Musical Instrument Recognition

EECS 351 Final Report

Sophia Mehdizadeh, Peeravich Panlertkitsakul, Timothy Kennedy
EECS 351, College of Engineering
University of Michigan
Ann Arbor, Michigan
skmehdi@umich.edu, ppanlert@umich.edu, tikenned@umich.edu

I. Introduction

This paper examines the results of a code project designed to classify musical instrument samples into a set of predetermined categories. The program is set in this project to classify between three instruments: flute, saxophone, and violin, although it is designed with enough generality to accommodate other instrument classifications as well. We work with two data sets, a collection of reference training audio files, and a collection of random testing audio files. Given an appropriate library of sound samples of each instrument to be classified, the code library performs feature extraction on each sample and trains a computational model. This model can then be used to classify any random sample of any length, provided it is of the same type as one of the trained classifier instruments. In this implementation, feature extraction consists of collection of MFCC coefficients, as seen in speech recognition applications by Do [2] as well as previous instrument recognition such as Weng et al. Previously, two methods of instrument recognition that were widely used are the nearest neighbor method and the Bayesian networks, both shown by Donnelly et al [3]. The nearest neighbor method (k-NN) is an instance-based learning algorithm that classifies a sample with the most common class amongst its neighbors, which are predetermined by a distance metric. The Bayesian network method is based on probabilistic graphical models and uses prior probability of a class as well as conditional probabilities to determine the most likely class label.

For this project, we worked with two methods of instrument classification techniques: support vector machine (SVM), and convolutional neural network (CNN). The first method involves feature extraction using the Mel frequency cepstrum coefficients (MFCCs) and the second method involves various techniques of inputting data into the CNN. Both of these methods will be discussed in further detail in section II.

II. Methods

A. Coding Environment

Our main coding environment for this project is Matlab. We chose to use Matlab as it is a platform that we are comfortable with and we know that there are existing toolkits on speech recognition that we can adapt to our project. For this project, we integrated the code from various open sources to the code that we have written. The first method of this project makes use of an open source version of the MFCC feature extractor from the Hidden Markov Model Toolkit (HTK). The original HTK MFCC is intended for speech recognition application, and required several reworkings in order to properly handle a musical sample as opposed to human voice, which we will explain in section B. The MFCC feature vectors obtained using this method are then fed into the support vector machine for training. This process will be explained in section C. The second method involves cepstrograms, a type of spectrogram in the logarithmic Mel scale, as input to a convolutional neural network. Here, the Matlab Neural Network Toolkit is used, which we will further explain in section D.

B. Mel Frequency Cepstrum Coefficients

Mel frequency cepstrum coefficients are used to represent and characterize the power spectrum of a signal. For our purposes, the MFCCs are used to characterize various one-second-long samples of our chosen three instruments. These characterizations, or feature vectors, are then used to train our computational model. We therefore chose our training samples to consist of various playing styles and tones so as to create a more thorough and robust model. The MFCCs for each signal are generated as follows.

First, the signal is windowed in the time domain. Windowing in the time domain increases resolution in the frequency domain. For each window of N samples, the discrete fourier transform is taken, which can be represented by the following equation:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk} \qquad (1)$$

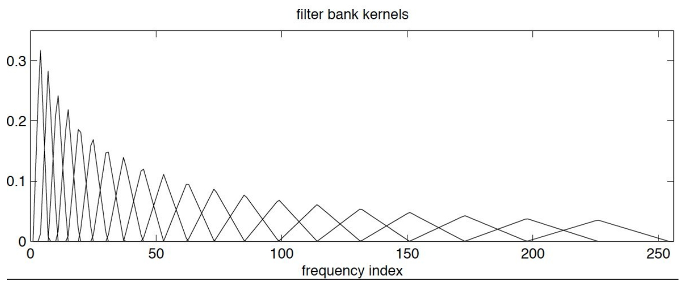

Fig. 1: Triangular filterbanks in Mel scale

Once this power spectrum has been obtained, a series of triangular, overlapping filters is applied (Fig.1). This maps our power spectrum onto the Mel scale, converting our units of frequency from hertz to mels. This conversion is represented by the following equation:

