Hyperbolic tangent sigmoid transfer function is another commonly used activation function in neural networks

function (in addition to the output from the previous node's activation function). When a neural network gets trained using the backpropagation algorithm, these connection weights get systematically adjusted to minimize the classification error.

A highly simplified explanation of the backpropagation algorithm is as follows. First a training sample with a known class (or target) is presented to the neural network. Next, the neural network calculates the output by allowing the input to "feed-forward" through all the layers and neurons in the network using the previously mentioned activation functions. Once the output of the network is calculated, the error (i.e., difference between the desired target and actual output) is "propagated" backwards through the network, allowing each layer's weights to be adjusted (scaled by the learning rate), to best minimize the error. This process is then repeated with the entire set of training samples until the network either converges to an allowable average error (i.e., mean square error or MSE) or trains for a maximum number of epochs (where each epoch represents a single pass through the entire training set).

A helpful analogy of the backpropagation algorithm is adjusting the knobs on a television set (e.g., antennae, contrast, color, brightness, etc.) to get a clear picture. In this case, the person watching the television has an idea of the target (i.e., a clear picture), and the knobs, representing the weights, get adjusted accordingly.

When using the backpropagation algorithm to train the network, there are two main paradigms to choose from: incremental or batch. In the incremental implementation, the weights get updated after each sample is presented to the neural network. In batch mode, the weights only get updated to the average weight change after all samples have been presented to the network (i.e., after each epoch). One of the main advantages to using a batch implementation is

that the algorithm can be parallelized to utilize multi-core technology and achieve linear speed-up of the training process.

2.2 Parallelization of an Artificial Neural Network

Implementing a multi-threaded neural network to achieve faster training is rather straightforward in batch mode. As previously mentioned, in batch mode the weight matrix only gets modified to the average weight change at the end of each epoch. In the previous literature, algorithm speed-up was achieved by using parallelization in a variety of ways: from faster matrix multiplication of the feed-forward pass using a Message Passing Interface (MPI), to parallel network structure using Distributed Memory Multi-processors (DMM) in various topologies, to training level parallelism using threads on Shared Memory Multi-processors (SMM) and Parallel Virtual Machines (PVM). However, the calculated speedup results of each of these methods are not linear, and tend to plateau after 4 threads or processors. In this thesis, the training level parallelism approach was enriched to approximate ideal speedup (e.g., 2 cores are twice as fast) on an 8-core SMM machine.

Novokhodko and Valentine (2001) implemented parallel matrix arithmetic in C using an MPI on 4 Sun Ultra 5's. The authors then used Matlab's neural network toolbox with their version of matrix multiplication and attempted to classify over 5000 samples with a large neural network (20,000 neurons). Though the authors only tested 1 and 4 processors, their best speedup was a far from ideal 1.46.

Yoon, Nang and Maeng (1990) attempted to speedup training by distributing a single neural network horizontally amongst a topology of computers, where each processor got a portion of the input, hidden and output nodes. Parallelization was achieved by simultaneously broadcasting the delta weights amongst all processors in this DMM setup. A more promising

speedup of about 6.0 was reported with 8 processors. However, this approach requires multiple machines to implement the ring topology, which may not be as portable or practical for a user with a single machine.

When it comes to training level parallelism, the basic algorithm works by allowing N threads, each with their own copy of the original (i.e., master) neural network, to train on disjoint sets of the training data and then update a master network. After the threads accumulate the delta weights for an epoch, the threads join and one processor sequentially updates the master network and calculates the MSE (Ahmad, Zulianto & Sanjaya, 1999; Fedorova & Terekhoff, 1999). This process is repeated until either the master network is trained (i.e., acceptable MSE threshold) or some maximum number of epochs is achieved.

Ahmad, Zulianto and Sanjaya, (1999) used PVM's to implement the training level parallelism algorithm mentioned above. In particular, they relied on PVM software in UNIX to distribute the training work-load across many computers. Their results show that this method does not approximate the ideal linear speedup: 1.35 with 2 processors, 1.5 with 3 processors, 1.6 with 4 processors and 1.7 with 6 processors. Perhaps the overhead associated with the message passing between computers is what slowed this down. Tsaregorodtsev (2005) implemented this algorithm on a dual-core SMM to utilize threads and avoid the overhead associated with message passing between computers. The parallel implementation was tested using 1 and 2 threads, resulting in speedups that were not ideal: between 1.2 and 1.6 with an average of 1.5.

Turchenko and Grandinetti (2009) implemented training level parallelism and tested it using a variety of network and training data sizes on an 8-core machine with 2 and 8 threads. They performed an efficiency analysis of the training on 2 and 8 processors, with results showing that 2 processor were nearly ideal with efficiencies between 90 and 100 percent. However, with

8 processors the efficiencies were far from ideal: between 45 and 75 percent. Turchenko and Grandinetti also make the very strong conclusion that going from 2 to 8 processors is simply too inefficient for practical use.

Based on the timing, speedup and processor efficiency results of the previous literature, it is clear that the presented algorithm for training level parallelism on SMM's does not approximate an ideal linear speedup between 4 and 8 threads. In particular, the speedup seems to plateau after four threads. Upon further review of the algorithm, it appears that although the weight changes are calculated in parallel, much of the required work is still executed sequentially on a single processor: i.e., calculating the MSE and accumulating the delta weights. In this thesis a modified version of the parallel training algorithm was implemented to more evenly distribute the workload and approximate an ideal linear speedup.

2.3 Improved training level parallelism for Shared Memory Multi-processor

In this thesis, two enhancements were made to the aforementioned parallel training algorithm to avoid the sequential bottlenecks. First, the task of calculating the MSE for the training set was distributed among threads and second, the delta weights were accumulated in parallel by utilizing the parallelism inherent in matrix arithmetic. The summary of this enhanced algorithm is listed in .

The algorithm works as follows: first, the training data is divided into N disjoint subsets, where N is the number of threads. Next, after initializing the master neural network, all synchronization primitives and global data structures, N threads are created and started; and they do not return until convergence is met. Within each thread, first a local weight matrix is overwritten by the master network's weights, then the squared error is calculated for the thread's training data subset and accumulated in a global structure. After all threads complete this step,

Divide data into N disjoint subsets

1. Initialize master network and synchronization primitives
2. Spawn N threads
    a. Repeat until convergence: (within each thread)
        i. Overwrite local network with master network
        ii. Calculate local squared error for subset of training data
        iii. Block
        iv. Last thread to finish:
            1. Calculates global MSE
            2. Checks for MSE convergence
                a. Sets flag
            3. Broadcasts for threads to resume
        v. Check flag for convergence
            1. Exit thread if necessary
        vi. Accumulate weight changes for training subset via batch mode backpropagation
        vii. Block
        viii. Last thread to finish:
            1. Zeros out shared weight matrix
            2. Broadcasts for threads to resume
        ix. Sum a portion of shared weight matrix
            1. Parallel matrix addition
        x. Block
        xi. Last thread to finish:
            1. Updates the master network to average
            2. Increments epoch counter
            3. Broadcasts for threads to resume
        xii. Check for convergence via epochs
            1. Exit thread if necessary
3. Join N threads
4. Write master network to file

Figure 3: Algorithm used for a parallelized implementation of a neural network in batch mode

they wait on the conditional variable, and the last thread to finish calculates the overall mean squared error. The last thread then checks the MSE against the threshold set by the user, sets a global flag and broadcasts on the conditional variable, resuming all blocked threads.

At this point all threads check the global flag for convergence and exit if appropriate. If the flag was not set, each thread begins accumulating the necessary weight changes via the feed-forward backpropagation batch mode algorithm. After each thread completes training on its data subset, it blocks on the conditional variable, and the last thread to finish zeros out a temporary global weight matrix and broadcasts to restart all blocked threads.

When the threads resume execution, each thread updates a different portion of the global weight matrix, accumulating the total weight change over all threads for a particular index range (Figure 4). Again, after each thread completes this step, it blocks on the condition variable and the last thread updates the master network as the average of the temporary global weight matrix and increments a global epoch counter. The last thread then broadcasts on the conditional variable, resuming the execution of all threads. Finally, each thread checks the global epoch counter and exits if appropriate.

The implementation was achieved by modifying an open-source neural network library (i.e., NNF - Neural Net Framework written in C++ by Alessandro Presta, 2005) to support threads and utilize multi-core technology using the Portable Operating System Interface for Unix (POSIX threads).

2.4 Testing the parallel network

The implementation of the enhanced parallel training algorithm was tested with one, two, four and eight threads on an 8-core machine, namely samwise.eecs.utk.edu. One such test,

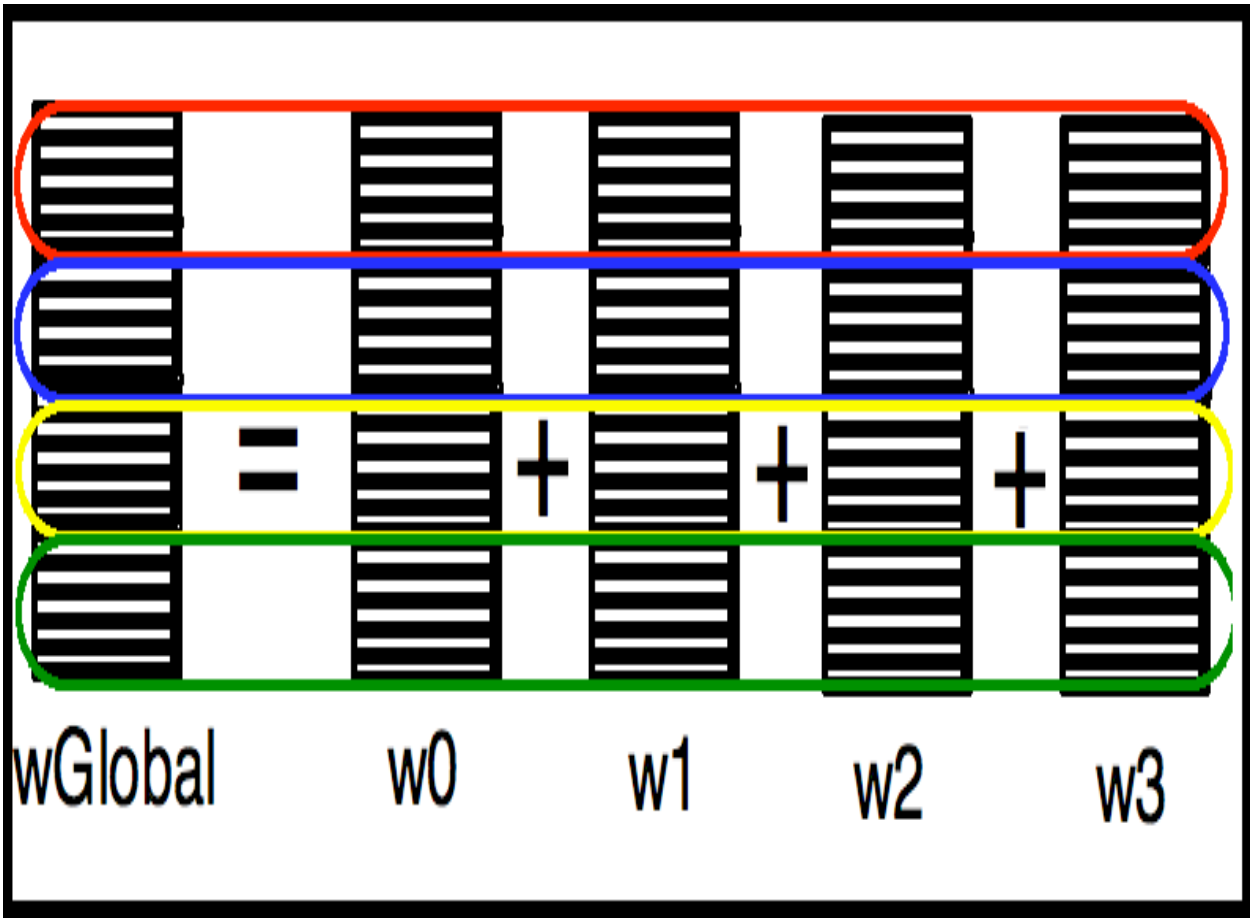

Figure 4: Parallel summation of delta weights. In this picture there are 4 threads, each with its own calculated weight matrix w0 – w3. Each thread (red, blue, yellow, green) updates a portion of the global weight matrix, wGlobal, in parallel.

training a computer to classify a three input XOR problem, is presented here (i.e., 0 0 0, 0; 0 0 1, 1; etc.,). The XOR data was synthesized to contain 16,000 rows in order to replicate a very large training database. Although the XOR classification task has been done to no end, it still successfully simulates the amount of work and time required to train a modest sized network (i.e., 3 inputs nodes, 10 hidden nodes, and 1 output node) with a substantially large dataset.

For the testing, the parallelized neural network was trained for 10,000 epochs in attempts to reach a convergence mean square error (MSE) of 0.000001. The extremely high number of epochs and very low MSE for this relatively simple classification task were used to mimic a more difficult classification problem such as instrument identification. The results show that as the number of threads increase from 1 to 8, the time to train decreased dramatically (Figure 5) and there is almost ideal linear speedup (Figure 6). Also, unlike the previous literature, the processor efficiency did not drop below 95% (Figure 7).

It is crucial to note that the percent mis-classified did not change when scaled from one up to eight threads (Figure 8). It is also important to note that the final weight matrices of the trained neural network did not differ as the number of threads varied (Figure 9). One last essential observation includes the convergence of the MSE (Figure 10), which shows that, after each epoch, the neural networks were training in the exact same manner, regardless of the number of threads.

With the speedup achieved while maintaining performance, a parallelized neural network provides faster training, which, in turn, allows for larger training sets and more classification experiments to be completed. As previously mentioned, due to the enormous number of combinations possible for automatic instrument classification, having a faster classifier is critical for completing more experiments in a reasonable amount of time.

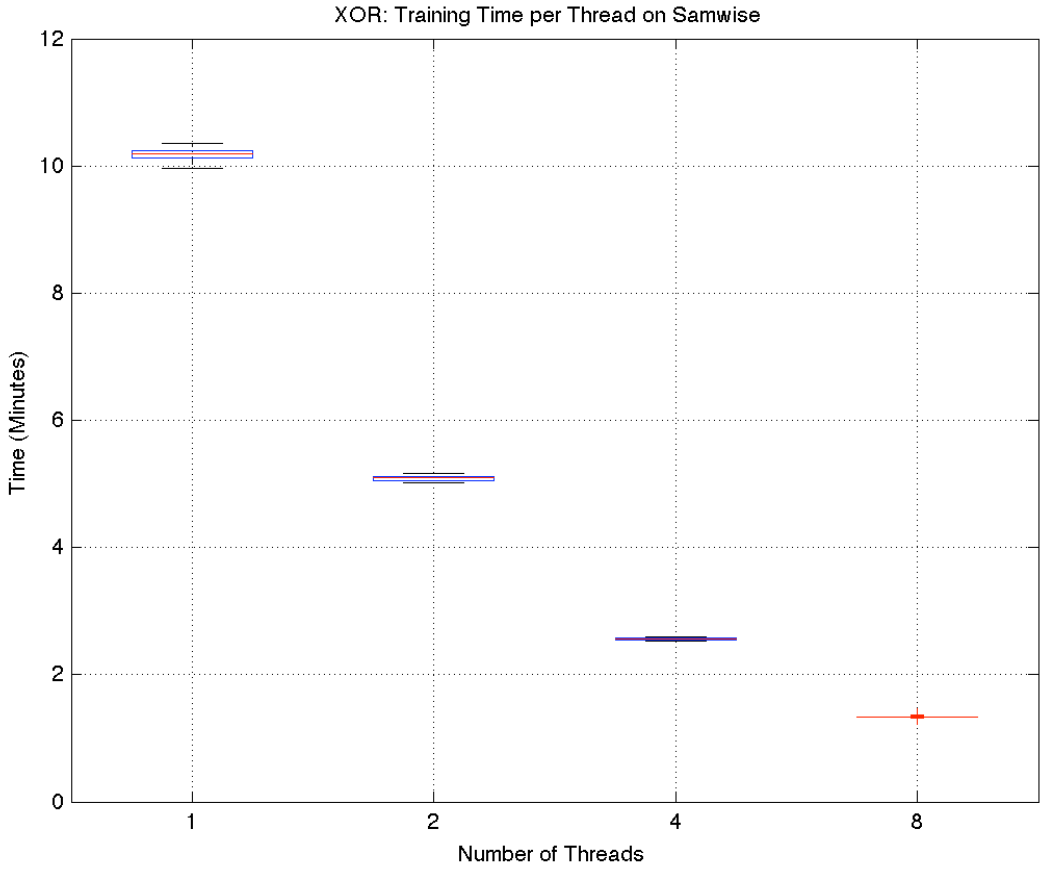

Figure 5: Large XOR dataset, box-plot of time to train (the red line is median)

