Musical Instrument Classification Using a Hybrid Neural Network

Landon H. Buell[1] and Kevin M. Short[2]
[1]lhb1007@wildcats.unh.edu and [2]kevin.short@unh.edu
[1]Department of Physics and Astronomy
[2]Department of Mathematics and Statistics
University of New Hampshire, Durham, New Hampshire, USA

***Abstract*—Classifying audio signals with machine learning has become an important topic of research in the past few years. Models often involve the input of a 2-D spectrogram or 1-D feature vector into a unimodal network such as a Convolutional Neural Network (CNN) or Multilayer Perceptron (MLP). In this study, we explore automatic classification of musical instruments using new hybrid neural-network architecture that combines the CNN and MLP models and provides superior performance over models that rely solely on one or the other. This hybrid network uses two branches, one being a CNN to process an image-like 2-D spectrogram, and the other being an MLP to process a 1-D feature vector. Within the model, a hidden layer combines activations from the two branches by concatenating them into a single 1-D dense layer, thus all predictions are a product of both branches. We describe in detail the creating of the spectrogram and features, as well as how they influence the chosen network architecture. We finish with a practical demonstration that uses this classifier model to match waveforms from a chaotic music synthesizer to real-world musical instruments. Training data is from studio recordings of the Philharmonia Symphony Orchestra and University of Iowa's Electronic Music Studios**

I. Introduction

Digital audio analysis and classification has become a very prolific field in the last few years. The exact nature of each task can differ drastically from archival management, to security, or to commercial usage. In each case, we seek to place audio files into distinct categories called *classes* based in inherent properties within the data [3], [4], [7]. Consider a collection of audio files, each containing a single note performed by some musical instrument and the task of matching the contents of the file to the instrument that it most closely resembles. This type of classification task is near trivial for humans, but given the volume of audio data in the modern world, it is impractical at a large scale. However, a computer has no difficulty processing large volumes of raw information, but encoding an explicit instruction set for classification is unreasonable. For this reason, we turn to a *neural network* to combine the computational efficiency of a computer, with the decision-marking architecture of a brain [1].

A neural network is a machine learning model that allows a set of inputs, x, called *features*, to be transformed into a set of outputs, y, called *predictions* using a set of *parameters*, Θ [1], [2], [14]. Audio classification with machine learning is also a well-explored field, with much research in developing an appropriate set of features, also called *predictors*, for input [3], [7], [9], [16]. Since the each classification task or data set may differ greatly from any other, each model often requires a unique combination of features which allows for the best possible performance [14]. For musical instrument classification we have chosen predictors that represent the contents of an audio file using two different *modalities*. We call this type of classification *multimodal learning* [10].

Multimodal learning differs from other types of supervised learning in that a model accepts and processes multiple inputs that are each different representations of the same data sample. Data sets that include or can be transformed into audio + visual, audio + text, or even text + text information, are all examples of multimodal data sets [6]. For digital audio classification, we have chosen to represent the contents of an audio file by decomposing it into *image* + *vector* information format. We do this by constructing a 2D spectrogram and a 1D vector of features from the same waveform. Data from these different modes encodes complementary information which allows a model to develop parameters that can more thoroughly describe a sample when compared to it's unimodal learning methods [6]

To account for this multimodal input, we have constructed a *hybrid neural network* (HNN) that utilizes two input branches, each with its own set of layers to handle an input mode. The spectrogram image is processed with a Convolutional Neural Network (CNN) which uses layers of 2D convolution and 2D pooling to generate a feature map, which is then flattened and transformed by repeated dense layers, and is visualized on the left side of Fig. (5). The feature vector input is processed by a Multilayer Perceptron (MLP) which used only repeated dense layers, and is visualized on the right side of Fig. (5). The output each branch are concatenated into a single layer, which is further transformed into a single output prediction found at the bottom on Fig. (5). This means that the neural network learns a set of parameters which allows for mapping of *both* input modes into a single prediction. Since the properties of the input features are critically important to classification success, multirepresentation learning shows a great deal of promise and wide spread applicability [4], [6], [7], [14].