$$m = 2595 \cdot log_{10}(1 + \frac{f}{700}) \qquad (2)$$

Finally, the logarithms of the energies is taken and the discrete cosine transform (DCT) is applied. The purpose of the DCT is to compress the energy representation of the signal in a small number of coefficients, which in our case are a series of 13 MFCCs. The discrete cosine transform can be represented as follows:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot cos[\frac{\pi}{N}(n + \frac{1}{2})k] \qquad (3)$$

The result is a set of 13 MFCCs for each window of the original signal. We chose to consolidate these multiple sets of coefficients per signal by averaging across all of the windows to obtain one vector of 13 MFCCs per signal. A summary of the MFCC feature extraction process is shown in Figure 2.

All of this is done in our *trainSVM.m* file. As our code loops through all of our training samples and generates the average MFCC feature vector for each, it compiles each feature vector in a matrix *X1* which will be used as input for the SVM. The vectors are arranged inside *X1* such that each row is an observation, or audio sample, and each column is a predictor, or coefficient.

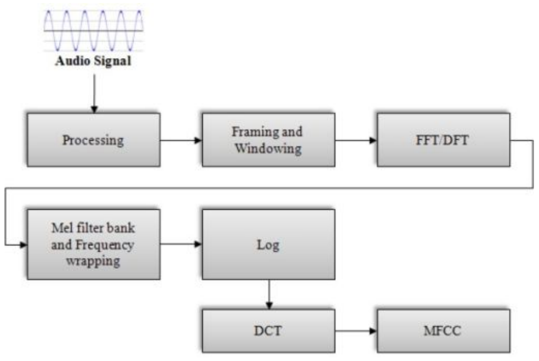

Fig. 2: MFCC feature extraction diagram

C. Support Vector Machine

A support vector machine (SVM) is a supervised learning machine that is widely used to classify data. The SVM does this by using a set of training examples to define a separating hyperplane in order to be able to classify the data. SVMs are binary learners, and are commonly used for binary classification, but in this project, we need a multiclass SVM in order to classify three instruments. In this project, we decided to input the a series MFCC coefficients to the SVM in order to train it. The code for training the SVM can be found in the *trainSVM.m* file. After going through the main for loop in this file, there will be two variables the are generated, *X1* (explained in section B.) and *Y*. *Y* contains the labels of each of the observations in *X1*, which in our case are violin, flute, or saxophone (represented by the numbers 1, 2, and 3, respectively). This will be used by the SVM to identify what type of instrument each observation in *X1* corresponds to. The code that we used to train the SVM is the *fitcecoc* function, which is preexisting in Matlab.

To obtain results from the testing samples, we need to obtain the MFCCs for the testing samples in a similar way in which we did for the training samples. In the *Classify_all.m* file, we use the *predict* function in the toolkit to predict the posterior probability that a value (coefficient) belongs to a certain class. In this algorithm, the probability distribution function of class k is *P*(k). The posterior probability that observation x belongs to class k is defined as follows:

$$P(k \mid x) = \frac{P(x \mid k) P(k)}{P(x)} \qquad (4)$$

D. Neural Network

An Artificial Neural Network (ANN) is an information processing method that operates similarly to biological nervous systems, such as the brain. The key element is the structure, a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. Our application makes use of a specific type of ANN, a convolutional neural network (CNN), which operates similarly to the visual cortex.

A typical CNN consists of multiple layers, including convolutional, max-pooling and fully-connected layers. The most widely used neural networks have been trained to operate within image processing, so this explanation will focus on that particular type of implementation. During training, the network defines a set of filters, or kernels, which are small grids of pixel data corresponding to a specific feature. This kernel is convolved with the entire image, and the new matrix formed is called a Feature Map, where each feature map discovers different features based on the response of the convolution. Each feature map is then downsampled to consolidate the information. A spatial neighborhood is defined, and either the largest element from that window is taken in the case of max pooling, or the average of all the sums in Average Pooling. This process of convolution and pooling comprises one layer, and multiple layers are constructed with each layer input being the output of the previous layer.