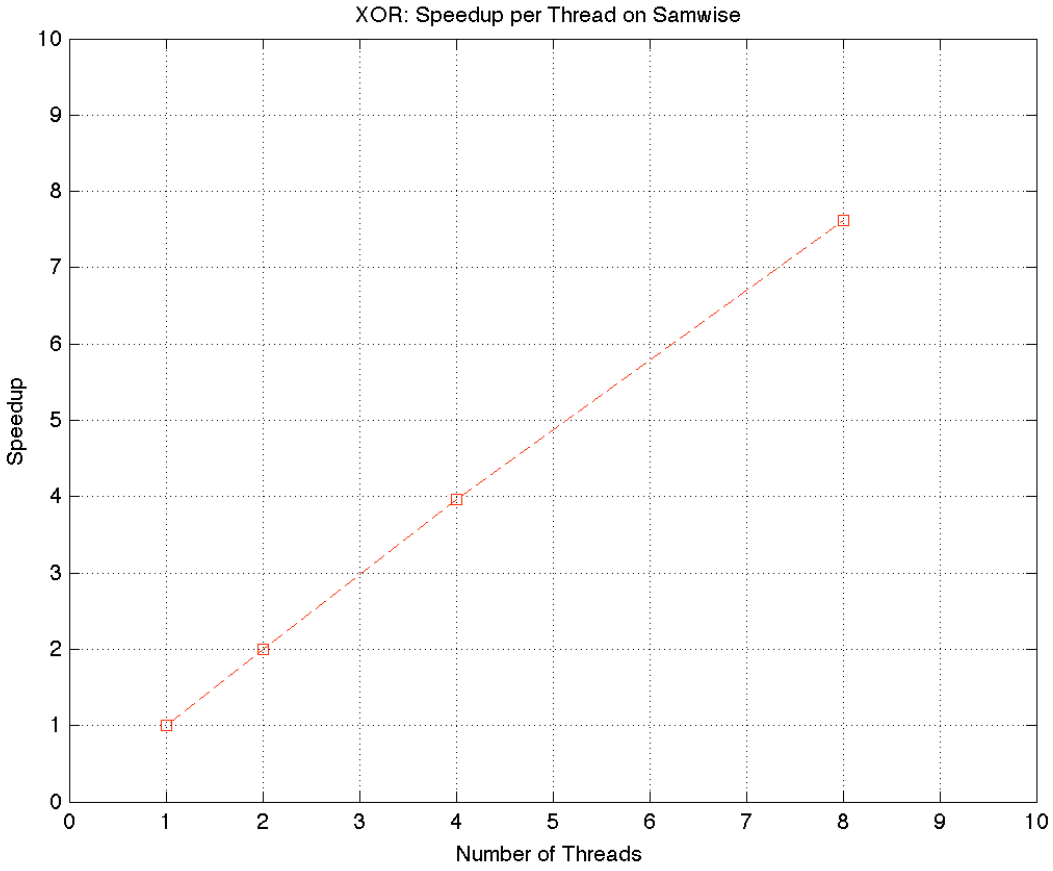

Figure 6: Large XOR dataset, calculated speed-up

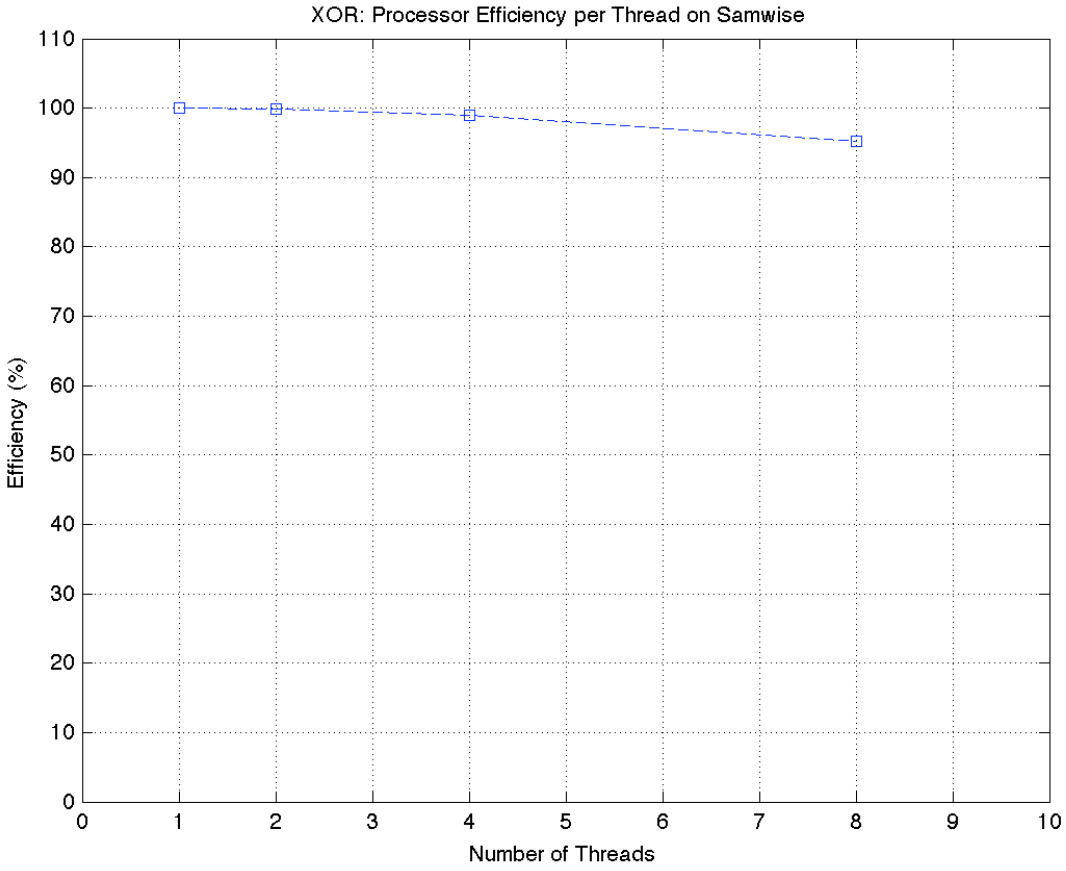

Figure 7: Large XOR dataset, processor efficiency

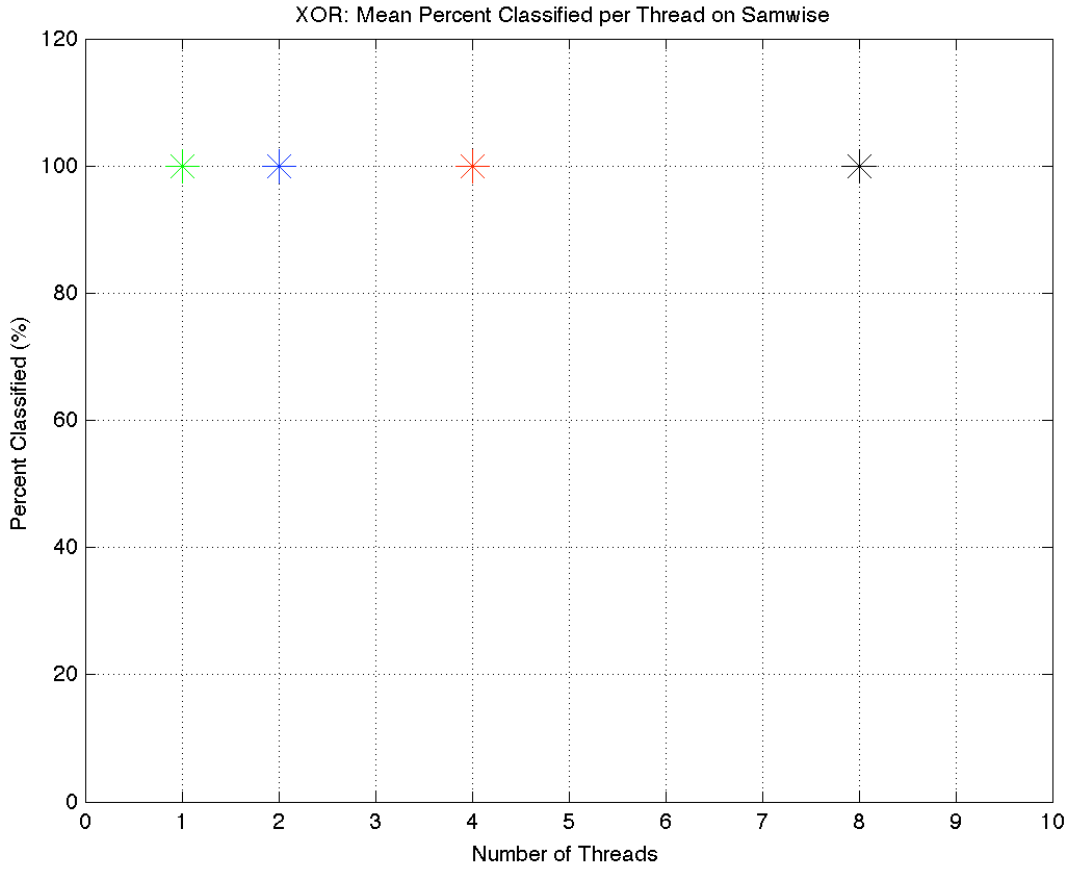

Figure 8: Mean percent classified for the large XOR dataset per 1, 2, 4 and 8 threads. Note how the percent classified does not vary as the threads increase

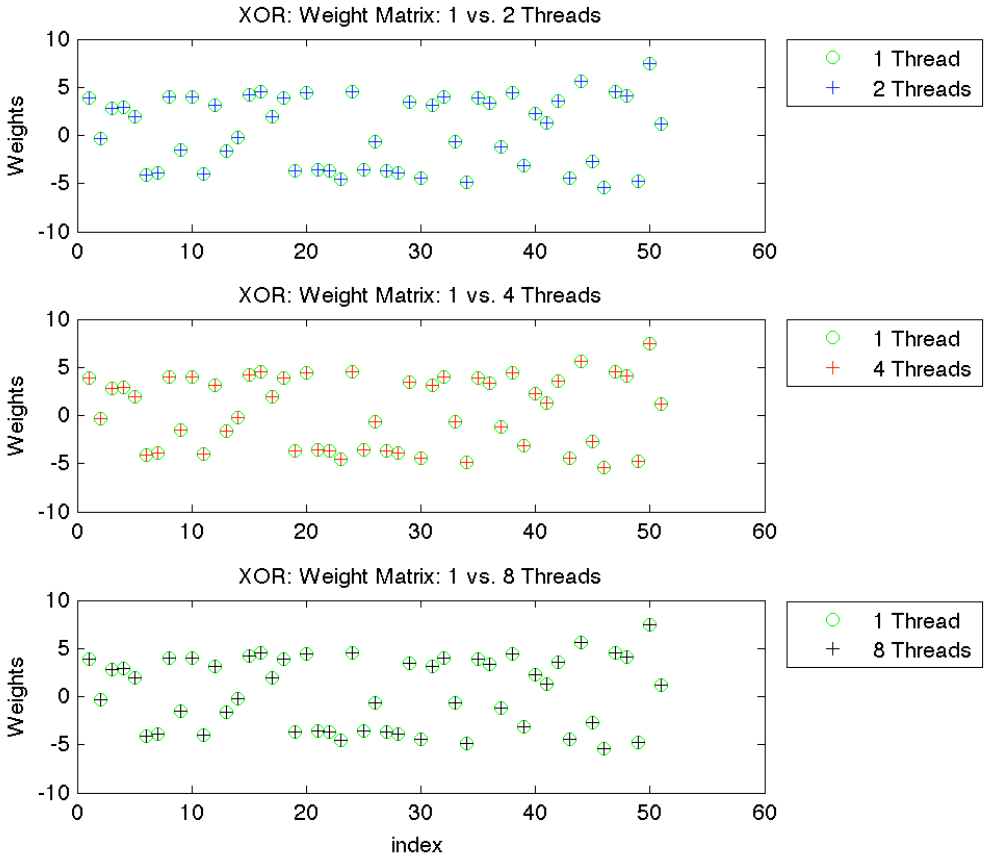

Figure 9: Weight matrices for the final trained neural network for the first run. Note how all the weight matrices are equal

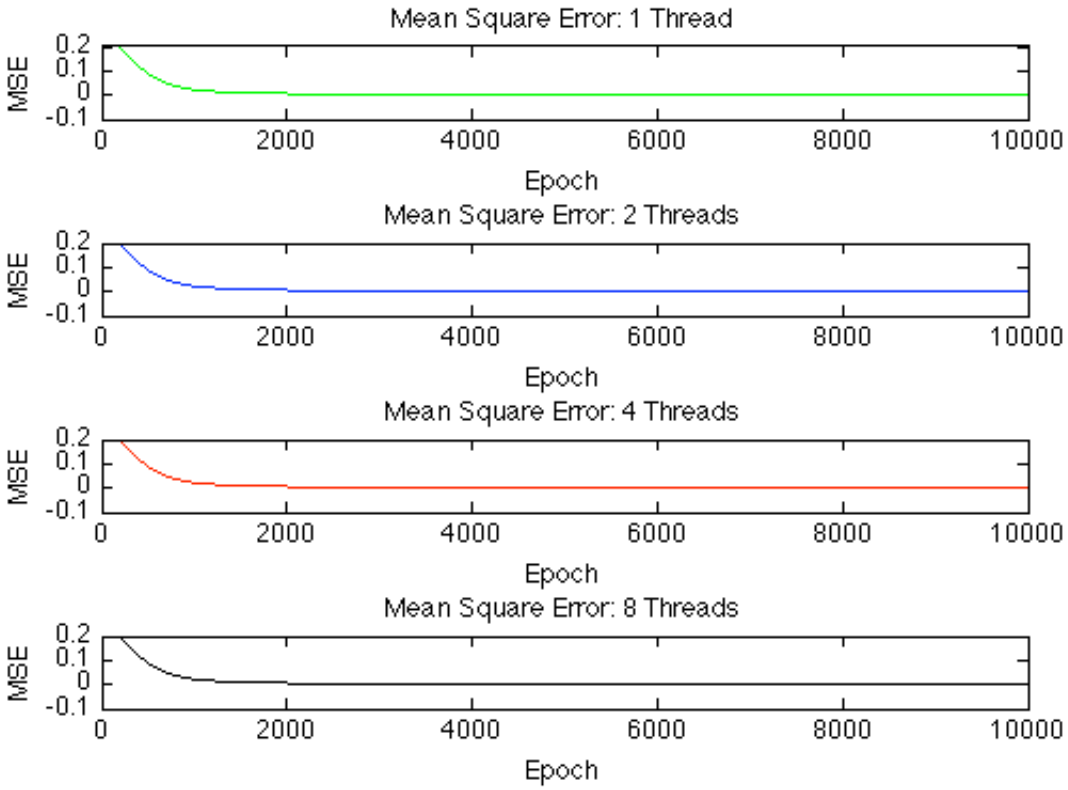

Figure 10:Mean square error per thread over 10000 epochs. The MSE's for each thread did not differ, as the 'diff' utility in UNIX did not reveal any differences between the error

Chapter 3: Experimental Setup

Several classification experiments were conducted on many combinations of instrument samples using the aforementioned parallelized neural network. The experimental design is broken down into the input to the system, the computing involved, and the results obtained. A statistical analysis of the results and future direction follows.

3.1 Input

The input to the system consisted of 2 – 15 second audio samples of many different instruments, from orchestral standards (e.g., violin, oboe, bassoon, etc.) to more contemporary instruments (e.g., electric guitar, organ, etc.). The instrument samples were taken from the McGill University Master Samples (i.e., MUMS), a three DVD collection of nearly 6600 audio (i.e., .wav) files totaling more than 10 gigabytes of audio data. All of the recordings on the MUMS DVD's were played by professional musicians and recorded with top of the line audio equipment and engineers. McGill University created the Master Samples to provide musicians with a comprehensive set of samples to add to their music, as well as supply researchers with a standardized set of instrument recordings to use in a variety of experiments, from psychoacoustics to artificial intelligence.

The audio data in the MUMS DVDs is all-encompassing, covering every possible note that can be played by the instrument in question, and containing several different voicings of each instrument. For example, there are over 400 different recordings of a violin, with voicings that include: bowed, plucked, harmonics and vibrato. The high quality of the audio combined with the multitude of notes and voicings allow the samples from the MUMS to be highly generalizable, which provides a solid basis for the experiments presented.

3.2 Computing Involved

Every set of musical instruments presented to the multi-threaded neural network was preprocessed the exact same way. All of the features extracted from the audio file were done so with the MIRToolbox, an extensive, open-source toolbox for Matlab written by researchers at the University of Jyvaskyla in Finland (Lartillot & Toiviainen, 2007). MIRToolbox was created to give researchers an easy to use framework to extract high-level musical features describing the signal's tonality, rhythm and timbre with simple and intuitive function calls.

3.2.1 Feature Extraction

The steps involved in preprocessing were as follows. Each audio file was first "trimmed" which removed any excess noise before and after the instrument played (Figure 11, Figure 12). The MIRToolbox described this as "trimming the pseudo-silence beginning and end off the audio file." In this case, the pseudo-silence is defined as 0.06 * (Medium RMS), where RMS stands for the root mean square of the segment; a feature that often describes the perceived loudness of an audio signal (see Appendix B for more details).

After the signal was trimmed of excess noise, the audio was divided into 50% overlapping, 50 ms frames (Figure 13). Once the signal was divided into frames, 70 features were extracted to describe the entire signal. The features are listed in Table 1. For a more detailed explanation of the features, please consult Appendices A and B. Any data rows containing missing data (i.e., NaN's) were omitted from the final datasets.

After the audio file is preprocessed, an identifying label is assigned (i.e., a binary label), and the row of data is appended to the data set in question (e.g., bass vs. trumpet, violin vs. viola, etc.,). The binary label is not a true binary representation; for example, a three-class problem would have labels 0 0 1, 0 1 0 and 1 0 0. Once all the instruments have been preprocessed, each

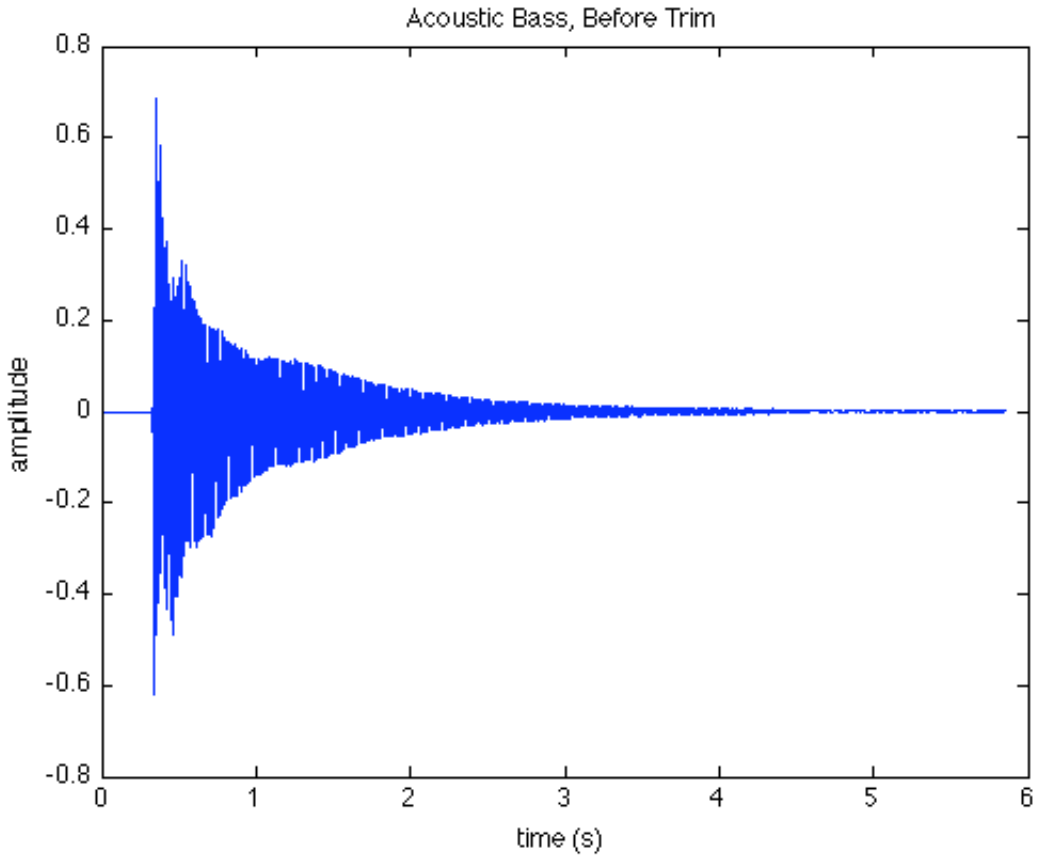

Figure 11: Acoustic bass, before trimming off the psuedo-silence

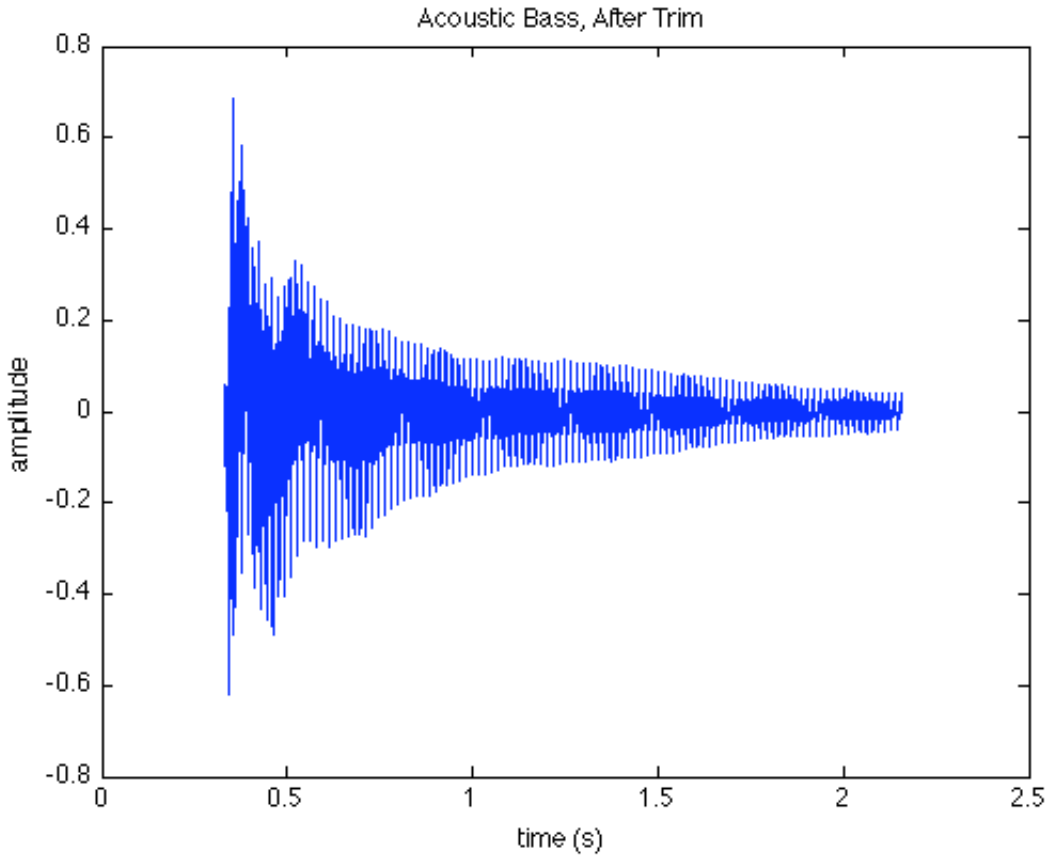

Figure 12: Acoustic bass, after trimming the pseudo-silence from the beginning and end of signal

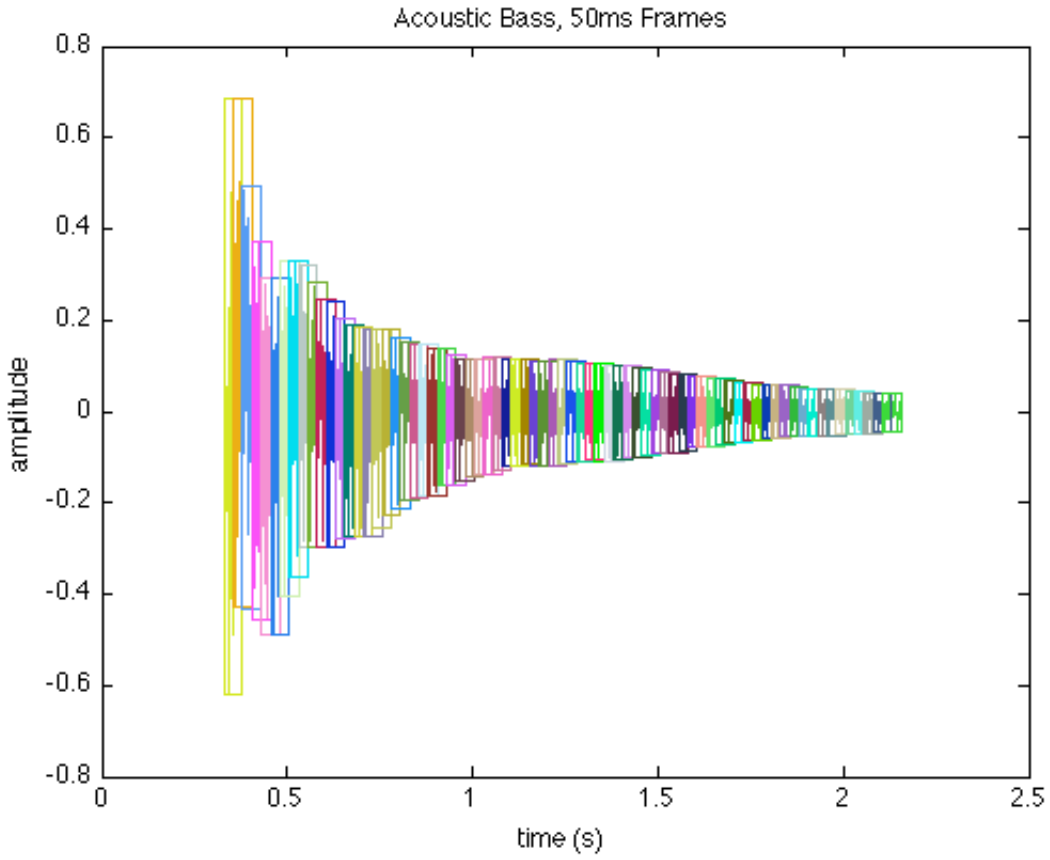

Figure 13: Acoustic bass divided into 50% overlapping, 50 ms frames

Features Extracted From Each Audio File
Computed From Entire Signal:
Total Zero-crossings
Fundamental Frequency
Mean and Standard Deviation of all Frames of the Following:
Attack Slope
Brightness
Event Density
Inharmonicity
Log Attack Time
Mel-Frequency Cepstral Coefficients (13 total bands)
Regularity
Rolloff (85%)
Root Mean Square
Roughness
Spectral Centroid
Spectral Flatness
Spectral Flux
Spectral Kurtosis
Spectral Skewness
Spectral Spread
Temporal Centroid
Temporal Flatness
Temporal Kurtosis
Temporal Skewness
Temporal Spread
Zero-crossing Rate

Table 1: Features extracted from each audio file processed

feature column is scaled by dividing each element by the maximum value in that column, leading to a dataset that is between –1.0 and 1.0. This scaling is performed because neural networks typically train faster and more thoroughly with more normalized datasets (Sola & Sevilla, 1997).