II. The Neural Network

A. Structure

A neural network is a type of machine learning algorithm that is inspired from the human brain [1], [2]. Where biological brains are constructed from neurons and connected through axons, neural networks are constructed from artificial neurons and connected through weighting functions. The numerical value contained within an artificial neuron is the *activation* of that neuron [5]. Neurons are organized into groups called *layers*, which interact with other layers through mathematical transformations such as matrix multiplications or convolutions [2]. Because of this, layers in the network can be considered a function, $f^{(l)}$ that accepts an array of activations and used a set of parameters to return an array of modified activations.

$$f^{(l)} : x^{(l-1)} \to x^{(l)} \tag{1}$$

By connecting multiple layers in succession, we form a layer-chain structure that describes the flow of information in the neural network [1]. This allows for the computational modeling of a network as a computational graph [2]. In a feed-forward network with L layers, inputs $x^{(0)}$ and outputs y^*, we can represent a layer-chain such as:

$$x^{(0)} \to f^{(0)} \to f^{(1)} \to ... \to f^{(L-2)} \to f^{(L-1)} \to y^* \tag{2}$$

The structure of a neural network's layer-chain defines it's *architecture*. The choice of architecture indicates the *hypothesis-space* of the model a determines that set of all possible solutions that a trained model can arrive at. For deep neural networks this function chain can be dozens of layers long, and contain upwards of millions of parameters. Increasingly complicated networks allow for a high dimensional solution space, which enables the creation of models with potentially greater performance [1], [2].

B. Input and Output

The inputs to a neural network or any machine learning algorithm are called *features* or *predictors* [3]. These are compact, low-dimensional representations of a data sample reflect its important characteristics [7]. Features should be chosen as to have low variance within each class and high variance between classes. For digital audio classification of musical instruments, features of the same musical instruments should exhibit very similar properties, while features from difference musical instruments should exhibit non-similar properties. In any learning algorithm, the role of feature extraction is to translate the raw information into descriptors that maximize the classification performance [14]. Although the neural network will classify the musical instrument within the digital audio sample, it will never interact with the raw waveform directly, instead will rely solely on these features. Suppose we use p predictors, for a classifier model. These features are usually combined into a single vector-like object for each sample, and then presented as input to the neural network [1]. We detail the classification features for this neural network in section (III-B).

The outputs to a classification neural network encodes the prediction that model has made for a particular sample. For a classifier with k classes, there are k output neurons transformed such that the sum of the activations is identically 1 [2], [3]. This means the k neurons can be treated as a probability density of each of the classes, where the neuron with the highest activation value is the class prediction of the sample. For classification of musical instruments, each class represents a musical instrument type. We use 37 classes of instruments ranging from woodwinds, to brass, to strings, to mallet percussion, to synthesizers. Each audio file contains exactly one musical instrument sustaining one note.

C. The Cost Function

A neural network is trained by providing a batch of input samples X, with a corresponding set of expected predictions Y. Each sample x in the batch input is expected to produce the label y, and when passed through the network, creates the prediction y^*. For an untrained network, we anticipate that y^* and y may differ greatly- meaning that the prediction is very different that the expected output, and classification performance is likely to be very poor. Since we know each sample in the data set to contain one, and only one musical instrument, we *one-hot-encode* the expected label y. For any sample $x^{(i)}$ that belongs to class j, we have a corresponding target vector $y^{(i)}$ given by:

$$y^{(i)} = \left[y_0, y_1, ..., y_j, ..., y_{k-1}\right] = \left[0, 0, ..., 1, ..., 0\right] \tag{3}$$

The same sample will produce a similarly sized vector, $y^{*(i)}$, which all elements sum to 1.