With each layer, more and more complex features can be analyzed and compared. The final step is a fully connected layer, which takes each layer as an input, and weights each based on complexity as an input to a function, called the Activation Function. This activation function is within a hidden layer, meaning it is not visible as a network output. The function returns probabilities as output that sum to one, and predict the classification of the original input image. Through this process, a neural network is able to apply feature extraction as well as classification, as shown in Figure 3. In the realm of audio classification, CNNs are often applied to images of spectrogram data, and as shown in Lee et al [] can equal or surpass MFCC methods.

In our attempts to implement a CNN, we tested various methods of inputting data into the network. For this project, we used the default CNN Toolbox in Matlab. The source code that we used was from the Matlab Script provided on the Mathworks website. This CNN source code was written for image classification purposes, such as letter recognition or object recognition. However, we believe that we can adapt this method to be used with audio inputs. In order to input the signal to the CNN, we tried three methods: reformatting MFCC data, using the spectrogram, and using the cepstral coefficient plot.

Since we already had the MFCC coefficients that were extracted from the training samples, we decided to reformat the coefficients into a 28x28 matrix as the first method of input into the CNN. We use this 28x28 matrix because this is the dimension that the CNN is expecting as an input. The function *cnnMFCCInput.m* creates two 4D matrices, one for training data and one for testing data (*XTrain* and *XTest*) and their corresponding label vectors (*TTrain* and *TTest*). These four matrices are used as inputs to the CNN.

The second method of input to the CNN is the spectrogram. At first, we tried inputting the spectrogram of the one input sample at a time, since our samples were one second long. However, we realized that this would not give us the desired results as the testing samples vary in length, and this time constraint will become an area that causes an error. Realizing that, we decided to window the original signal then plot the spectrogram of each window. Even though this increased the computation time by a lot, we believe that it would yield more accurate results. After plotting the spectrogram of window, we converted the image to grayscale in order to be able to feed the data to the CNN. Lastly, we resized the image so that it would have a size of 28x28. This is done in the file *cnnSpectroInput.m* where the images are again stored in 4D matrices before they are inputted into the CNN.

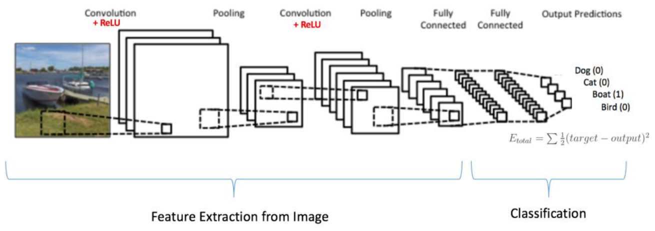

Fig. 3: Flowchart of CNN implementation

The third method for inputting data into the CNN is the Mel frequency cepstrum plot. We believe that using this will yield the best result. This is because the Mel frequency cepstrum captures the characteristics of a signal and is less sensitive to the frequency content of the signal when compared to the spectrogram plot. Contrary to the spectrogram method, we took one Mel frequency cepstrum plot per signal. Before inputting this plot into the CNN, we had to convert it to grayscale and resize it, just as we previously did with the spectrogram plots. This is done in the file *cnnCepstralInput.m*.

III. Results

E. SVM Results

Our first method is quantitatively evaluated using an accuracy percentage. A label vector similar to the vector *Y* described in section C. is generated for our testing samples. We then compare the results from the *predict* function to the "answer key" stored in the label vector, and find how many of the test files were correctly identified out of the total. The accuracy percentage is the output of the *Classify_all.m* function.