3.2.2 Feature Selection

After the feature extraction completes for a particular set of instruments (e.g., bass vs. trumpet samples), the result is a dataset that may have extraneous features, columns that do not help differentiate the instruments in question. In order to select only the most salient features that separate the instruments, a sequential forward selection (SFS) algorithm is applied to the dataset, with a p value of 0.001. SFS is an automatic procedure that systematically adds the columns that most best describe the targets (Koutroumbas & Theodoridis, 1999). The sequential feature selection algorithm is as follows: SFS starts with the feature most highly correlated with the targets and adds new columns which, combined with the old one(s), most accurately predict the targets. The algorithm halts when a new feature does not significantly reduce the prediction error where the statistical significance is measured by a partial F-test. In this experiment, an open-source feature selection toolbox in Matlab from Oxford's Pattern Analysis and Machine Learning Group was used (i.e., the FSBOX written by the Pattern Analysis & Machine Learning Research Group of Oxford University 1999).

Using the SFS algorithm to reduce the dimensionality of the data had two benefits: 1) it drastically reduced the time to train for the parallelized neural network (Figure 14), and 2) it improved the classification performance of the trained neural network (Figure 15).

Figure 14: The time (in minutes) for the neural network to train, before and after selecting features using the forward selection algorithm

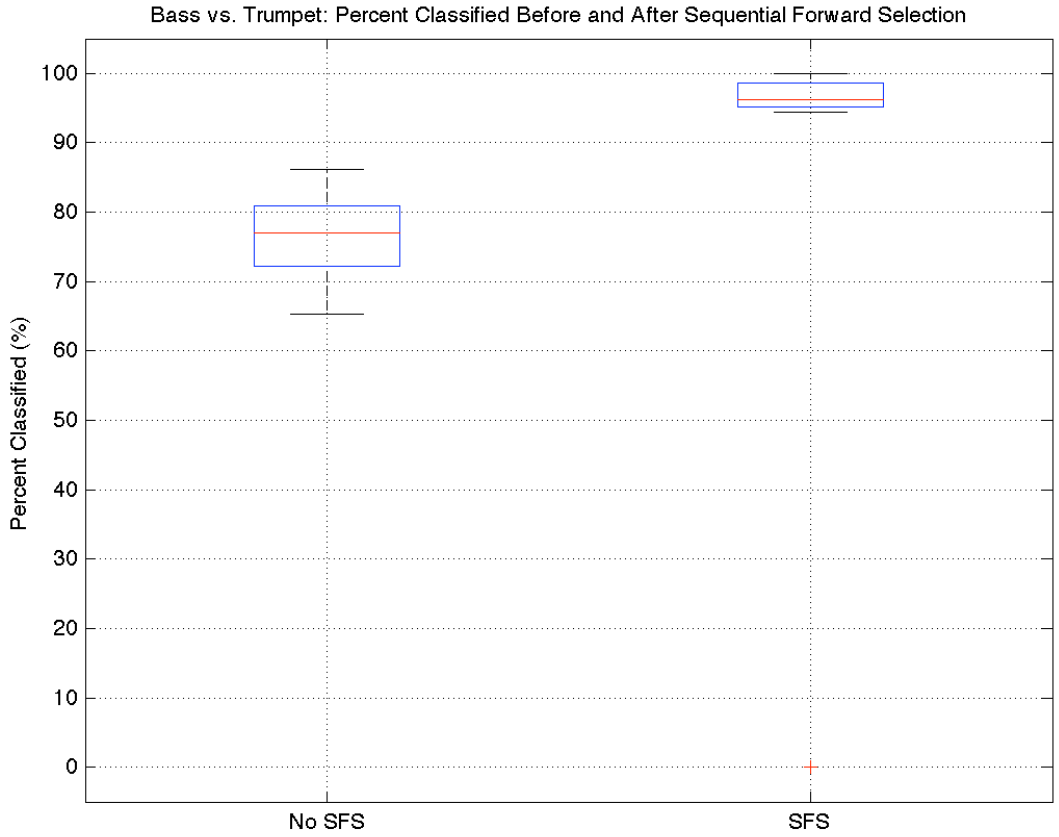

Figure 15: Percentage of samples classified correctly by a neural network trained with all features (No SFS), and only selected features (SFS)

3.2.3 Training and Testing the Parallel Neural Network

Each classification experiment utilized a 50/50 cross validation paradigm, where 50% of the input data was used for training the neural network, and 50% was used for testing; after which the training and testing sets were swapped and the training/testing was repeated with the new sets and fresh neural networks. This process was repeated ten times for each set of instruments, using a different starting seed to randomly initialize the weight matrix on each run. For the testing phase, if the neural network's output was within 0.4 of the target, it was counted as correctly classified.

In order to reduce the number of combinations of possible training parameters (i.e., learning rate, number and size of hidden layers, maximum epochs, convergent mean square error), an attempt was made to keep the training parameters equal throughout all the classification experiments. The learning rate was set to 0.7, the maximum number of epochs was 10,000 and the convergent average mean square error was 0.01.

The number of neurons in each layer slightly differed between various sets of instruments based on the number of features selected and the number of classes. In order to combat this, in all cases the hidden layers were arranged as follows: the first hidden layer was twice as large as the input layer, and each subsequent hidden layer was half as big (i.e., using integer division with no remainders) as the previous, until the number of neurons equaled the number of output nodes or classes.

Chapter 4: Results and Analysis

Several experiments regarding automatic instrument recognition were conducted, starting with instruments that sound notably different and leading to musical sounds that are similar: i.e., in the same family of instruments (e.g., strings) and are perceptively similar (e.g., violin, viola). Since the majority of the previous literature classified the instruments based on instrument family, (e.g., Woodwinds, Brass, Strings), there was little need to explore it again here. Please note that the group scatter plots below are of the three canonical principal components (from a principal component analysis) of the entire dataset and are used for the purpose of visualization only.

Starting with notable different sounding instruments, 288 total audio samples of a 155 basses and 133 trumpets (Figure 16) were tested using the aforementioned computational process. The results provided a good start, with a median classification rate of 96% (Figure 17, Table 2). The SFS algorithm selected 16 features (listed in Table 3).

To make training a bit more difficult, two more instruments were thrown into the mix: a flute and a piano. On this test, 547 total instruments were processed and tested (Figure 18). The neural network was able to classify the correct instrument 80% of the time and the SFS algorithm selected 13 features.

After attempting to classify instruments that were notably and markedly different (perceptually), a set of similar sounding instruments were tested from each family or class of instruments (i.e., guitars, keyboards, strings, woodwinds, brass and percussion). The first set of similar sounding instruments consisted of guitars: acoustic vs. electric guitar. The data consisted of 195 total audio samples: 157 electric and 38 acoustic (Figure 19). The classification test resulted in a median percent classified of nearly 95% of novel stimuli for six selected features.

Figure 16: Group scatter plot of the first three canonical principal components of bass and trumpet

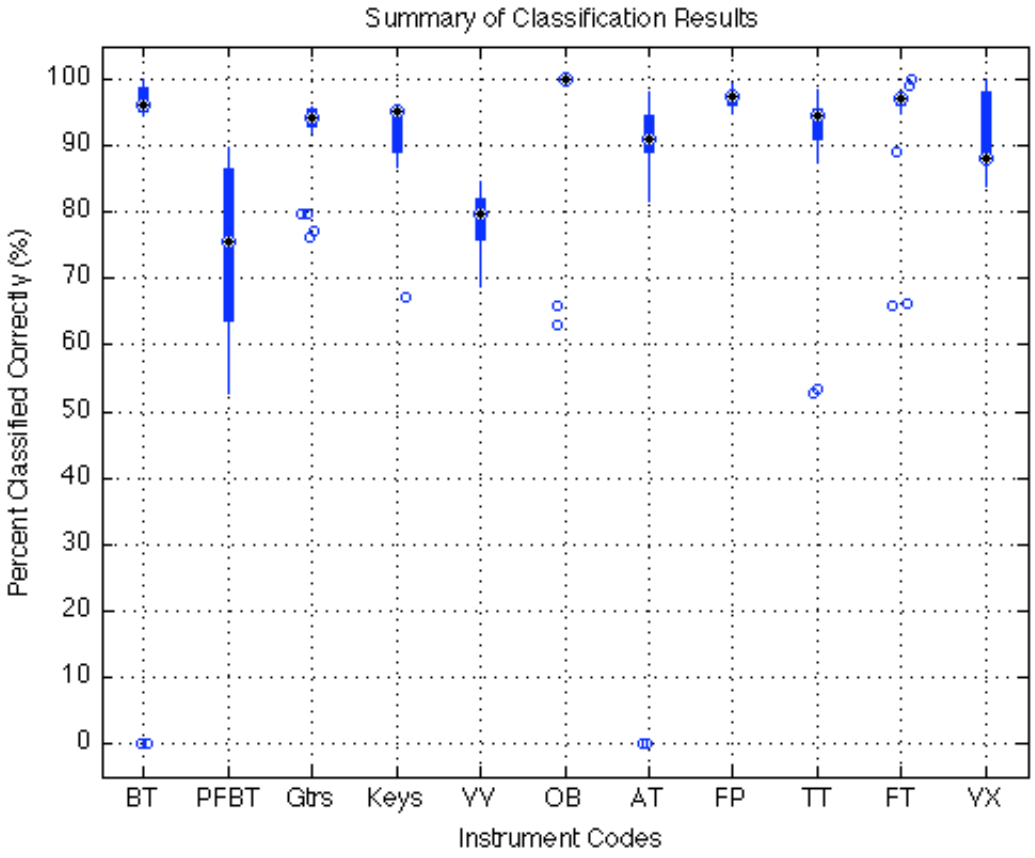

Figure 17: Summary of classifications

Percent Classified Correctly					
Instruments	**Mean**	**Median**	**Standard Deviation**	**Minimum**	**Maximum**
Bass vs. Trumpet	87.33	96.18	29.92	0.00	100.00
Piano, Flute, Bass, Trumpet	74.20	75.73	13.36	52.75	89.78
Electric vs. Acoustic Guitar	91.38	94.36	6.87	76.29	95.92
Piano, Harpsichord, Celesta, Organ	92.23	95.31	6.81	67.22	96.25
Violin vs. Viola	77.99	79.63	5.12	68.89	84.44
Oboe vs. Bassoon	96.45	100.00	10.94	63.16	100.00
Alto vs. Tenor Saxophone	82.70	91.07	28.51	0.00	98.18
Flute vs. Piccolo	97.28	97.46	1.35	94.93	99.28
Trombone vs. Trumpet	89.88	94.61	12.82	52.89	98.35
French Horn vs. Trumpet	93.58	97.00	9.60	66.00	100.00
Vibraphone vs. Xylophone	91.47	88.24	5.46	84.00	100.00
Average of all instruments	88.59	91.78	11.89	56.92	96.56

Table 2: Summary of Classification Results

Selected Features for Each Set of Instruments using SFS algorithm		
Bass vs Trumpet	**Piano, Organ, Harpsichord, Celesta**	**Alto vs. Tenor Saxophone**
event_density_mean	brighness_mean	mfcc_mean_02
flux_mean	flux_std	mfcc_mean_05
mfcc_mean_01	mfcc_std_01	mfcc_mean_06
mfcc_mean_02	mfcc_std_03	mfcc_mean_07
mfcc_mean_04	regularity_mean	mfcc_mean_08
mfcc_mean_08	rms_std	regularity_std
mfcc_mean_11	rolloff_mean	roughness_std
mfcc_mean_13	rolloff_std	
mfcc_std_01	roughness_std	**Trumpet vs. Trombone**
mfcc_std_02	spectral_centroid_std	log_attack_time_mean
mfcc_std_05	spectral_flatness_mean	mfcc_mean_01
mfcc_std_12	spectral_kurtosis_std	mfcc_mean_02
rms_std	spectral_skewness_mean	mfcc_mean_04
spectral_centroid_mean	spectral_spread_mean	mfcc_mean_05
spectral_kurtosis_mean	spectral_spread_std	mfcc_mean_06
temporal_spread_mean	temporal_kurtosis_std	mfcc_mean_09
	temporal_spread_mean	mfcc_std_04
Piano, Flute, Bass, Trumpet		temporal_flatness_std
attack_slope_mean	**Violin vs. Viola**	
flux_mean	event_density_mean	**French Horn vs. Trumpet**
mfcc_std_01	log_attack_time_std	mfcc_mean_01
mfcc_std_02	mfcc_mean_02	mfcc_mean_03
mfcc_std_03	mfcc_mean_04	mfcc_mean_04
mfcc_std_11	mfcc_mean_11	mfcc_mean_05
roughness_mean	mfcc_std_01	mfcc_mean_09
spectral_flatness_std	regularity_std	mfcc_std_08
spectral_spread_std	spectral_spread_mean	mfcc_std_09
temporal_centroid_mean	temporal_centroid_mean	spectral_kurtosis_std
temporal_centroid_std		temporal_centroid_mean
temporal_kurtosis_mean	**Oboe vs. Bassoon**	
temporal_spread_mean	mfcc_mean_01	**Vibraphone vs. Xylophone**
	mfcc_mean_02	temporal_flatness_mean
Acoustic vs. Electric Guitars	spectral_flatness_mean	temporal_flatness_std
brightness_std		temporal_centroid_std
flux_std	**Flute vs. Piccolo**	spectral_flatness_std
mfcc_std_03	mfcc_mean_01	temporal_spread_std
regularity_mean	mfcc_mean_02	temporal_spread_mean
spectral_spread_mean	spectral_flatness_mean	
zero_crossing_std	flux_mean	
	mfcc_std_02	

Table 3: List of selected features for each set of instruments using the SFS algorithm with a p value of 0.001

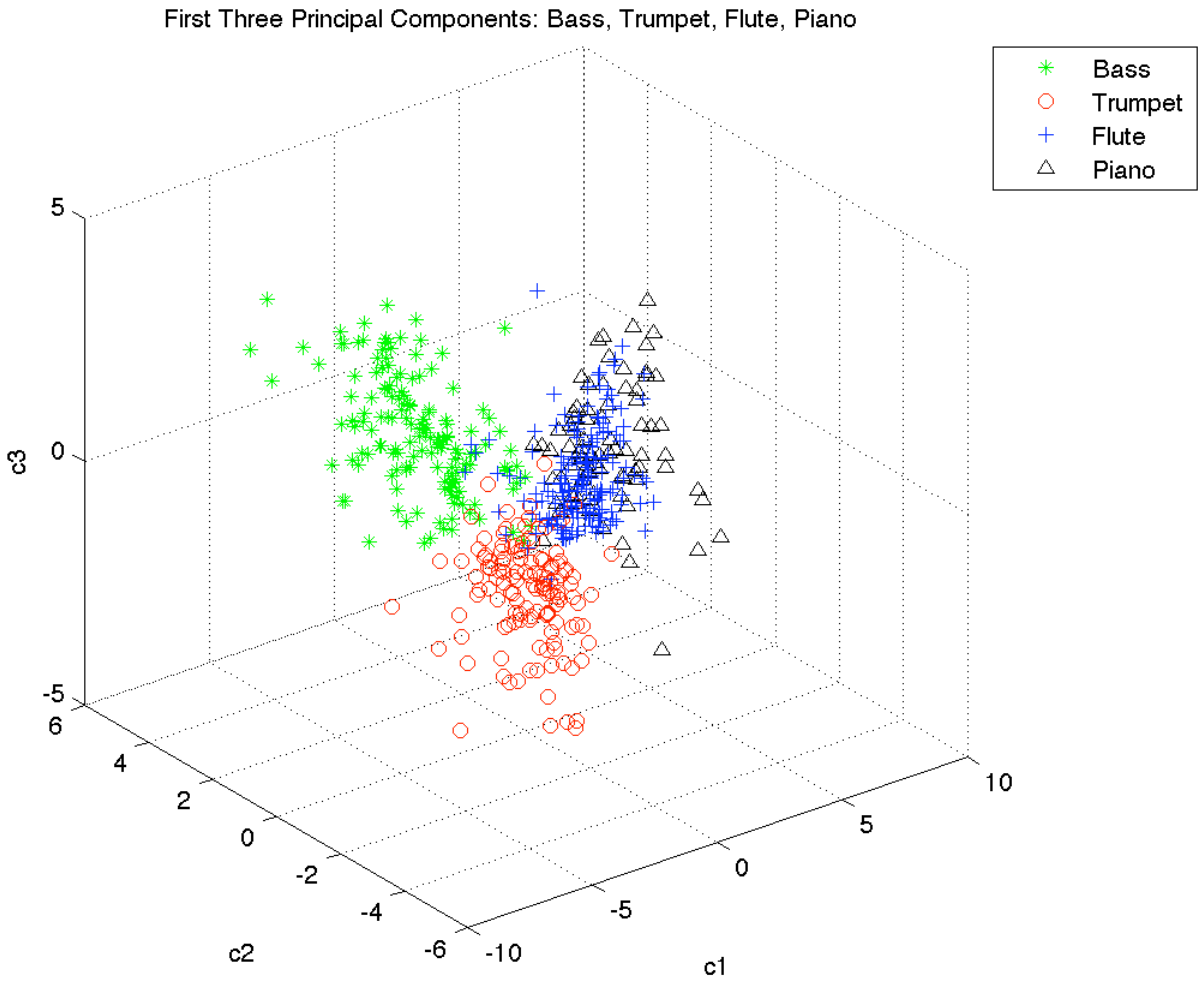

Figure 18: Group scatter plot of the first three canonical principal components of a piano, flute, bass and trumpet

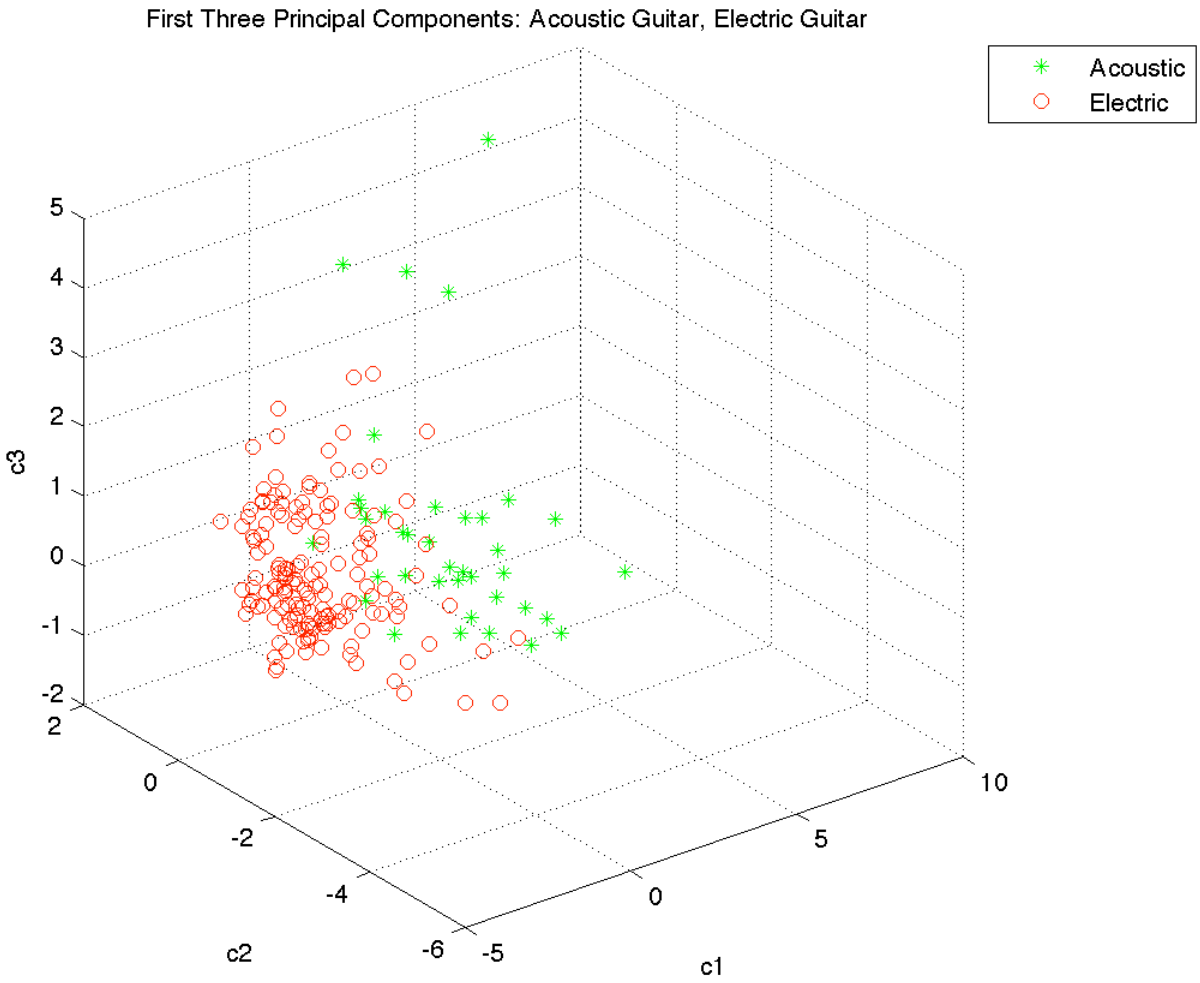

Figure 19: Group scatter plot of the first three canonical principal components of electric and acoustic guitars

The next set of similar sounding instruments is keyboards: a piano, organ, celesta and harpsichord. In this set, 959 audio samples were used: 223 pianos, 644 organs, 77 harpsichords and 15 celestas (Figure 20). The classification test resulted in a median of nearly 95% novel instruments classified correctly from 17 selected features.