To quantity the difference between y and y^*, we introduce a *cost function*, $J(y, y^*)$ which is also called an objective function or a loss function [3]. The exact cost function used can differ based on the nature of the machine learning model or task, but typically multiple - category classification tasks use a *categorical cross-entropy* (CXE) function [1], [2]. The CXE cost for a single sample is defined as [2], [14]:

$$\text{CXE}[y, y^*] = -\sum_{n=0}^{k-1} y_n \ln(y_n^*) \tag{4}$$

Recall a sample belongs to class j. Since y is one-hot-encoded, the only non-zero term in the sum is $y_j \ln(y_j^*)$, where $y_j^* \in [0, 1]$. This returns a negative number, which we multiply by -1 to always yield a positive cost value. When a sample produces $y_j^* << 1$, the CXE cost returns a large value and when $y_j^* \approx 1$, then the CXE cost returns a small value. We provide a visualization of this behavior in Fig. (1).

We characterize the cost function as producing a value inversely proportional to the prediction confidence- A large cost value indicates a *poor* prediction label and a low cost value

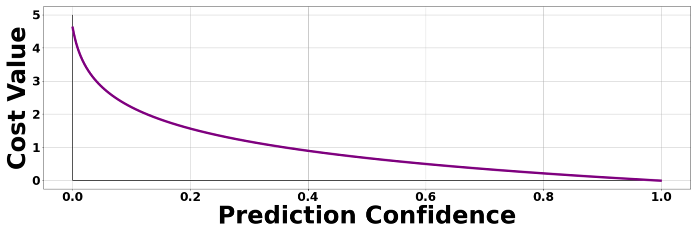


Fig. 1. Categorical Cross Entropy Cost Function given values for $y_j^* \in [0, 1]$

indicates a *good* prediction label [3], [14]. Thus, we expect a trained neural network to produce consistently low cost function values across all samples in a data set. When a neural network does return consistently low cost function values for previously unseen samples, we consider it to be a *fitted* or *trained* network [1]–[3].

D. Training and Optimization

A neural network makes predictions by successive transformations of input features through layer functions until a final output array is produced. Each layer $f^{(l)}$, of the model is a function that contains a set of parameters $W^{(l)}$ and $b^{(l)}$ that are used to transform activations through the layer chain [2]. The process of training the network is the manipulation of these parameters in each layer, such that each input x can produce a output, y^* that is reasonably close to y. Since we can quantify the difference between the expected output and the given output with a cost function, we can use this as a metric for measuring how "trained" a model is [3]. Specifically, we train a neural network by *optimizing* the parameters in each layer to allow for this consistently low cost function value [1], [2]. Note that we optimize the cost function under the assumption that doing so will also improve the classification performance. This is called indirect optimization [2].

Since a the function chain that comprises a neural network can become exceedingly complicated for deeper and wider networks, producing a single analytical expression for the optimization of every single is either impractical or impossible. We instead optimize the network with a numerical iterative method based on *gradient descent* [1], [2]. Gradient descent is the process of (i.) computing the gradient of the cost function with respect to every element in Θ and (ii.) adjusting each parameter in Θ according to the elements in that gradient vector, shown in Eq.(5), often scaled by a *learning rate* α. We compute the gradient of the cost function, $\nabla_\Theta J$ through a process call *backwards propagation* [1]. This algorithms uses the derivative chain-rule from multivariate calculus to begin at the last layer of the network, and work backwards through the layer chain in Eq. (2) to construct the elements of $\nabla_\Theta J$ [2], [14].

$$\Theta^{(i)} = \Theta^{(i-1)} + (-\alpha)\nabla_\Theta J \tag{5}$$

Standard gradient descent optimization is cumbersome, prone to numerical errors, and may often become unstable when encountering discontinuities in solution-space [1], [3]. Because of this, we employ a more robust and aggressive optimization algorithm called *Adaptive-Moments* or ADAM for short. ADAM uses adaptive learning rates, $s^(i)$ and $r^(i)$, that record and update exponentially decaying averages of past gradients and past squared gradients respectively. The ADAM update is given by Eq. (6).