We found that our accuracy was higher when our test samples very closely resembled our training samples (both were from the same sound library, but no files were identical). This makes sense because it is much easier to find common features among files that have similar sonic characteristics. The distribution of correctly identified samples across the three different instruments can be seen in Figure 4 below. When we included test files from different sound libraries, our accuracy slightly dropped, but still remained at a respectable level. This is reasonable because since musical factors such as vibrato, reverb, and the amplitude envelope were no longer as similar to those in the training samples, the feature vectors derived from the test samples differed more. The distribution of correctly identified samples across the three different instruments for this case can be seen in Figure 5.

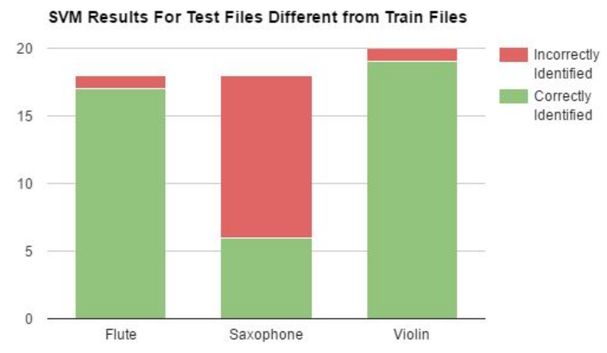

Fig. 4: Chart showing SVM accuracy per instrument

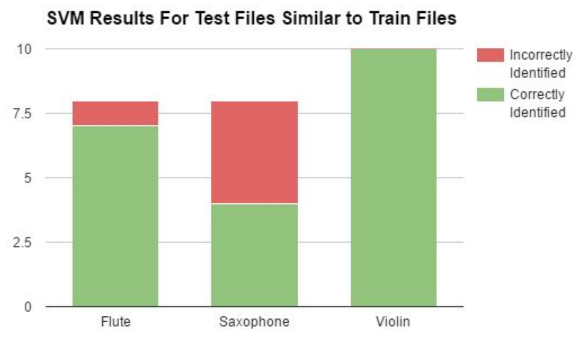

Fig. 5: Chart showing SVM accuracy per instrument

We can see that in both cases saxophone was the instrument with the most incorrectly identified samples. On the other hand, the flute and violin samples were almost all correctly identified. We believe that this is partly due to the fact that saxophone is an instrument with many timbres. Different saxophones in the same family (baritone, tenor, alto, soprano) can sound very different. Furthermore, a player can create many different sounds with the saxophone through various playing techniques. In some test files, the similarities between the training saxophone sound and the training violin sound are clearly audible.

A summary of the accuracy percentages for the SVM method is provided in Table 1. We found that other's implementations of this solution yielded accuracies of around 80% [1],[3]. Therefore we are satisfied with our results and consider the MFCC/SVM method successful.

F. CNN Results

An accuracy percentage is also used to quantitatively evaluate this method as well. The results for this method are separated into three different sections corresponding to the different methods we attempted: reformatting MFCC data, using the spectrogram plot, and using the cepstral coefficient plot. For all three methods, the file *Classify_cnn.m* takes the appropriate inputs and reports the accuracy percentage.

The first method, reformatting the MFCC data gave us inaccurate results. After observing the output, we realized that the CNN is classifying all the testing samples as one instrument (flute). This explains the accuracy percentage of approximately one third. We believe this is probably because of incorrect formatting of the MFCC coefficients to resemble an image. The feature information captured by the MFCCs may have been distorted in this process.

In the second method, using the spectrogram as input yielded improved but fairly inaccurate results as well. This time, the CNN output only classified input samples as either flute or violin, but never saxophone.

Lastly, inputting the Mel frequency cepstrum plot gave us similar results to the second method. This time the CNN output only classified input samples as either violin or saxophone. For a long period of time, this exact code was outputting undefined results. Previously, after going through extensive troubleshooting of the input to the CNN, we could not understand why the results were undefined. If the input to the CNN wasn't formatted correctly, or if the input wasn't in the correct dimensions, we will not be able to run the algorithm as there will be an error message (asserts). Since the procedure for formatting the data for our Mel frequency cepstrum plot is very similar to the procedure of formatting the spectrogram data, we expected it to return a valid accuracy percentage.