Next, two very similar stringed instruments, a violin and viola were tested. In this case, 541 total audio samples were presented to the neural network: 373 violins and 168 violas (Figure 21). The testing revealed a median classification rate of nearly 80%, which is rather impressive given the striking similarity between violins and violas. The SFS algorithm selected nine features.

The next family of instruments tested was the double reed woodwinds. For this test, 76 total samples of an oboe and bassoon were used: 27 oboes and 49 bassoons (Figure 22). The neural network was able to train effectively on these instruments, resulting in a median classification rate of 100% correct. It should be noted that there were only four runs (out of 20) not with 100% classification accuracy on three selected features.

The next set of instruments comes from the single reed woodwind family. In this case, 111 audio samples of saxophones were tested with: 57 alto and 54 tenor (Figure 23). The results indicate a median percent classified of roughly 91% accuracy from seven selected features.

The final set of instruments from the woodwind family contains no reeds at all. In this set, 276 audio samples were used: 218 flutes and 58 piccolos (Figure 24). The results show a median classification rate of nearly 98% accuracy from five selected features, even with two very similar sounding instruments.

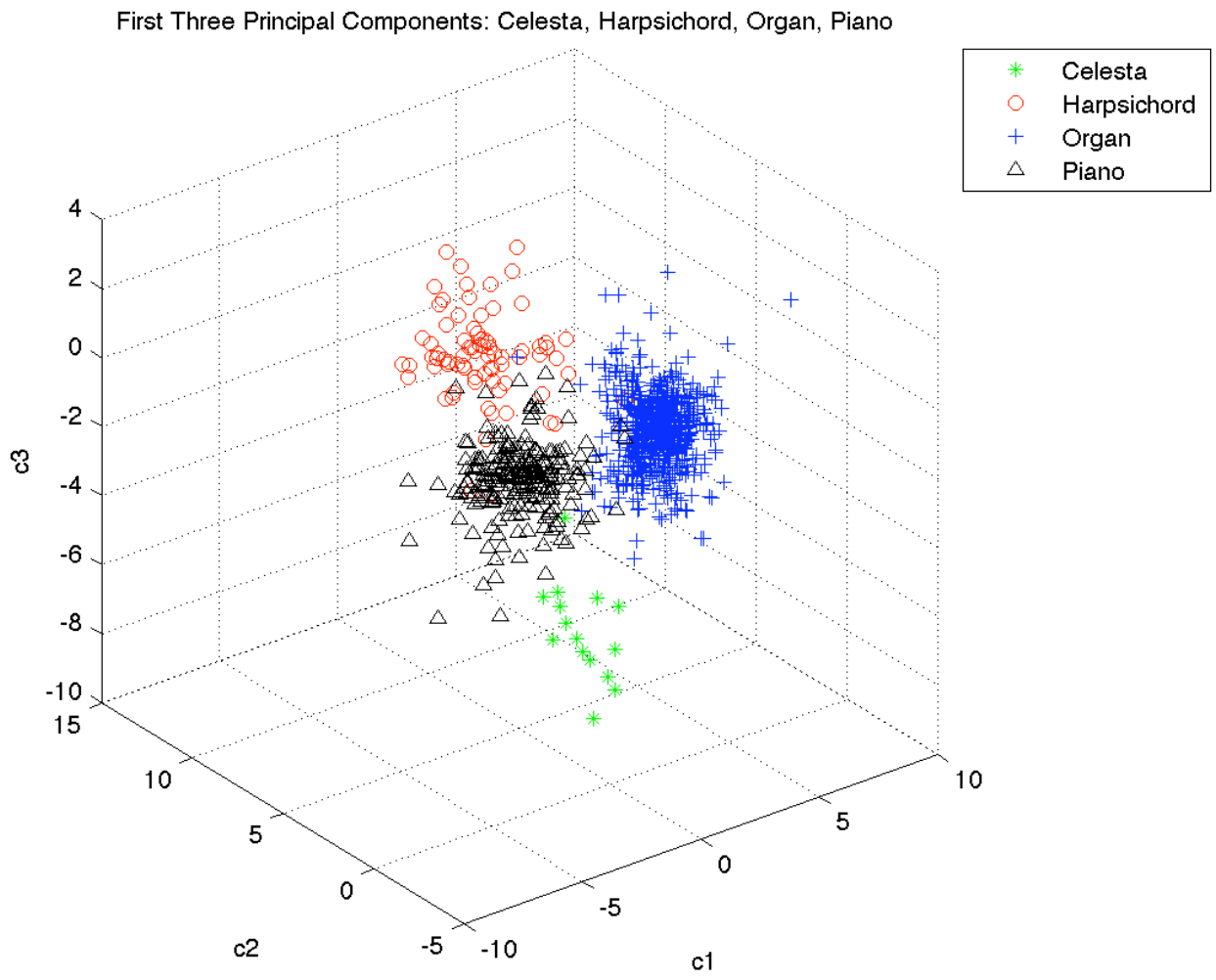

Figure 20: Group scatter plot of the first three canonical principal components of celesta, harpsichord, organ and piano

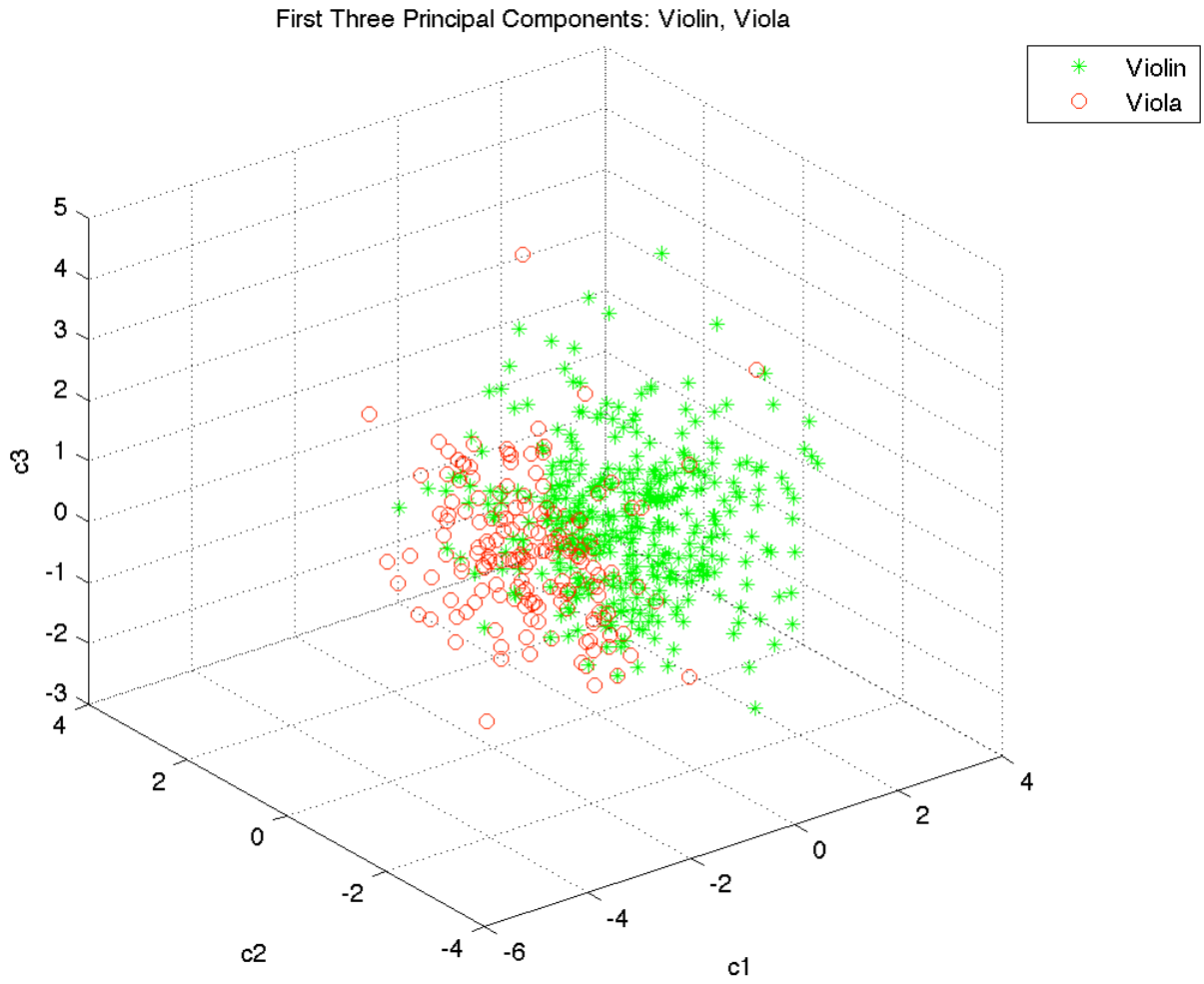

Figure 21: Group scatter plot of the first three canonical principal components of violins vs. violas

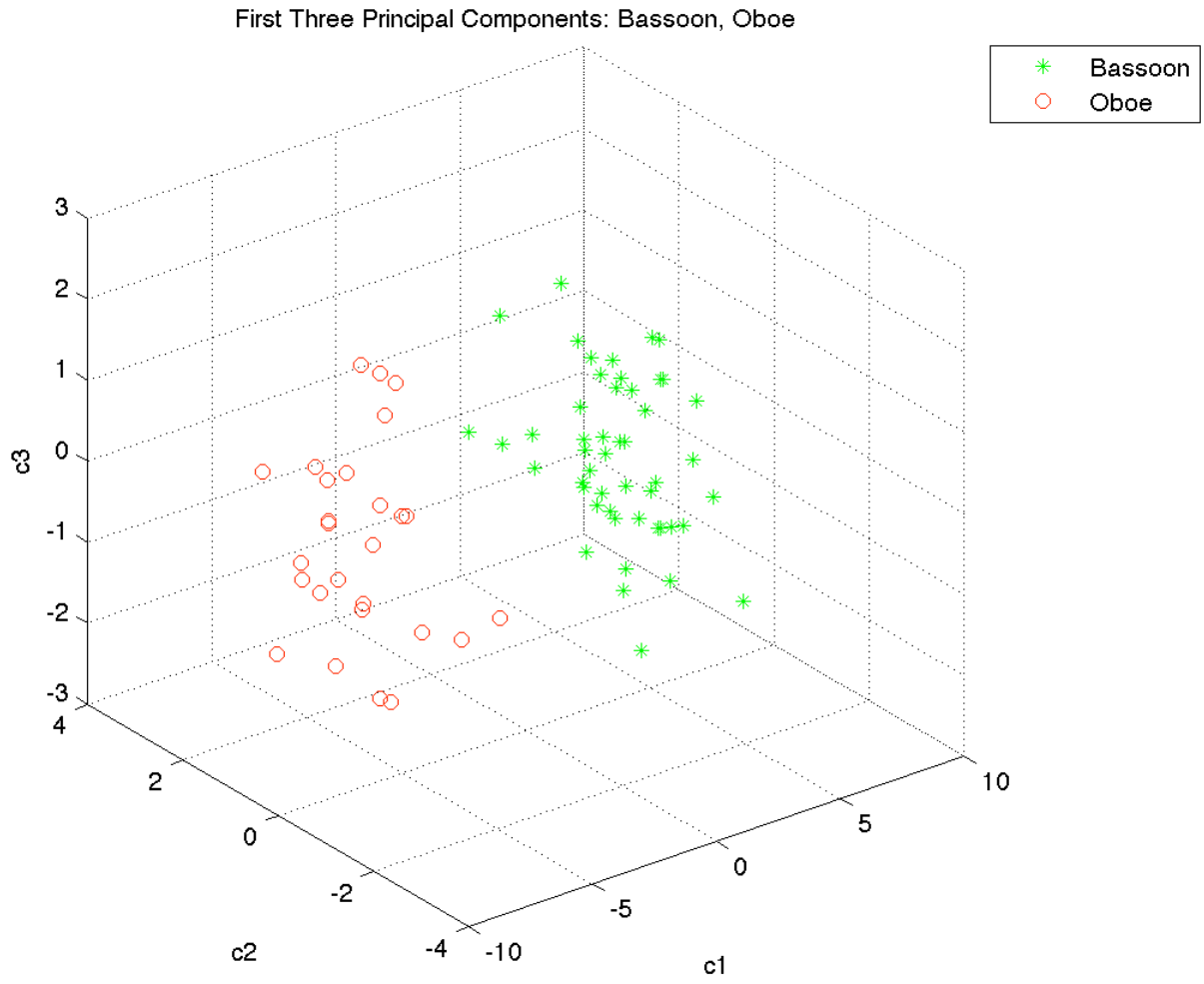

Figure 22: Group scatter plot of the first three canonical principal components of bassoons and oboes

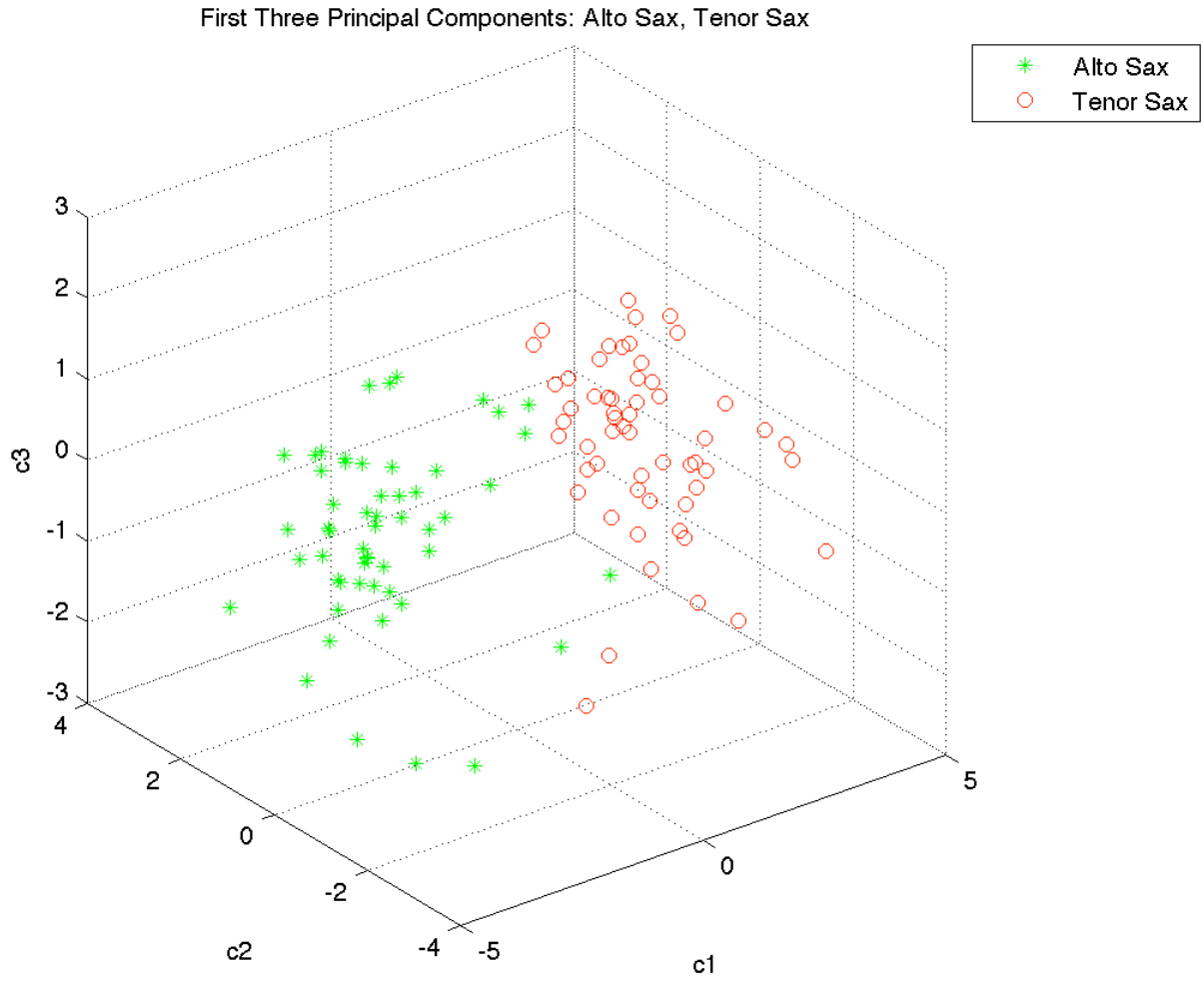

Figure 23: Group scatter plot of the first three canonical principal components of alto and tenor saxophones

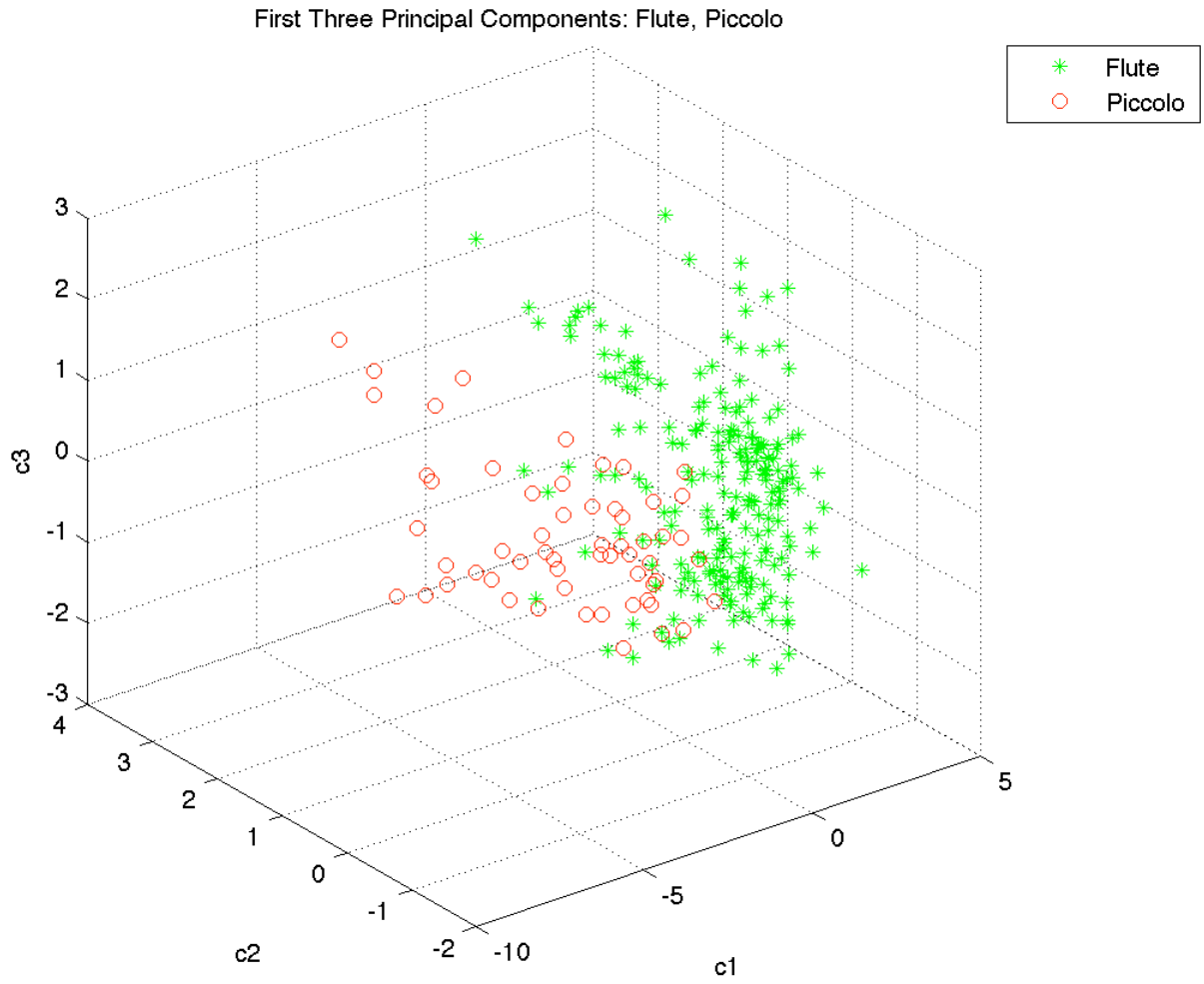

Figure 24: Group scatter plot of the first three canonical principal components of flutes and piccolos

Moving on to the brass instrument family, two sets of instruments were tested: first, a trumpet vs. a trombone and then a trumpet vs. a French horn. In the first set, 241 audio samples of trumpets and trombones were presented, with 133 trumpets and 108 trombones (Figure 25). The results were outstanding, with a median classification rate of 94% from nine selected features.