$$\begin{aligned}
s^{(i)} &= \rho_1 s^{(i-1)} + (1-\rho_1)\nabla_\Theta \bar{J} \\
r^{(i)} &= \rho_2 r^{(i-1)} + (1-\rho_2)\Big[\nabla_\Theta \bar{J} \odot \nabla_\Theta \bar{J}\Big] \\
s'^{(i)} &= \frac{s^{(i)}}{1-\rho_1^i} \\
r'^{(i)} &= \frac{r^{(i)}}{1-\rho_2^i} \\
\Theta^{(i)} &= \Theta^{(i-1)} + (-\alpha)\frac{s'^{(i)}}{\sqrt{r'^{(i)}}+\delta}
\end{aligned} \tag{6}$$

ADAM provides a much more powerful and stable optimization algorithm at a higher computational cost [1]. Note that the superscript (i) indicates an iteration index where a superscript i is exponentiation. Typically we initialize $\rho_1 = 0.9$, $\rho_2 = 0.999$, and introduce $\delta \approx 10^{-7}$ to avoid possible division of near-zero values [2]. All operations in the final three steps are applied element-wise. ADAM has experimentally shown to be a very effective optimizer and useful for a wide range of data sets and architectures which is why we have chosen to implement it for this task [1], [2].

III. Features and the Hybrid Network

The architecture that we have developed for this model incorporates a convolutional neural network (CNN) and a multilayer perceptron (MLP) and is visualized in Fig. (5). The later hidden layers of the CNN branch flatten a 2D grid of activations into a 1D vector for each sample and pass it them into it's own MLP. The outputs of each branch are concatenated into a single dense layer, and a new output is drawn from the combined array of activations. We detail the structure and input features of each of the two input branches.

A. Convolutional Network Features

A spectrogram is a 2D representation of the energy distribution as a function of time and frequency in a periodic waveform-like signal [14]. The spectrogram is particularly useful because it allows us to examine the frequency composition of a signal, and how that composition evolves over time. On a typical spectrogram, the passing on time is shown on the x-axis, and frequency is shown on the y-axis. Because it is represented as a 2D array or floating-point numbers, the spectrogram is effectively an image-like representation of a sound wave. Since each musical instrument produces it's own harmonic structure and transient response, each instrument can be identified by patterns within the spectrogram [14]. This means than when only use this modality, the audio classification becomes very similar to an image classification task [10]. We provide a few example spectrograms for reference in Fig. (4).

A spectrogram is produced by applying the technique of *frame-blocking* to a waveform to produce a matrix of short-time analysis frames [7], [14]. We choose each frame to contain $N = 1024$ samples, with a 768 sample overlap between adjacent frames.

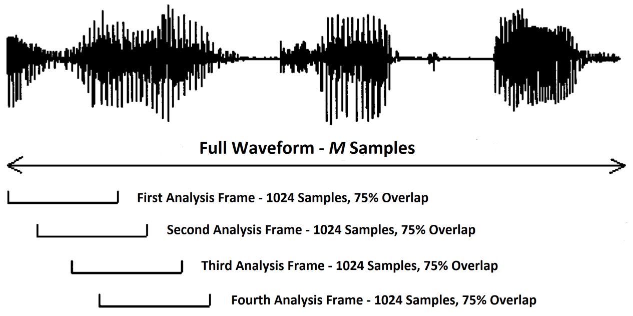

Full Waveform - *M* Samples
First Analysis Frame - 1024 Samples, 75% Overlap
Second Analysis Frame - 1024 Samples, 75% Overlap
Third Analysis Frame - 1024 Samples, 75% Overlap
Fourth Analysis Frame - 1024 Samples, 75% Overlap

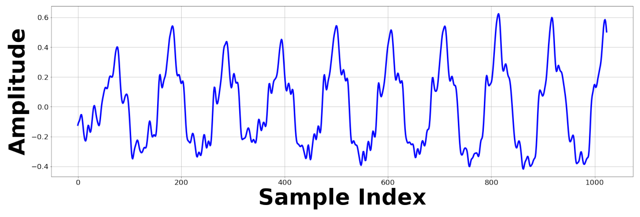
Fig. 2. Analysis frames in relation to a full waveform. This figures has been adapted from [7].