As a group, we believe that this method will eventually yield a high percentage accuracy, but we were not able to achieve that in the timespan of this project. Because there are many variables that affect the performance of the CNN such as the convolutional layers, max pooling, epochs, iterations, we think that there is an optimal number that would increase the performance of the system. The method that we use to input data to the CNN is very likely another factor that affects the performance. However, these are our hypotheses and there may be other factors involved as well.

Table 1: Summary of SVM results

	With Test Files Similar to Train Files	With Test Files Different from Train Files
SVM Accuracy Percentage	81%	75%

Table 2: Summary of CNN results

	With Test Files Similar to Train Files	With Test Files Different from Train Files
CNN Accuracy Percentage with MFCC Input	31%	32%
CNN Accuracy with Spectrogram Input	50%	42%
CNN Accuracy with Cepstrogram Input	26%	30%

IV. Conclusion

In this paper we study the effectiveness of instrument recognition by both SVM and CNN by use of MFCC's as input alone. We obtained reasonable accuracy with our SVM testing, however it dropped noticeably when more complicated samples were provided for classification. Our implementation of a CNN was unable to achieve accuracy at the level of the SVM, however given more time to fully optimize the system to train and test with spectrogram inputs would likely greatly improve given the results obtained in the current state. In addition, providing more various features for classification would improve accuracy. Research was done into envelope detection in order to analyze ADSR information, however due to the nature of the input samples this application was bypassed in favor of implementing the CNN. Future implementations could also include more instruments to classify, as well as subtypes (e.g. separating alto and tenor saxophone).

Acknowledgments

We would like to acknowledge Achilleas Anastasopoulos, our professor for the semester, for his sharing of his knowledge in his instruction. We also would like to acknowledge Nathan Sawicki, whose suggestions, advice and guidance was extremely instrumental in the completion of this project. Finally we acknowledge Kamil Wojcicki, who is the author of the MFCC HTK code package sourced for this project.

References

[1] D. Bhalke, C. Rao, D. Bormane and M. Vibhute, "SPECTROGRAM BASED MUSICAL INSTRUMENT IDENTIFICATION USING HIDDEN MARKOV MODEL (HMM) FOR MONOPHONIC AND POLYPHONIC MUSIC SIGNALS", ACTA TECHNICA NAPOCENSIS, vol. 52, no. 2, p. 9, 2017.

[2] M. Do, "DSP Mini-Project: Speaker Recognition", Ifp.illinois.edu. [Online]. Available: http://www.ifp.illinois.edu/~minhdo/teaching/speaker_recognition/. [Accessed: 8- Apr- 2017].

[3] P. Donnelly and J. Sheppard, "Classification of Musical Timbre Using Bayesian Networks", Computer Music Journal, vol. 37, no. 4, pp. 70-86, 2013.

[4] A. Eronen, "Automatic Musical instrument Recognition," M.S. thesis, Dept. IT, Tempere University, Tampere, Finland, 2001.

[5] J. Glover, "Sinusoids, noise and transients: spectral analysis, feature detection and real-time transformations of audio signals for musical applications," D.P. thesis, Dept. of Music, Nat. University of Ireland, Maynooth, Ireland, 2012.

[6] J. R. Jang, C. Lin and C. Weng, "Music Instrument Identification Using MFCC: Erhu as an Example", pdfs.semanticsscholar.org. [Online]. Available: https://pdfs.semanticscholar.org/4ec7/a1c395bdc2425c7892360f8e9d81d0c4c4a8.pdf. [Accessed: 8- Apr- 2017].

[7] H. Lee, Y. Largman, P. Pham and A. Ng, Unsupervised feature learning for audio classification using convolutional deep belief networks, 1st ed. Stanford, CA: Stanford University, 2009, p. 9.

[8] J. Lyons, "Practical Cryptography", Practicalcryptography.com, 2012. [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/. [Accessed: 21- Mar- 2017].

[9] "An Intuitive Explanation of Convolutional Neural Networks", the data science blog, 2016. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Accessed: 10- Apr- 2017].