The second set of brass instruments tested were trumpets vs. French horns. In this test, 201 total audio samples were used: 133 trumpets and 68 French horns (Figure 26). The results indicate a median classification accuracy of about 96% from nine selected features.

The final set of similar instruments tested was from the percussion family. In this case, 101 audio samples were used: 89 vibraphones and 12 xylophones (Figure 27). The median classification rate in the case was nearly 89% on six selected features. It is interesting to note that even though the principal component analysis revealed separable instrument classes, the neural network still performed well below perfection.

4.1 Analysis of Results

4.1.1 Breakdown of Classifications

The classification results were broken down further by comparing the expected class versus the neural network's output in more detail (Table 4) over all ten runs. Such breakdown was computed to provide more insight as to how each set of trained neural networks performed. The columns of the table are as follows: "c1, c1" and "c2, c2" indicate the percentage classified correctly: i.e., the network was given an audio sample of class 1 (or 2) as input and the neural network correctly identified the sample. The columns labeled "c1, c2" and "c2, c1" represent the

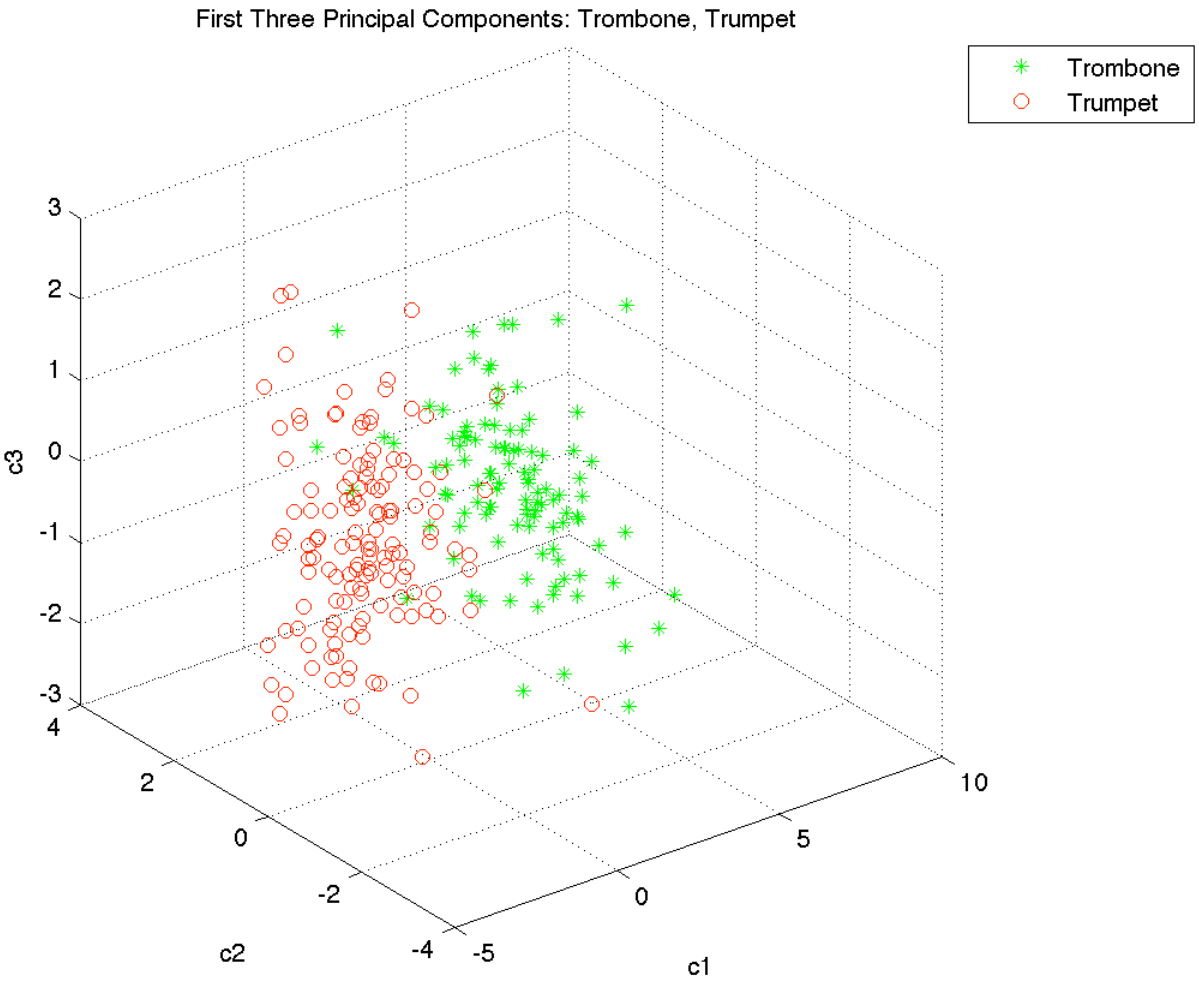

Figure 25: Group scatter plot of the first three canonical principal components of trombones and trumpets

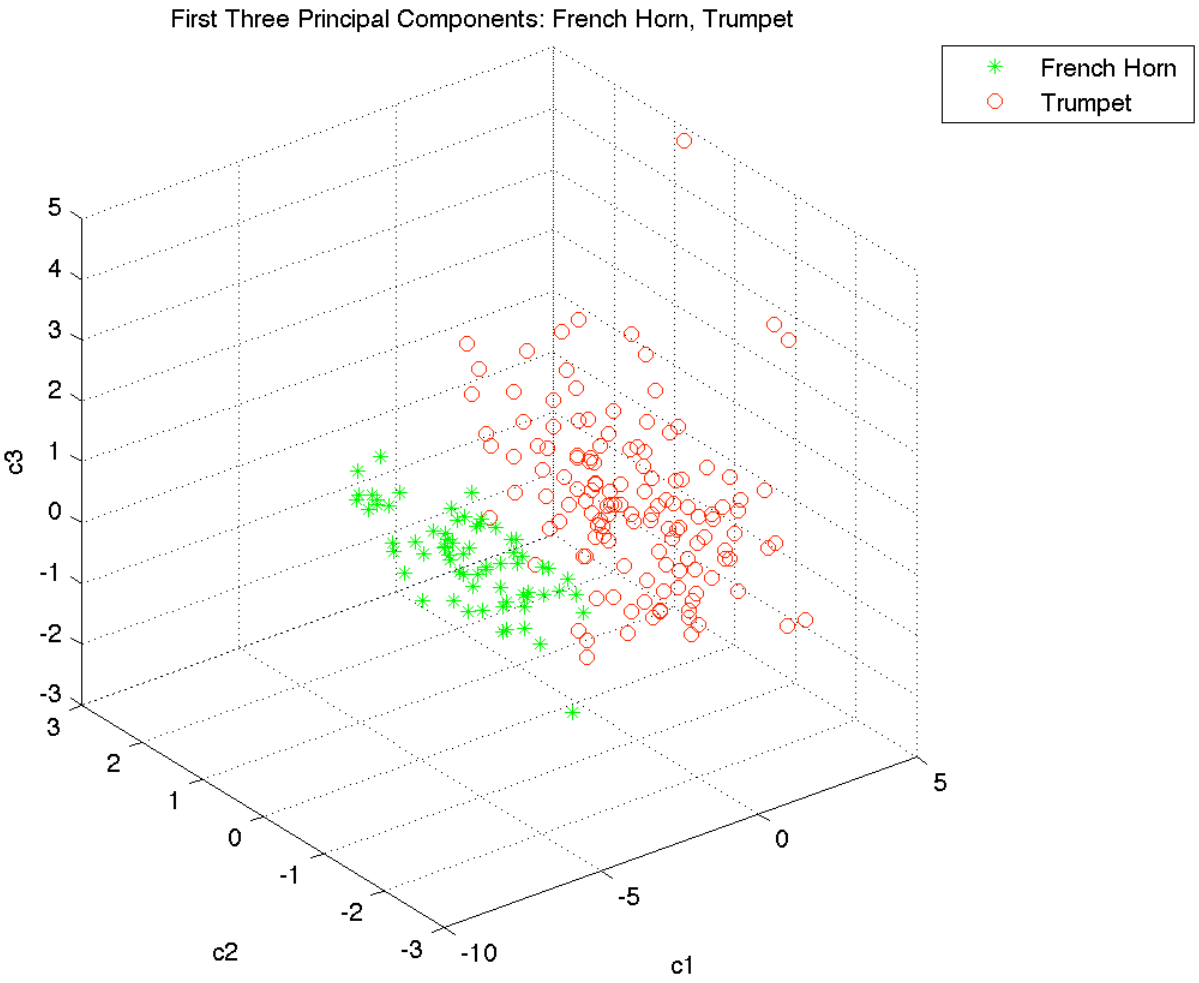

Figure 26: Group scatter plot of the first three canonical principal components of French horns and trumpets

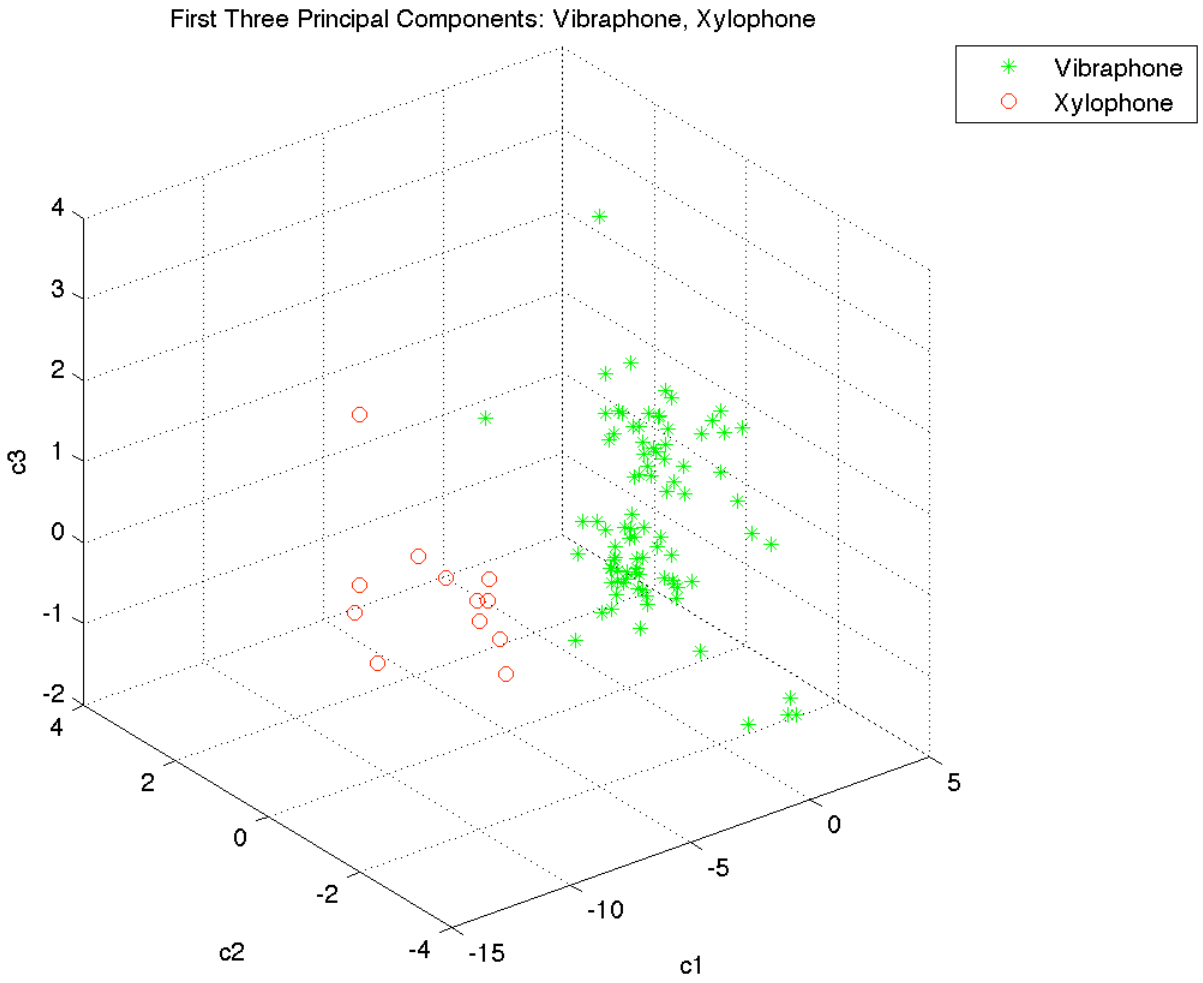

Figure 27: Group scatter plot of the first three canonical principal components of vibraphones and xylophones

	Classes		Percentage of Output in Each Set (expected result, neural network output)						
Experiment	c1	c2	c1, c1	c1, c2	c1, no class	c2, c2	c2, c1	c2, no class	total correct
Bass vs. Trumpet	Bass	Trumpet	87.226	2.000	10.774	85.113	3.308	11.579	86.250
Acoustic vs. Electric Guitar	Acoustic	Electric	**58.158**	**20.263**	**21.579**	97.516	1.147	1.338	89.846
Violin vs. Viola	Violin	Viola	88.204	10.295	1.501	**54.524**	**43.393**	**2.083**	77.745
Oboe vs. Bassoon	Oboe	Bassoon	99.796	0.000	0.204	89.630	10.000	0.370	96.184
Alto vs. Tenor Saxophone	Alto	Tenor	80.185	7.407	12.407	84.035	5.263	10.702	82.162
Flute vs. Piccolo	Flute	Piccolo	97.890	1.239	0.872	91.552	5.000	3.448	96.558
Trombone vs. Trumpet	Trombone	Trumpet	84.630	4.537	10.833	92.481	5.489	2.030	88.963
French Horn vs. Trumpet	French horn	Trumpet	84.412	13.529	2.059	96.165	2.632	1.203	92.189
Vibraphone vs. Xylophone	Vibraphone	Xylophone	99.101	0.225	0.674	**24.167**	**71.667**	**4.167**	90.198

Table 4: Breakdown of all 2-class classification experiments.

cases where the neural network was presented with an audio sample of class 1 (or 2) and the neural network incorrectly identified the instrument as the other class (i.e., class 2 and 1 respectively). The remaining columns (i.e., "c1, no class", "c2, no class") correspond to the instances where the neural network could not classify the audio sample as either instrument: a total mis-classification. The "total percent classified correctly" indicates the total number of instruments correctly classified over all 20 runs of each set.

The results show that given the nine sets of 2-class experiments, the trained neural networks seemed to have the most difficulty identifying the acoustic guitar, viola and xylophone. It is interesting to point out that, although over 70% of the xylophones presented to the trained neural network were incorrectly classified as vibraphones, less than 1% of the vibraphones were incorrectly classified as xylophones. However, the high error rate of the xylophone recognition did not seem to drastically affect the overall performance of that instrument set. This was a common theme with violas and acoustic guitars as well, with one class being more often recognized than the other (i.e., electric vs. acoustic, violin vs. viola).

Perhaps the misclassifications of the above mentioned instruments occurred due to overtraining the neural network to identify the more prevalent instrument (i.e., vibraphone, violin and electric guitar). The xylophones contributed only 11% of the total number of audio samples for that set, the violas were 30% of its samples and the acoustic guitars were roughly 20% of the samples. It is interesting to note that in the case of flute vs. piccolo, with piccolo containing only 21% of the samples, the piccolo was still identified correctly over 91% of the time. In future tests, scaling the network update inversely proportional to the sample size will help reduce the effect of biased training.

One could speculate that the misclassifications of the previously mentioned instruments were due to the striking perceptual similarities between the instruments in question. Even though the sequential forward selection algorithm attempts to select the features that best differentiate the classes, perhaps it is the case that the instruments sound so similar that even a human cannot out-perform this computerized classification paradigm. It would be interesting to test humans on the above instruments to see how it would compare.

4.1.2 Statistical Analysis

In attempts to further understand the results and better predict future tests, an extended multivariate analysis of variance (MANOVA) was performed on each set of instruments and correlated with the original classification results. The outcome of this test is summarized in

The columns of are as follows. In the results in Table 5, the "SFS count" is simply the number of features selected for each set of instruments using the sequential forward selection algorithm with a p of 0.001. The next column in, "Within", represents the trace of the within-groups sum of squares divided by the total within-group degrees of freedom; a measure that

Instruments	SFS count	Within	Between	Lambda	Chisq	mdist (mean)	mdist (min)	mdist (max)	gmdist
Bass vs. Trumpet	16.00	0.78	122.88	0.07	46.43	15.89	1.87	63.51	53.84
Piano, Flute, Bass, Trumpet	13.00	0.16	11.59	0.21	29.65	12.91	1.23	453.78	34.97
Electric vs. Acoustic Guitar	6.00	0.12	2.39	0.29	38.99	5.94	0.32	66.94	15.30
Piano, Harpsichord, Celesta, Organ	17.00	0.30	20.57	0.27	40.27	16.93	2.14	256.93	76.02
Violin vs. Viola	9.00	0.64	27.40	0.58	32.70	8.97	1.01	41.34	3.42
Oboe vs. Bassoon	3.00	0.10	48.90	0.04	78.90	2.92	0.18	16.85	107.02
Alto vs. Tenor Saxophone	7.00	3.92	30.45	0.12	31.66	6.87	1.04	27.20	28.19
Flute vs. Piccolo	5.00	0.83	39.17	0.23	80.08	4.96	0.82	31.54	20.16
Trombone vs. Trumpet	9.00	1.25	39.21	0.16	47.61	8.93	1.50	46.43	20.92
French Horn vs. Trumpet	9.00	0.76	45.22	0.14	42.01	8.91	0.40	71.25	26.47
Vibraphone vs. Xylophone	6.00	0.11	1.36	0.10	36.16	5.88	0.48	78.31	80.34
Correlation	SFS count	Within	Between	Lambda	Chisq	mdist (mean)	mdist (min)	mdist (max)	gmdist
mean	-0.37	-0.22	0.12	-0.43	0.71	-0.37	-0.30	-0.51	0.39
median	-0.21	0.05	0.41	-0.53	0.65	-0.21	-0.11	-0.58	0.33
std	0.34	0.62	-0.19	-0.47	-0.26	0.33	0.37	-0.05	0.04
min	-0.40	-0.65	-0.60	0.35	0.31	-0.39	-0.44	0.03	0.01
max	-0.21	0.14	0.33	-0.85	0.47	-0.22	-0.15	-0.40	0.45

Table 5: Results of MANOVA and correlation with classification statistics

represents the similarity of each group's data. The "Between" column is the trace of the between-group sum of squares divided by the total between-group degrees of freedom; a measure that represents that difference between groups. The next column, "Lambda", is Wilk's lambda test statistic that can be used to determine if there are differences between group means on a combination of dependent variables. Following that is "Chisq", which was computed as the transformation of Lambda to an approximate Chi-Square distribution (MATLAB 7.7). The next three columns are the mean, minimum and maximum Mahalanobis distances (i.e., "mdist") from each point to the mean of its group. The last column (i.e., "gmdist") is the Mahalanobis distances between each pair of group means.

Although correlation does not imply causality, a few correlations are worth mentioning because they seem to highlight how and why certain datasets performed better than others. The strongest correlation found was between the calculated Lambda value and the max classification rate, with a strong negative correlation between the two. This result suggests that, as the Lambda value gets smaller, the maximum percent classified correctly gets larger. Such an occurrence is not too surprising, seeing that Wilk's lambda measures group separability. In the same vein, another strong correlation was found between the mean classification rate and the Chi-squared test. In future studies, perhaps the dataset in question can be analyzed using Wilk's lambda and the resultant Chi-square approximation to determine if an appropriate classification is possible.

The within-class sum of squares was moderately correlated (negatively) with the minimum classification rate; suggesting that the more separable that data is within group, the lower the worst score. This seems intuitive: the more voicings an instrument has, the harder time a neural network will have identifying the instrument. With regard to between-class sum-of squares, there was a smaller positive correlation between it and the median percent classified, suggesting

that as the classes of instruments become more separable, the better than neural network trained and performed.

Lastly, an interesting result is that the number of features selected by the SFS algorithm was moderately correlated (negatively) with the mean classification rate. This suggests that as the number of features selected by the SFS algorithm increases, the ability for the neural network to train and classify the instruments decreases. Perhaps this is because more features leads to noisier data, which may make it harder for the neural network to train effectively.

4.1.3 A Look at Selected Features

To better determine which features out of the original 70 were the most important, the total number or count of the features selected is noted in Table 6. Future statistical directions would include testing combinations of selected features to determine which features really are the most salient. It is interesting to note the features that were not selected (i.e., Count = 0) in the experiments. For example, the mean 2nd band of the MFCC was the most selected feature, while the means of the 10th and 12th Mel-Bands were not used at all. It is also interesting to point out that pitch (i.e., note) was not selected for in any experiment, even though pitch is a straightforward way for humans to differentiate instruments (e.g., basses have low pitches while trumpets have much higher notes). One last feature not used in any experiment was the mean and standard deviation of the inharmonicity, which was used in almost every experiment consulted for this thesis.

4.1.4 Further Testing On Larger Sets

In attempts to extend the previously described experiment, 670 audio samples of 7 different Woodwind instruments (i.e., bassoon, clarinet, flute, piccolo, oboe, English horn and

Feature	Count		Feature	Count
mfcc_mean_2	6		mfcc_std_4	1
mfcc_mean_1	5		mfcc_std_5	1
mfcc_mean_4	4		mfcc_std_8	1
mfcc_std_1	4		mfcc_std_9	1
temporal_spread_mean	4		mfcc_std_11	1
flux_mean	3		mfcc_std_12	1
mfcc_mean_5	3		rolloff_mean	1
mfcc_std_2	3		rolloff_std	1
mfcc_std_3	3		roughness_mean	1
spectral_flatness_mean	3		spectral_centroid_mean	1
spectral_spread_mean	3		spectral_centroid_std	1
temporal_centroid_mean	3		spectral_kurtosis_mean	1
event_density_mean	2		spectral_skewness_mean	1
flux_std	2		temporal_flatness_mean	1
mfcc_mean_6	2		temporal_kurtosis_mean	1
mfcc_mean_8	2		temporal_kurtosis_std	1
mfcc_mean_9	2		temporal_spread_std	1
mfcc_mean_11	2		zero_crossing_std	1
regularity_mean	2		attack_slope_std	0
regularity_std	2		event_desity_std	0
rms_std	2		inharm_mean	0
roughness_std	2		inharm_std	0
spectral_flatness_std	2		mfcc_mean_10	0
spectral_kurtosis_std	2		mfcc_mean_12	0
spectral_spread_std	2		mfcc_std_6	0
temporal_centroid_std	2		mfcc_std_7	0
temporal_flatness_std	2		mfcc_std_10	0
attack_slope_mean	1		mfcc_std_13	0
brighness_mean	1		rms_mean	0
brightness_std	1		spectral_skewness_std	0
log_attack_time_mean	1		temporal_skewness_mean	0
log_attack_time_std	1		temporal_skewness_std	0
mfcc_mean_3	1		zero_crossings_total	0
mfcc_mean_7	1		zero_crossings_mean	0
mfcc_mean_13	1		pitch	0

Table 6: Total count of selected features through all experiments

saxophone) were trained and tested with the threaded neural network. The results were mediocre, with a mean classification rate of just 52% from 12 selected features. In a similar vein, 927 audio samples of 4 String instruments (i.e., bass, cello, viola and violin) were trained and tested on, resulting in a mean classification rate of just above 43% on 6 selected features.

Although the system performed above chance when classifying the Woodwinds and Strings, it is evident that much more work needs to be done to achieve a more universal classifying system. In particular, perhaps more features could be extracted, a better feature selection algorithm could be used or a different classifying paradigm could be employed. All in all, the multi-threaded neural network paired with the sequential forward selection algorithm provides a quick and powerful way to classify limited sets of perceptively similar sounding instruments.

Chapter 5: Conclusion and Future Direction

In the previous experiment, the first step of automatic music analysis, i.e., monophonic instrument recognition, was explored. First a feed-forward, backpropagation neural network was sped up by modifying the implementation of a batch mode training method to support threads and multi-core technology. Next, 11 sets of instruments were trained and tested using the parallelized neural network; starting with two sets of perceptively different sounding instruments and leading to subjectively similar sounding ones. With each set of instruments, 70 features were initially extracted and then paired down using the sequential forward selection algorithm. The results were promising, showing that a neural network can be used to classify similar sounding instruments.

Such a tool could be useful in the following way. Suppose there's a musician who wants to transcribe a musical piece but is having trouble telling the difference between instruments during a solo phase (e.g., a saxophone solo in a jazz piece). Instead of spending time researching what instruments the particular band used in the recording, the musician could use an audio segmentation tool (Figure 28) and feed the segment to an "expert" neural network that can differentiate the instrument in question (e.g., alto from tenor saxophones). Although limited in scope, such a system represents the proverbial "tip of the iceberg" of automated music analysis.

As for future research directions of monophonic instrument recognition, with the sheer number of instruments and voicings out there, a logical next step would include training and testing many more combinations with the system described above. Another avenue to explore would include the feature selection phase, where several other feature selection and clustering methods could be substituted and tested (e.g., Principal Component Analysis, Non-Negative Matrix Factorization, Sequential Backward Selection, etc.,). In addition to pair-wise training and

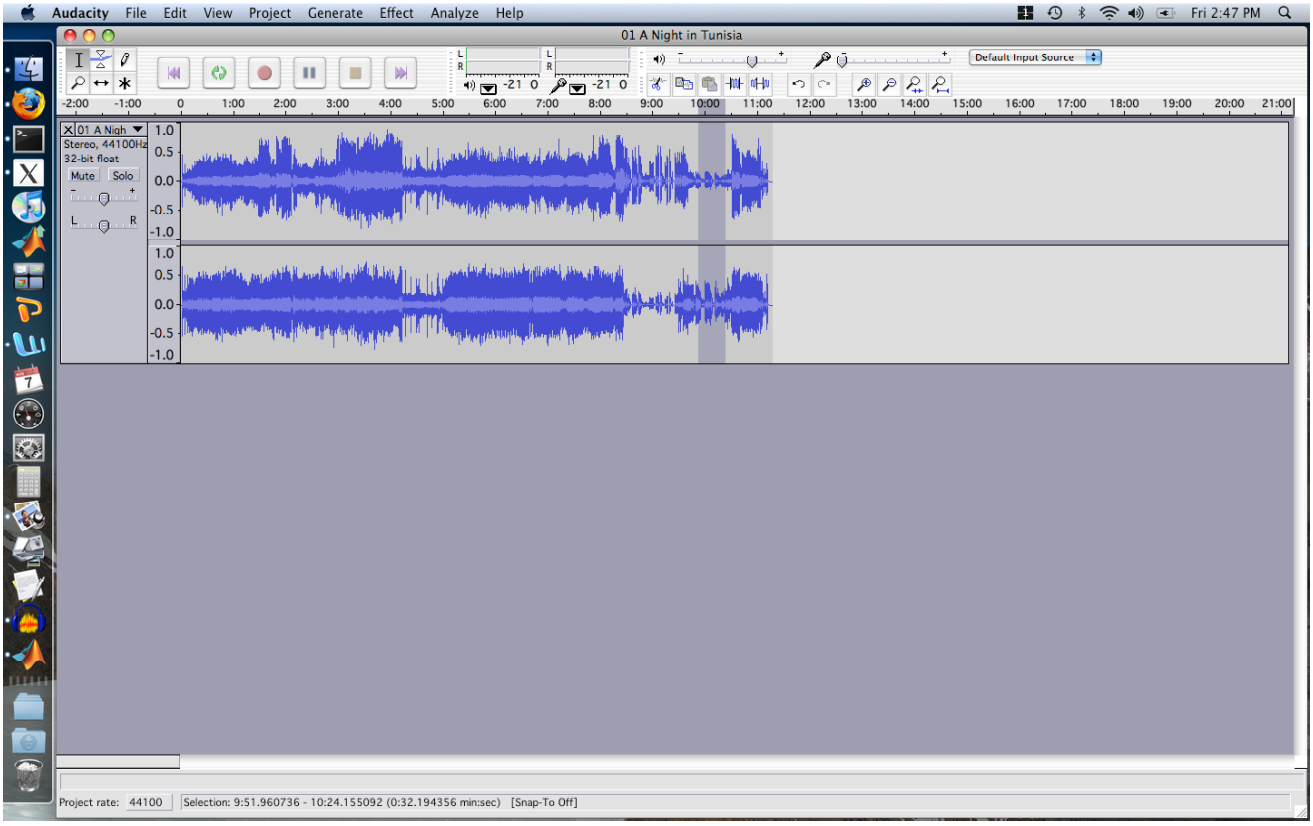

Figure 28: Screen shot showing how a musician could extract of segment of audio that could be tested on an expert neural network

testing, another direction would be to attempt to build a “universal” classifier that is capable of identifying any instrument presented to it. Such a system could be implemented as a “panel of experts,” where several smaller neural networks are all presented with the same data and the “best” result is used.

References

Agostini, G., Longari, M., & Pollastri, E. (2003). Musical instrument timbres classification with spectral features. *EURASIP Journal on Applied Signal Processing*, *2003*, 5-17.

Ahmad, A., Zulianto, A., & Sanjaya, E. (1999). Design and Implementation of Parallel Batch-mode Neural Network on Parallel Virtual Machine. Proceedings from the Industrial Electronic Seminar. Graha Institut Teknologi Supuluh Nopember, Surabaya.

Analysis & Machine Learning Research Group. (1999). FSBOX. Retrieved July 11, 2009, from http://www.robots.ox.ac.uk/~parg/software.html

Chafe, C. & Jaffe, D. (1986). Source Separation and Note Identification in Polyphonic Music. Proceedins from the IEEE Conf. Acoust. Sp. and Sig. Proc., Tokyo, Japan.

Cormican, B. (1991). *Mozart's Death - Mozart's Requiem: An Investigation*. Portland: Amadeus Press.

Deng, J., Simmermacher, C., & Cranefield, S. (2008). A Study on Feature Analysis for Musical Instrument Classification. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *38*(*2*), 429-438.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification (2nd Edition)*. New York: Wiley-Interscience.

Eronen, A., & Klapuri, A. (2000). Musical instrument recognition using cepstral coefficients and temporal features. *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, *2*, II753-II756 vol.2.

Farmer, H. G. (1988). *Historical Facts for the Arabian Musical Influence*. New York: Beaufort Books.

Fedorova, N., & Terekhoff, S. (1999). Parallel MPI implementation of training algorithms for Medium-Size Feed-Forward Neural Networks. Neural Networks, IJCNN '99. International Joint Conference on, 4, 2378-2379. Washington, DC.

Herrera-Boyer, P., Peeters, G., & Dubnov, S. (2003). Automatic Classification of Musical Instrument Sounds. *Journal of New Music Research*, *32*(*1*), 3-21.

Janata, P. (2009). The Neural Architecture of Music-Evoked Autobiographical Memories. Cerebral Cortex. http://cercor.oxfordjournals.org/cgi/content/full/bhp008?ijkey=odjcdhE8j4ugMRU&keytype=ref1

Janata, P., Tomic, S., & Rakowski, S. (2007). Characterisation of music-evoked autobiographical memories. *Memory*, *15*(*8*), 845-860.

Kitahara, T., Goto, M., Komatani, K., Ogata, T., & Okuno, H. (2005). Instrument Identification

in Polyphonic Music: Feature Weighting with Mixed Sounds, Pitch-dependent Timbre Modeling, and Use of Musical Context. Proceedings from the Int'l Conf. Music Information Retrieval (ISMIR), London, UK.

Klapuri, A. & Davy, M. (2006). *Signal Processing Methods for Music Transcription*. New York: Springer.

Koutroumbas, K., & Theodoridis, S. (1999). *Pattern Recognition*. Toronto: Academic Press.

Lartillot, O. (2009). MIRtoolbox 1.2 User's Manual, Finnish Centre of Excellence in Interdisciplinary Music Research, University of Jyväskylä, Finland.

Lartillot, O., & Toiviainen, P. (2007). A Matlab Toolbox for Musical Feature Extraction From Audio. Proceedings from the International Conference on Digital Audio Effects. Bordeaux, France.

Lu, L., Liu, D., & Zhang, H. (2006). Automatic mood detection and tracking of music audio signals. *Audio, Speech, and Language Processing, IEEE Transactions on*, *14*(*1*), 5-18.

Opolko, F., & Wapnick, J. (1991). McGill University Master Samples. Montreal, Quebec, Canada.

Peeters (2004). A large set of audio features for sound description (similarity and classification)

in the cuidado project. Technical report, CUIDADO I.S.T. Project Report. http://recherche.ircam.fr/equipes/analyse-synthese/peeters/ARTICLES/Peeters_2003_cuidadoaudiofeatures.pdf

Presta, A. (2005). NNF - Neural Net Framework. Retrieved June 2, 2009, from http://nnf.sourceforge.net/

Salembier, P. (2002). *Introduction to MPEG 7: Multimedia Content Description Language*. New York, NY: Wiley.

Sethares, W. A. (2004). *Tuning, Timbre, Spectrum, Scale*. New York: Springer.

Sola, J., & Sevilla, J. (1997). Importance of input data normalization for the application of neural networks to complex industrial problems. *Nuclear Science, IEEE Transactions on*, *44*(*3*), 1464 - 1468.

Terasawa, H., Slaney, M., & Berger, J. (2005). The thirteen colors of timbre. *Applications of Signal Processing to Audio and Acoustics, 2005. IEEE Workshop on*, *N/A*, 323-326.

Tsaregorodtsev, V.G. (2005). Parallel implementation of back-propagation neural network software on SMP computers. Proceedings from Parallel Computing Technologies, 8th International Conference, PaCT 2005, Krasnoyarsk, Russia, September 5-9, 186-192.

Tzanetakis, G., Essl, G., & Cook, P. (2001). Automatic Musical Genre Classification of Audio Signals. Proceedings from International Symposium on Music Information Retrieval (ISMIR), Bloomington Indiana.

Turchenko, V. & Grandinetti, L. (2009). Efficiency analysis of parallel batch pattern NN training algorithm on general-purpose supercomputer. *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, 5518/2009, 223-226*.

Vonnegut, K. (2007). *A Man Without a Country*. New York: Random House Trade Paperbacks.

Yoon, H., Nang, J.H. & Maeng, S.R. (1990). A distributed backpropagation algorithm of neural networks on distributed-memory multiprocessors. *Proceedings, 3rd Symposium on the Frontiers of Massively Parallel Computation,* 358-363.

Zhang, X. & Ras, Z.W. (2007a). Analysis of Sound Features for Music Timbre Recognition. Proceedings from the IEEE CS International Conference on Multimedia and Ubiquitous Engineering, Seoul, Korea.

Zhang, X. & Ras, Z.W. (2007b). Discriminant feature analysis for music timbre recognition. Proceedings from the ECML/PKDD Third International Workshop on Mining Complex Data (MCD), University of Warsaw, Poland.

Appendices

Appendix A: Digital Signal Processing of the Audio Signal: Common Techniques

To better understand the previous literature and presented experiment involving single instrument classification, a brief background of audio signal processing and a glossary of terms (Appendix B) is presented. Typically, features from the audio signal used in classifications experiments are extracted from the frequency and time domain. Please see Klapuri and Davy's (2006) excellent and thorough book on music signal processing for the complete details of the mathematics and calculations involved.

A.1 Spectral Features:

To transform an audio signal from the time to frequency domain, the Fourier transform is often used (Klapuri & Davy, 2006). Figure 29 shows the result of taking the discrete Fourier transform on the synthesize A 440 audio signal (Figure 30). The discrete Fourier transform of the audio signal is one of the primary first steps in music analysis.

Since the music signal can vary significantly over time (e.g., a 12-bar blues chord progression), a windowing technique of the audio signal has been widely accepted throughout the literature (Klapuri & Davy, 2006). In this model, the raw audio signal (Figure 31) is divided up into 20-100 ms windows or frames (Figure 32) where each subsequent frame overlaps typically 50% with the previous window. Usually a Hamming (Figure 33), Gaussian or Hanning window is applied to each frame of the raw audio signal, and the Fourier spectrum is then computed for each individual frame to create a spectrogram (Figure 34 and Figure 35) of the entire signal. As an important aside, the lowest frequency that can thus be computed within, for example, each 50ms frame is 20 Hz (1 cycle per 0.0050 seconds = 20 cycles/second or 20Hz).

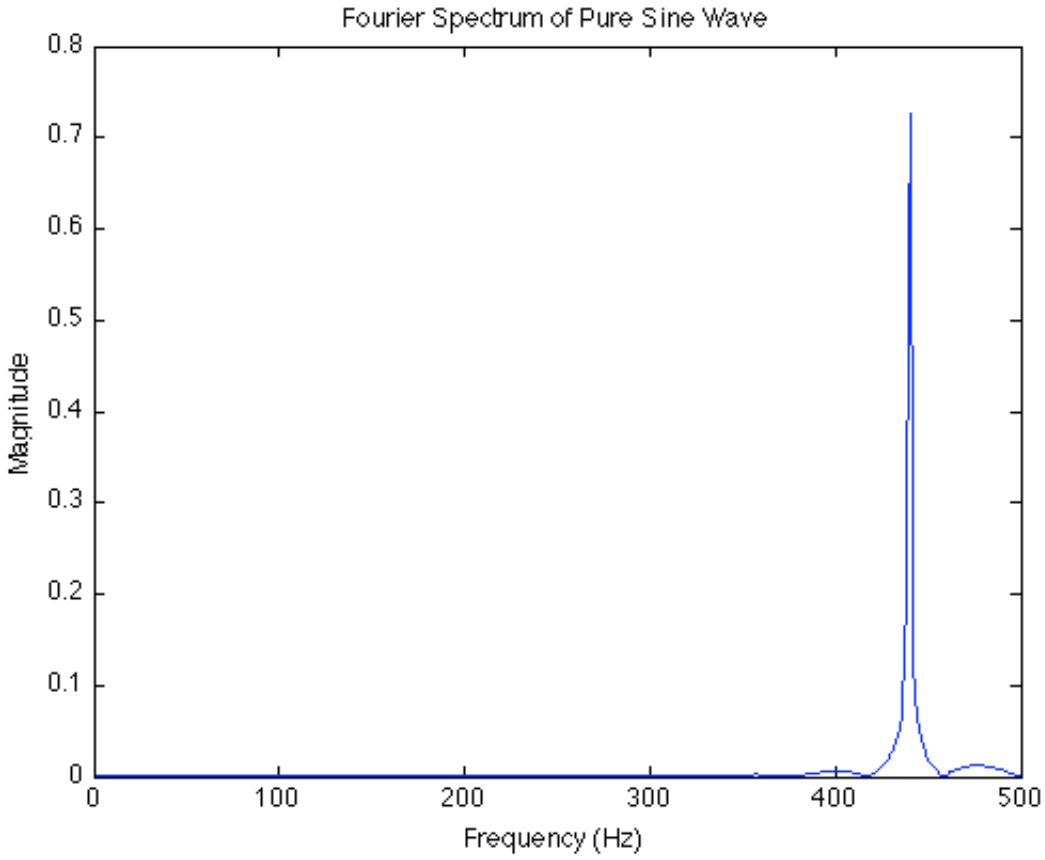

Figure 29: Discrete Fourier Transform of the synthesized A 440 sine wave for the entire signal

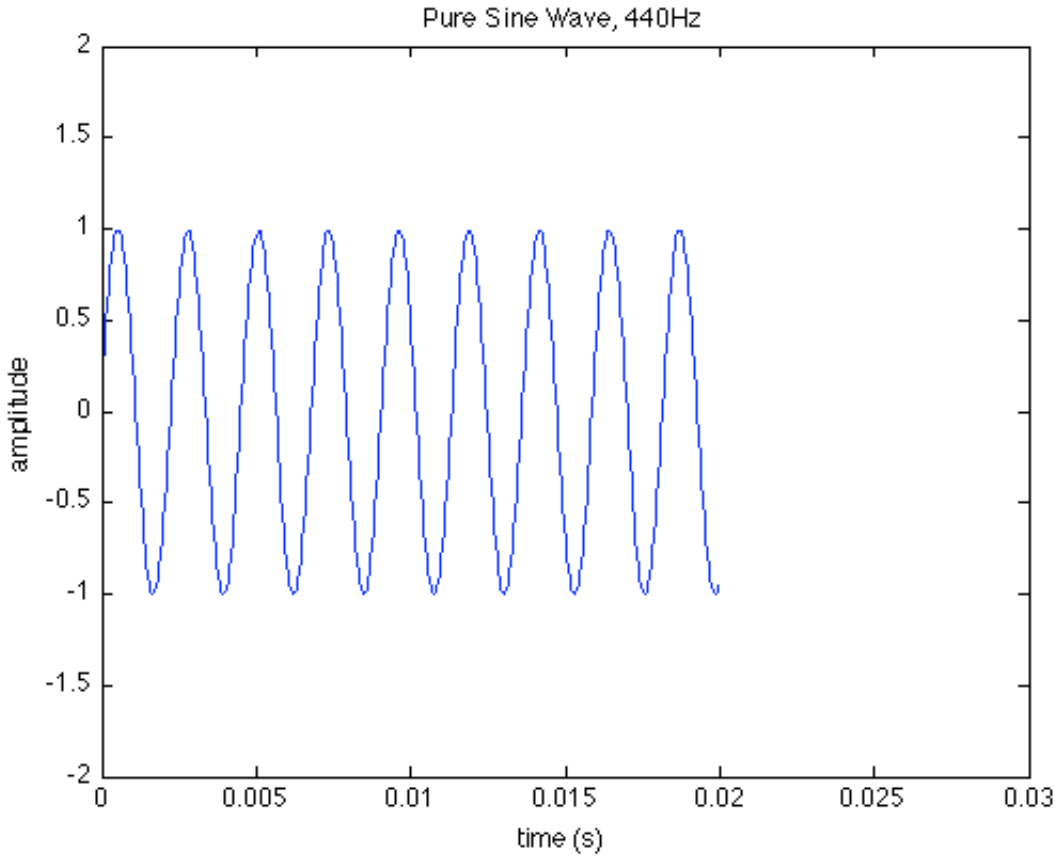

Figure 30: An artificially generated sine wave with a frequency of 440 Hz

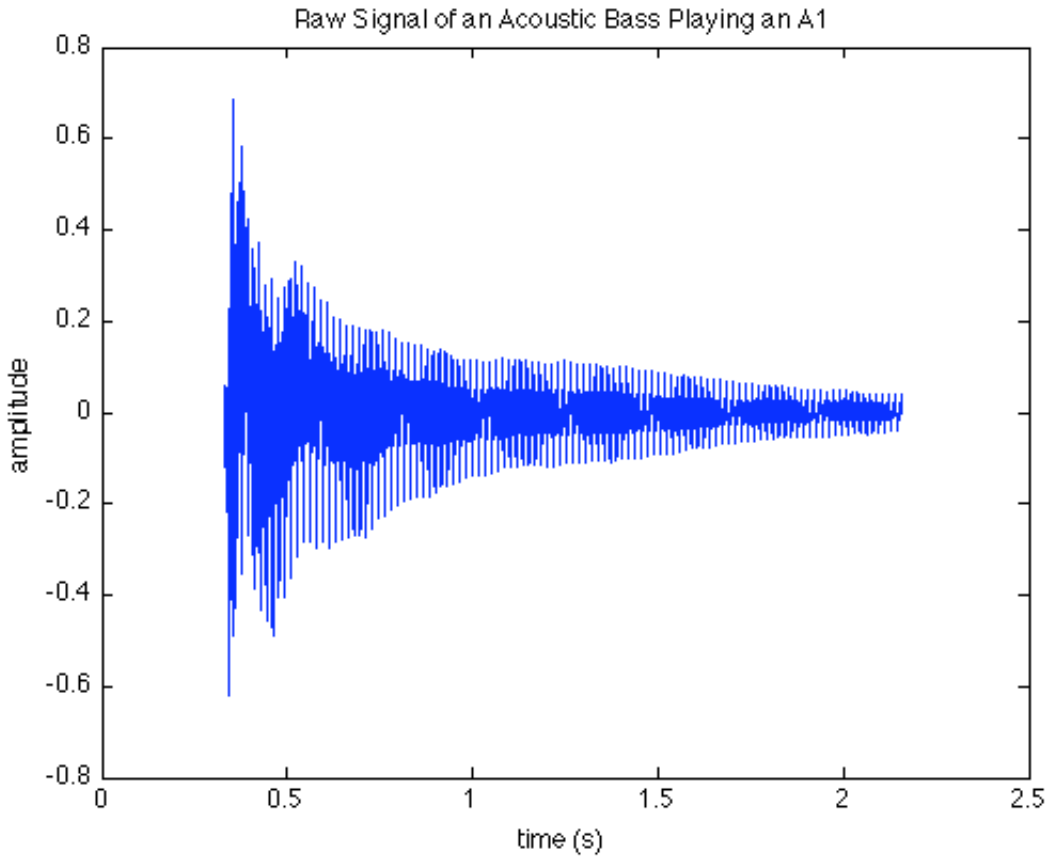

Figure 31: Raw Signal of Acoustic Bass

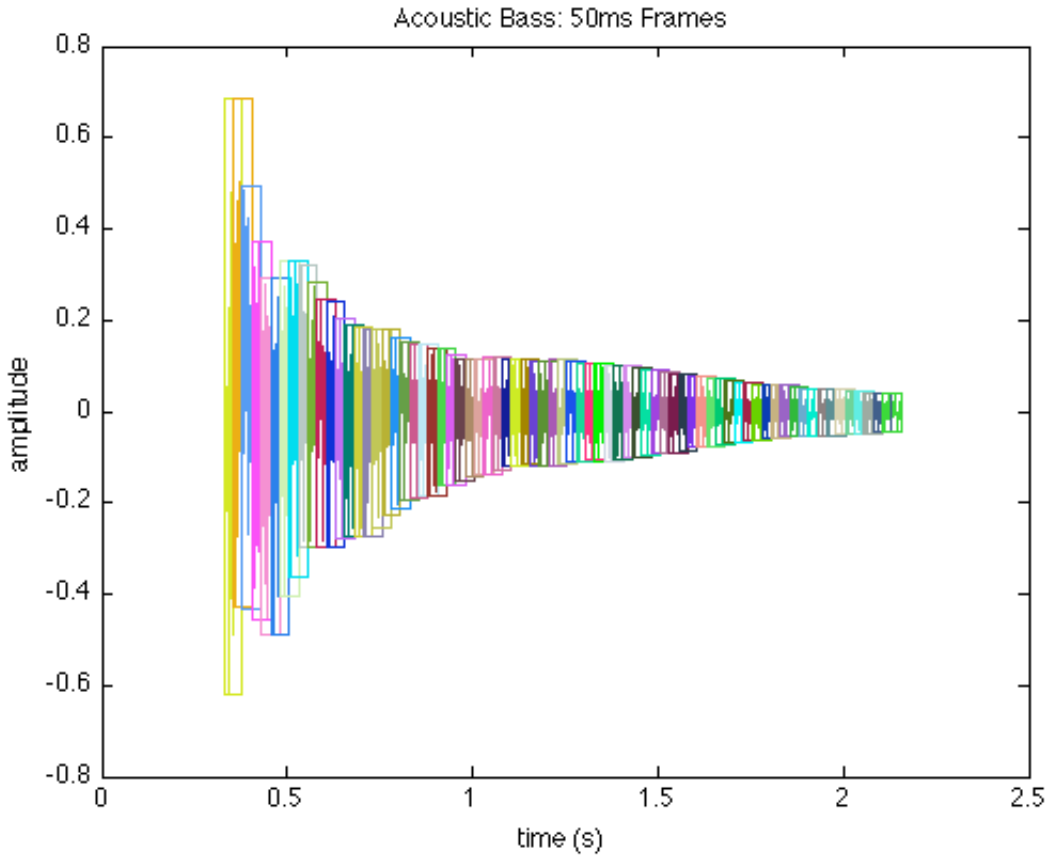

Figure 32: Acoustic Bass With 50 ms Frames

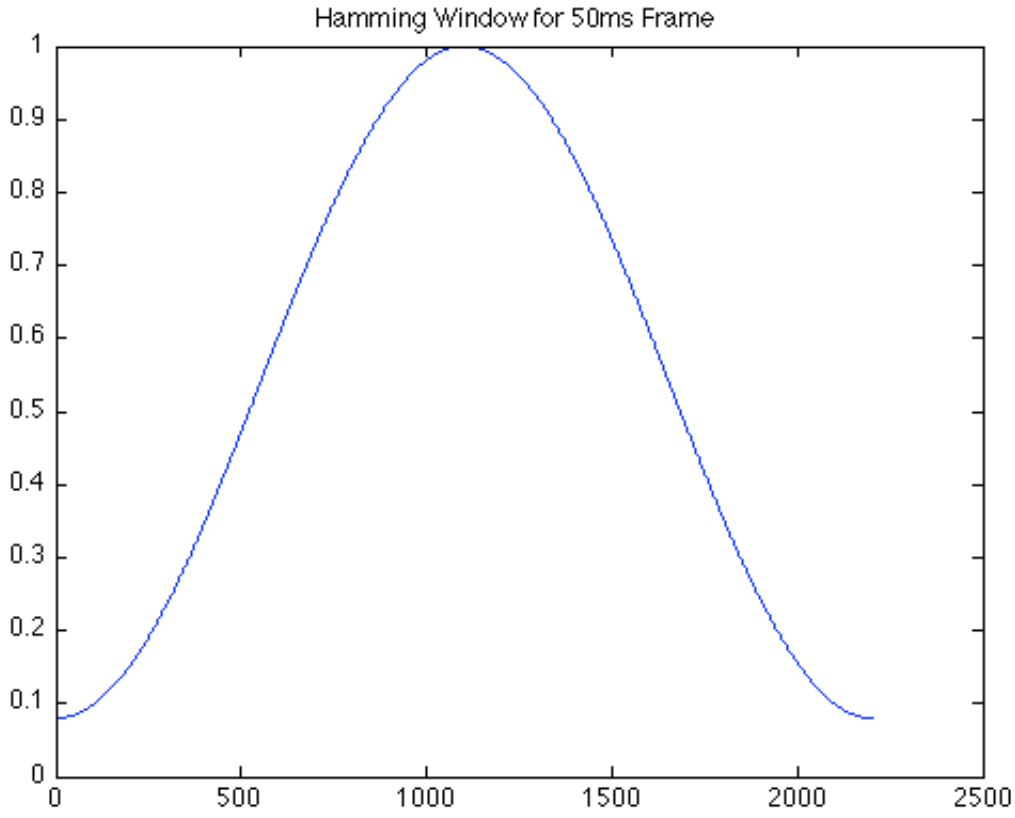

Figure 33: Hamming Window used for a 50 ms Frame. If the audio signal has a sample rate of 44100 samples per second, this corresponds to approximately 2205 samples per 50 ms window.

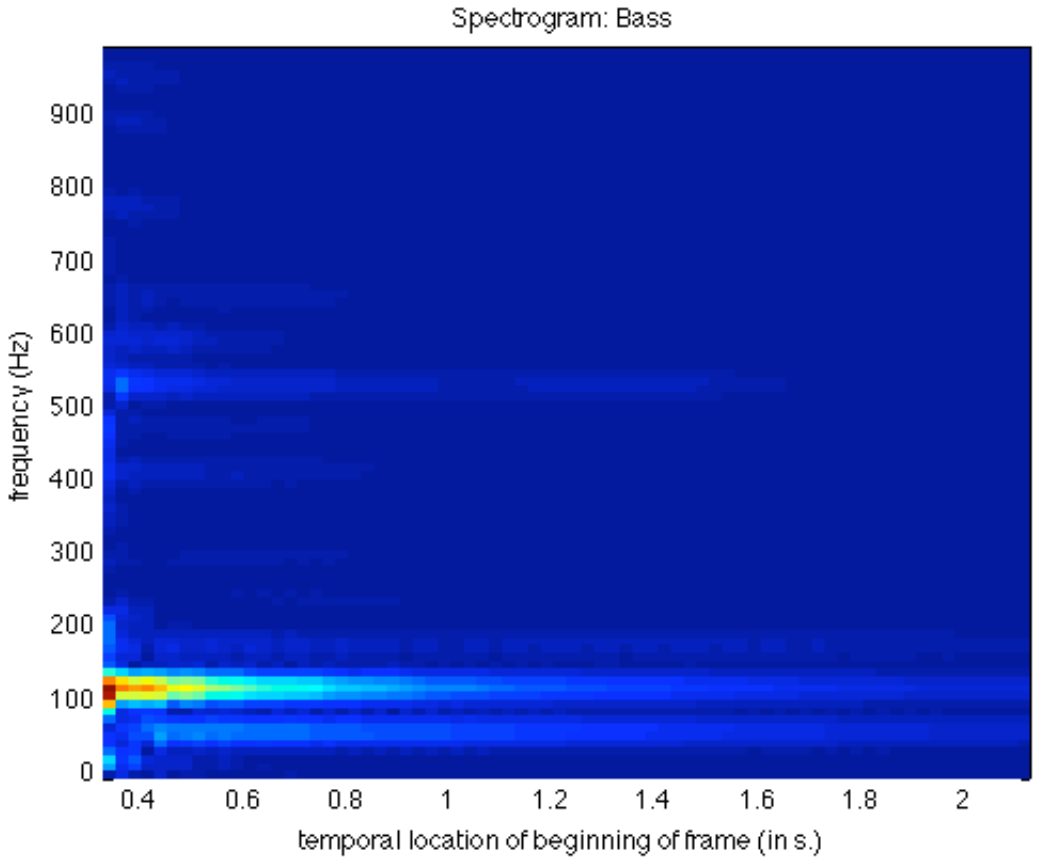

Figure 34: Spectrogram of Acoustic Bass, calculated using the DFT of 50 ms Frames

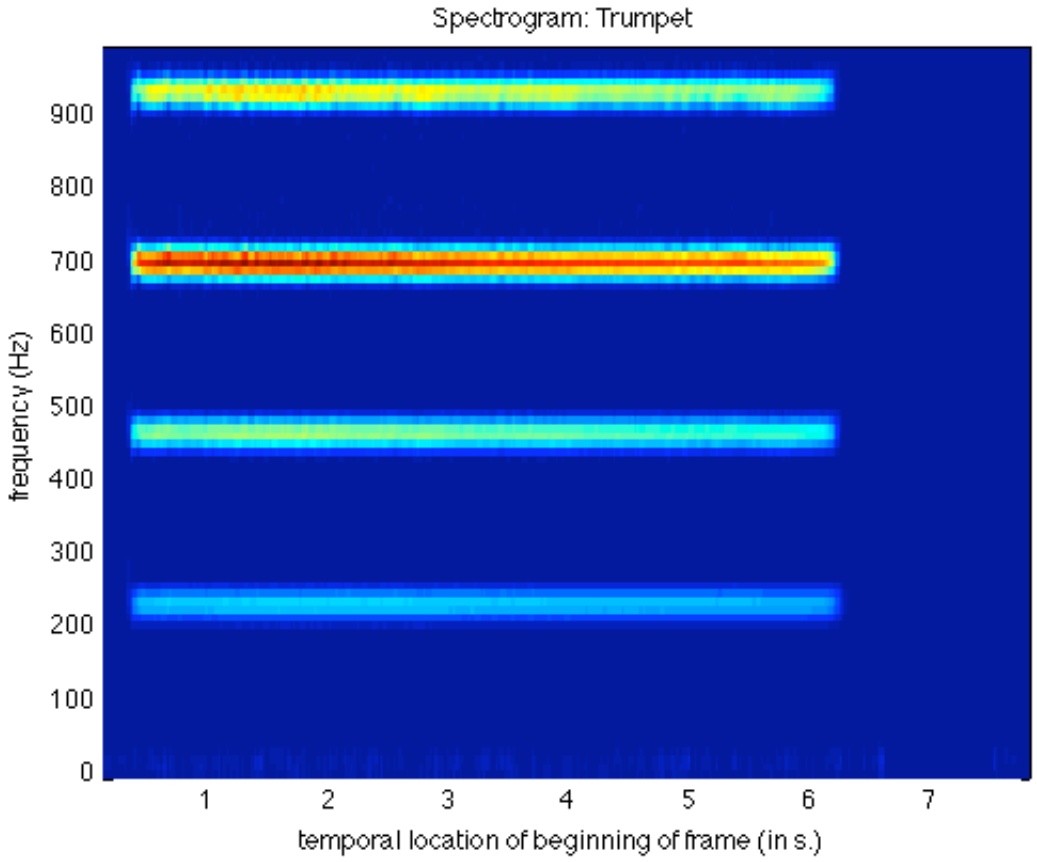

Figure 35: Spectrogram of Trumpet, calculated using the DFT of 50 ms Frames

After the audio signal is transformed into the frequency domain, several descriptors are computed from the spectrum and spectrogram, including the spectral centroid, skewness, spread, kurtosis, flatness, flux, irregularity, roll-off and Mel-Frequency Cepstral Coefficients (MFCC) (Zhang & Ras, 2007; Tzanetakis, Essl, & Cook, 2001). Other descriptive features that can be generated from the spectrum include brightness, pitch and roughness. Many of these descriptors have been standardized in the MPEG-7 Standard (Salembier, 2002). Appendix B contains brief descriptions of the features used in this experiment.

A.2 Temporal Features:

Aside from the spectral features, several temporal descriptors are used as well. Another technique often used in music signal processing is the extraction of the audio envelope for further analysis. The audio envelope has been described as the "global outer shape" of the musical signal (Lartillot, 2009) and provides loads of information about the onset, duration, and decay of the signal in question. Figure 36 is the raw signal of a short musical excerpt of ragtime piano, and Figure 37 is its audio envelope. From the audio envelope, several temporal descriptors can be calculated including the temporal centroid, flatness, kurtosis, skewness, spread; the attack time and slope, event density and zero crossings.

For more details of both temporal and spectral features, please consult Appendix B.

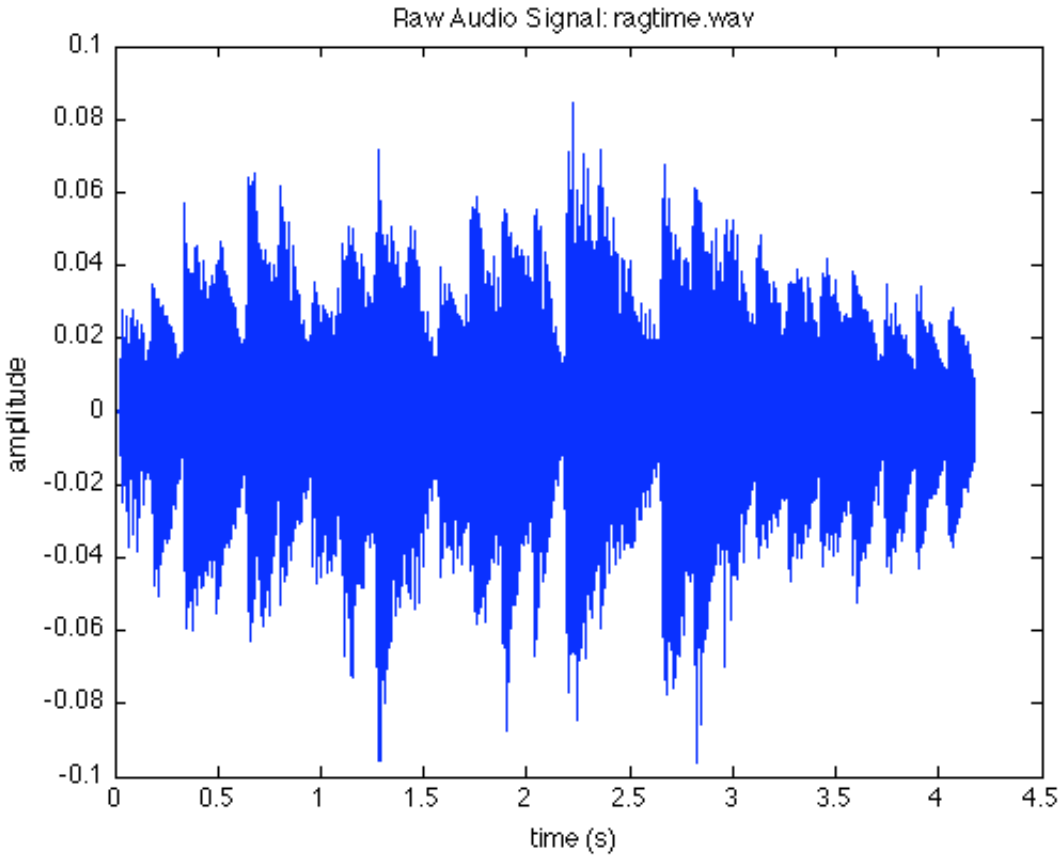

Figure 36: Raw Audio Signal of ragtime.wav

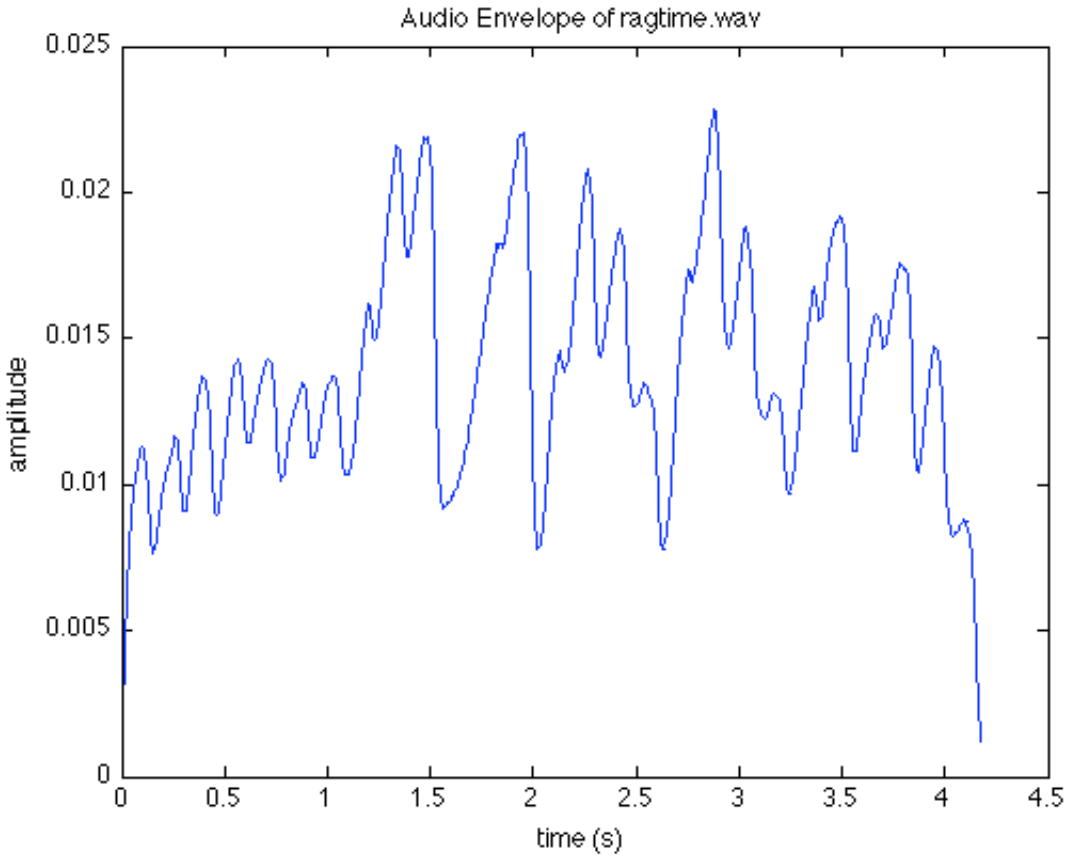

Figure 37: Envelope Extracted from ragtime.wav Using a Hilbert Transform

Appendix B: Glossary of Extracted Audio Features

The following is a list and brief description of all extracted features used in the experiment. All definitions are paraphrased from the work of Klapuri & Davy (2006), Lartillot (2009), Peeters (2006) and Salembier (2002). For more details on the mathematics involved in the calculations, please consult the previously mentioned literature (Appendix A). The following plots, figures and graphs were either generated in Matlab 7.7 using the open-source MIRToolbox (Lartillot & Toiviainen, 2007) or taken from the MIRToolbox's user guide (Lartillot, 2009). It is important to note that every feature listed can be easily extracted in Matlab using functions from the MIRToolbox.

Attack Slope- estimates the average slope of each note attack (Figure 38) (Lartillot, 2009).

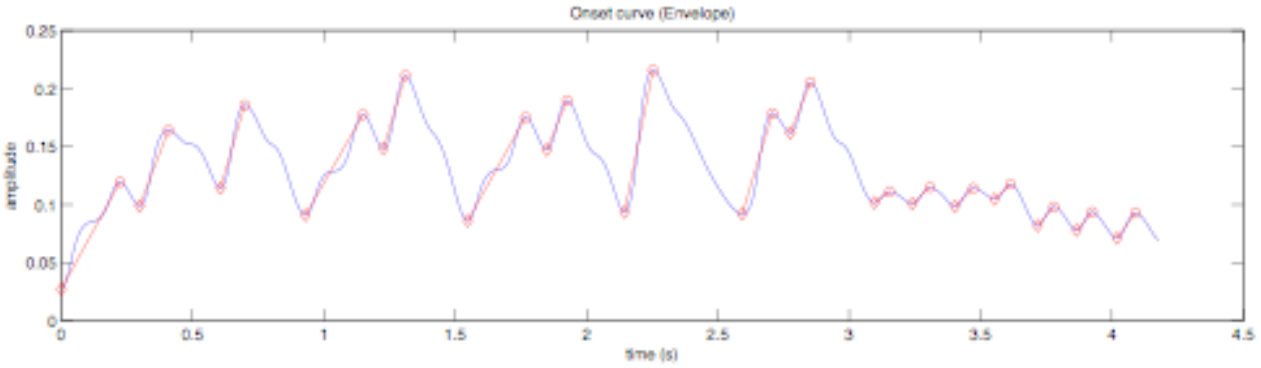

Figure 38: Attack slope of an audio signal. (Plot taken from MIRToolbox User's Guide)

Attack Time, Log- also known as "rise time" of the signal, and is the log of the time interval between the point the audio signal reaches 20% to 80% of its maximum value (Figure 39) (Klapuri & Davy, 2006).

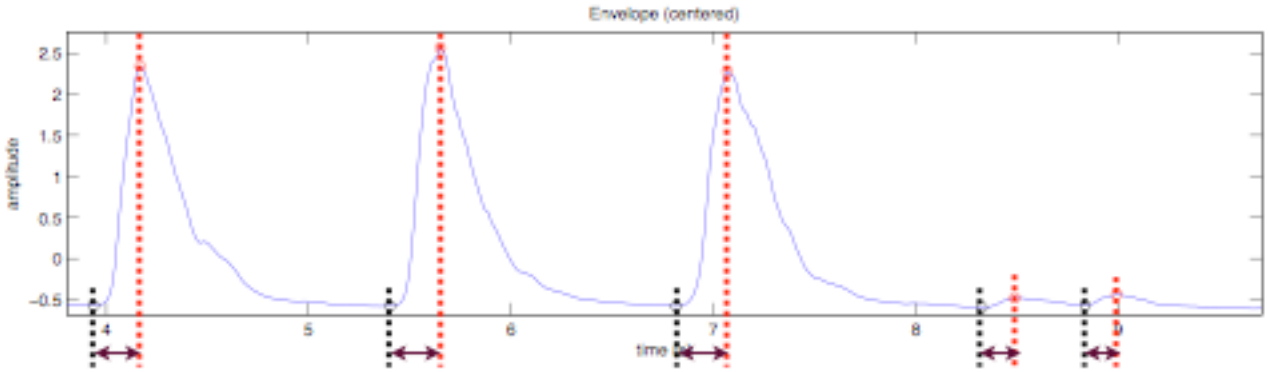

Figure 39: Attack time of a signal. (Plot taken from MIRToolbox User's Guide)

Brightness- the amount of spectral energy corresponding to frequencies higher than a given cut-off threshold (i.e., 1500 Hz). (Lartillot, 2009).

Event Density- estimates the average frequency of events, i.e., the number of note onsets per second (Lartillot, 2009).

Inharmonicity- the amount of partials that are not multiples of the fundamental frequency, as a value between 0 and 1. (Lartillot, 2009).

Mel-Frequency Cepstral Coefficients (MFCC)- often used in speech recognition, MFCC's represent a summary of the spectrum at each frame, across 13 Mel-frequency bands (Klapuri & Davy, 2006). The processing involved is depicted in Figure 40. The MFCC's of a Bass and Trumpet are in Figure 41 and Figure 42, respectively.

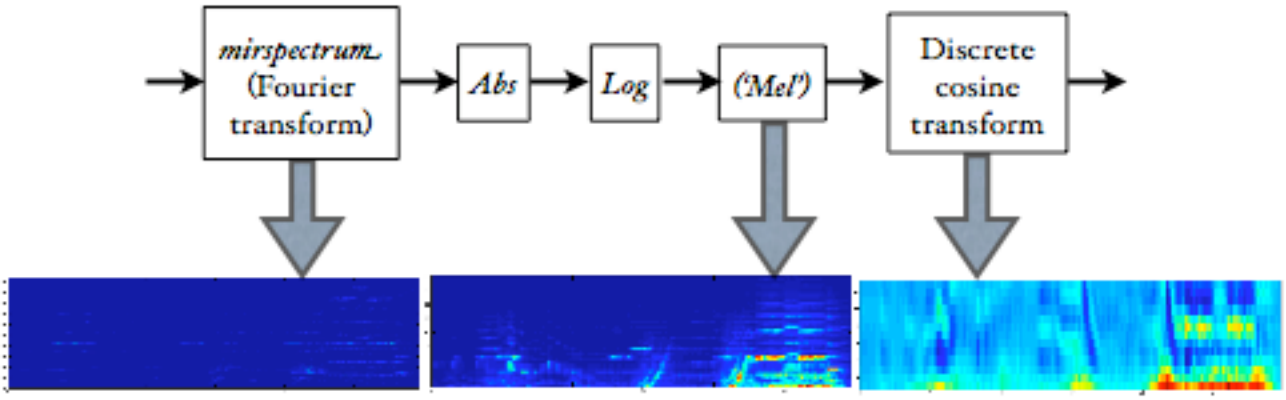

Figure 40: Steps for calculating MFCC's (Figure taken from MIRToolbox user's guide).

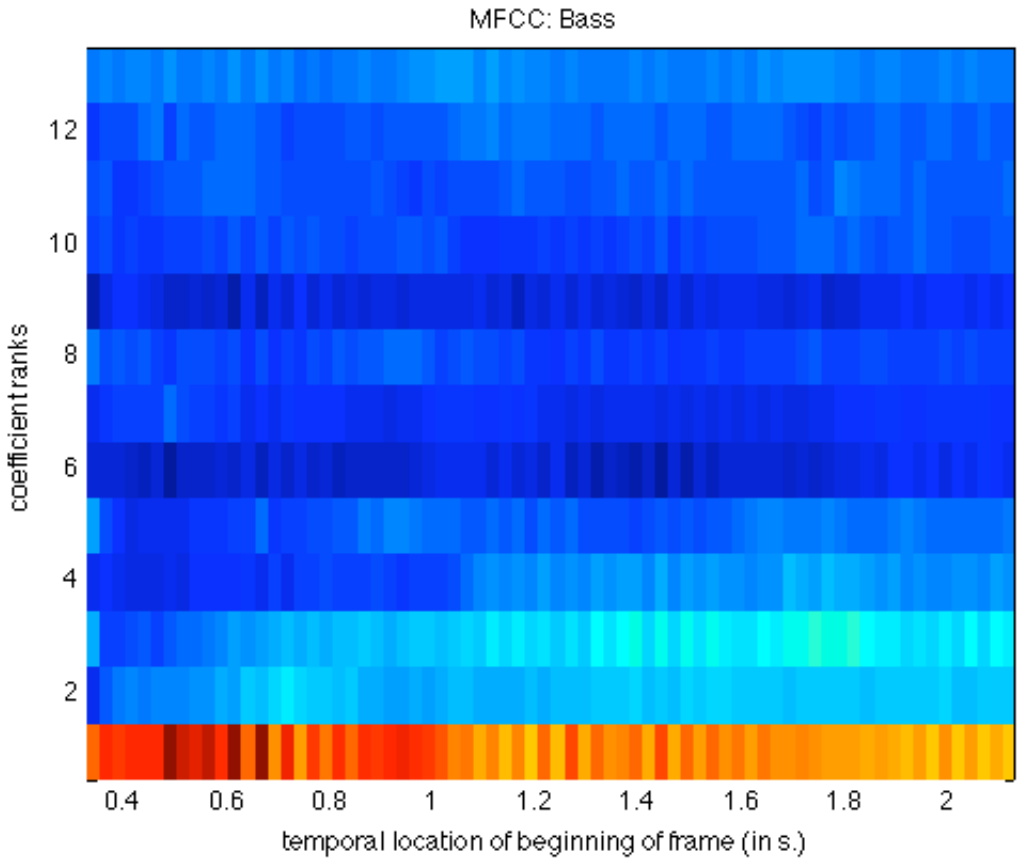

Figure 41: Mel-Frequency Cepstral Coefficients of a Bass

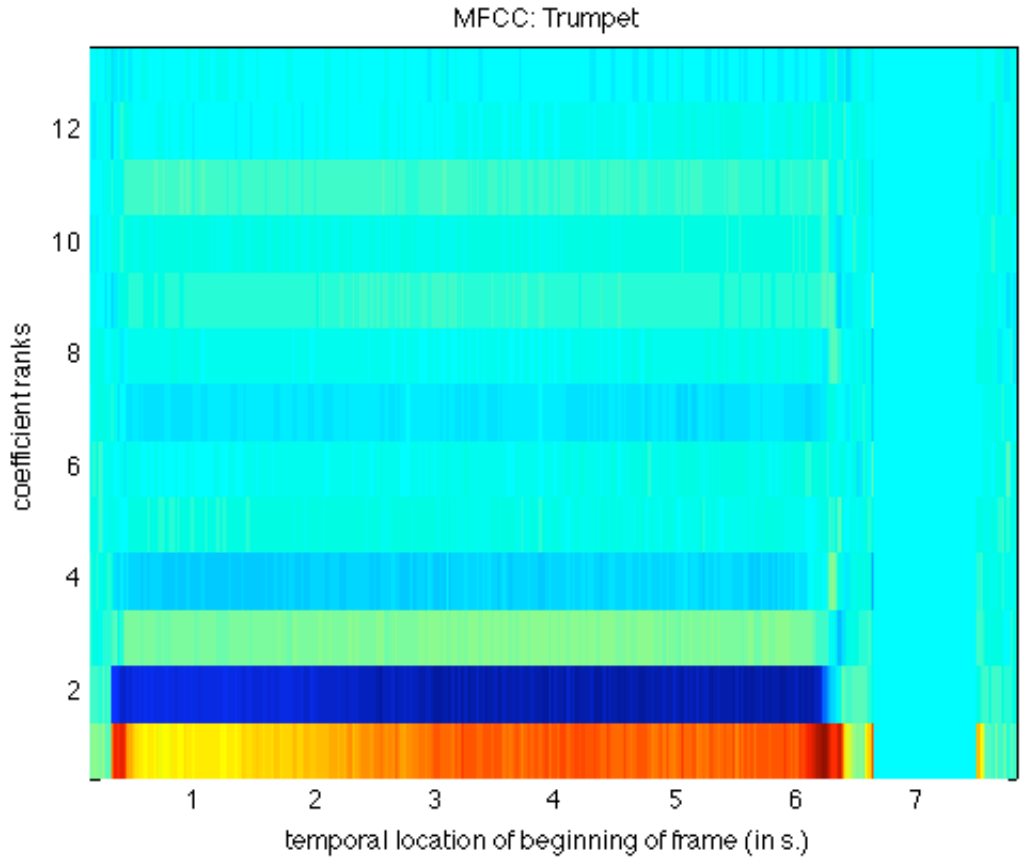

Figure 42: Mel-Frequency Cepstral Coefficients of a Trumpet

Pitch (or Fundamental Frequency)- the musical note, represented in Hz, found in the signal (e.g., A4 is 440 Hz) (Klapuri & Davy, 2006).

Regularity- the degree of variation of successive peaks of the spectrum (Lartillot, 2009).

Rolloff (85%)- the frequency at which a certain fraction (i.e., 85%) of the total energy is contained below that frequency (Figure 43) (Lartillot, 2009).

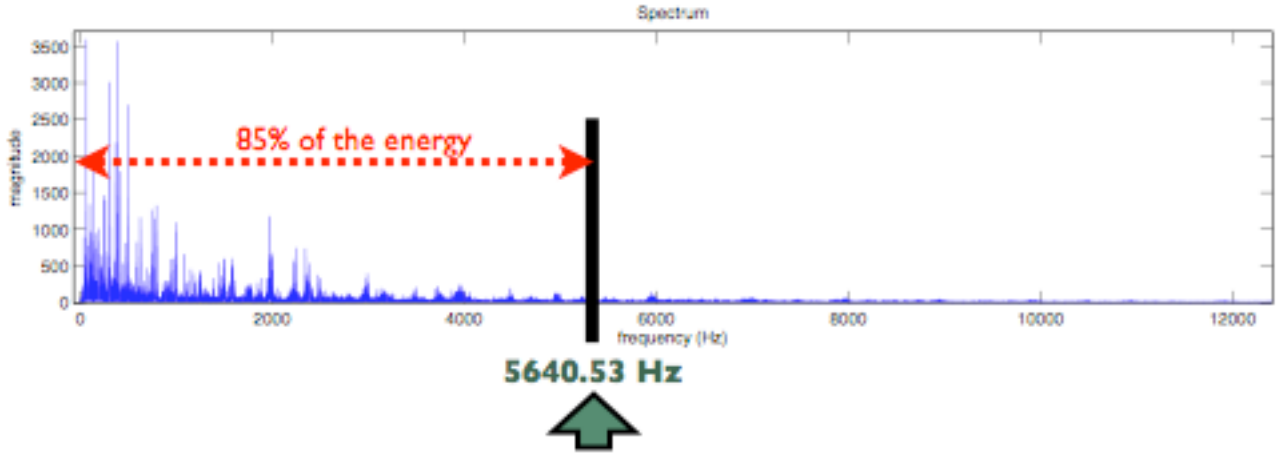

Figure 43: Rolloff (85%) of the audio signal. (Plot taken from the MIRtoolbox user's guide).

Root Mean Square (RMS)- the global energy of the signal. Computed by taking the root average of the square of the amplitude, also called root-mean-square (Lartillot, 2009).

Roughness- an estimation of the sensory dissonance, or roughness, related to the beating phenomenon whenever pair of sinusoids are closed in frequency (Figure 44) (Lartillot, 2009).

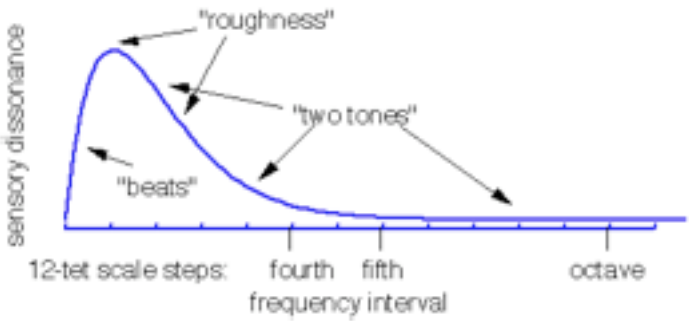

Figure 44: An estimation of roughness depending on the frequency ratio of each pair of sinusoids. (Figure taken from MIRToolbox user guide).

Spectral Centroid- has been described as the "center of gravity", "center of mass" and "barycenter" for the spectrum (Lartillot, 2009; Peeters, 2006).

Spectral Flatness- indicates whether the distribution is smooth or spiky, with low values indicating noisy sound. (Lartillot, 2009)

Spectral Flux- measures the local spectral change between the current and previous frames. (Lartillot, 2009)

Spectral Kurtosis- describes the "peakiness" of the spectrum; the smaller the kurtosis, the flatter the spectrum (Klapuri & Davy, 2006).

Spectral Skewness- describes the asymmetry of the frequency distribution about the spectral centroid (Klapuri & Davy, 2006).

Spectral Spread- represents the bandwidth of the spectrum (Klapuri & Davy, 2006).

Temporal Centroid- a time-based centroid of the signal envelope; often used to help distinguish percussive from sustained sounds (Peeters, 2006).

Temporal Flatness- indicates whether the audio envelope is smooth or spiky, with low values indicating noisy sound. (Lartillot, 2009)

Temporal Kurtosis- measures the "peakiness" of the audio envelope (Klapuri & Davy, 2006).

Temporal Skewness- measures the asymmetry of the audio envelope about the temporal centroid (Klapuri & Davy, 2006).

Temporal Spread- measures the distance from the largest specific loudness value to the total loudness (Peeters, 2006).

Zero-crossing- the number of times the signal crosses the X-axis or changes sign (Figure 45) (Lartillot, 2009).

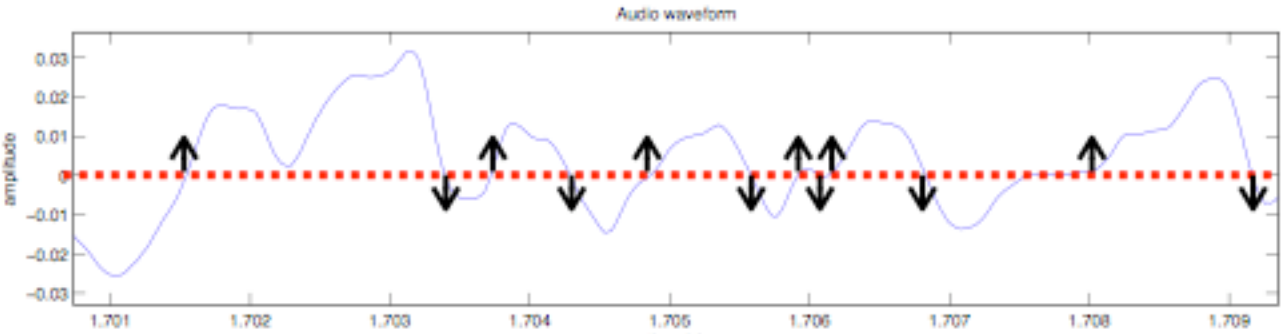

Figure 45: Zero-crossings in an audio signal. (Figure taken from MIRToolbox user guide).

Zero-crossing Rate- the rate of zero crossings per a fixed time interval (usually per second) (Lartillot, 2009).

Vita

Marc Rubin was born November 2, 1982 in Houston, TX where he graduated from Memorial High School in May of 2001. In May of 2005, Marc completed his Bachelor's of Arts in Psychology with a minor in Music. After working in a neuroimaging laboratory in Sacramento, California as a research assistant, Marc quickly became interested in understanding the mechanics of the computers that rendered all the fMRI graphics on the screen. After moving to Knoxville, TN in 2006, Marc began learning about computer science, both in academia and in industry.

Marc is currently pursuing a Master's of Science in Computer Science from the University of Tennessee and plans on pursuing a Ph.D. in Computer Science as well.