We organize the analysis frames into a matrix, A with shape $N \times k$, such that each row is a frame. We apply a Hanning window of length N to each row of A. This eliminates discontinuities at the edges of each analysis frame, and allows for a cleaner transform into frequency-space [14]. The effect of applying an Hanning window function to an analysis frame of 1024 samples can be found in Fig. (3). We tail-pad each time-series analysis frame by an additional 1024 zeros which brings each frame to 2048 samples, and gives a higher frequency-space resolution [12]. The spectrogram, S, is produced by computing:

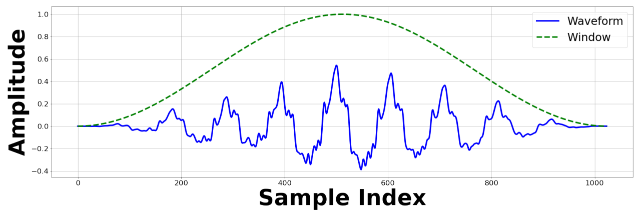
$$S = \left(\mathbb{W}A^T\right) \odot \left(\mathbb{W}A^T\right)^* \tag{7}$$

Where $\mathbb{W}$ is the standard *Discrete Fourier Transform* (DFT) matrix, and * indicate element-wise complex conjugation. For k analysis frames, matrix S has shape $2048 \times k$.

The standard western concert musical instruments that make up our base data set rarely have fundamental frequencies that extend beyond $6,000$ Hz. This means, that we will rarely see much energy or characteristic behavior above the $12,000$ Hz mark in the spectrograms. For this reason, we choose to crop each matrix as to include only information corresponding to the region of frequency-space form 0 Hz to 12 kHz. We also choose to zero pad, or crop matrix S so that there are exactly $k = 256$ columns. The result is an image-like representation of a digital audio file with shape 558×256, which is passed as a feature to the input of the convolutional neural network branch of the HNN. This is the first of the two modalities that we choose to represent our data sample.



(a)



(b)

Fig. 3. (a) A standard time-series analysis frame made up of 1024 samples and (b) The same frame with a standard Hanning Window Applied to it. This sample is taken from a bowed violin playing an $A4$ note.

B. Multilayer Perceptron Features

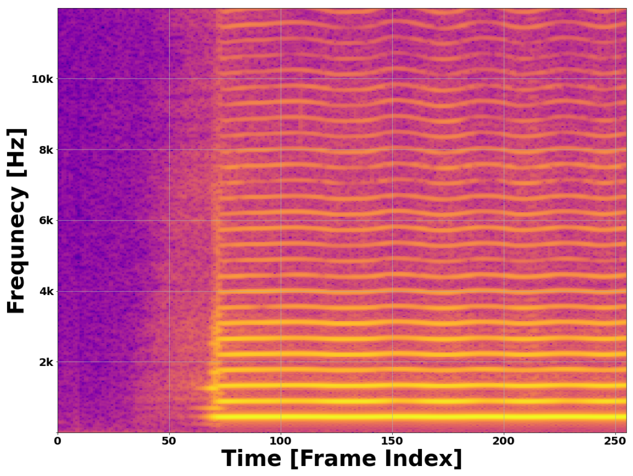
The feature is this section are derived from the time-domain or frequency-domain representation of a waveform. Each feature explored represents the values computed from each audio file, which are then assembled into a 1D array-like object called a feature vector. This vector is provided as input to the MLP branch of the neural network, which can be found on right side of Fig. (5). To ensure a consistency of all extracted features, all waveforms have been truncated or zero-padded to contain a consistent M samples.

1) **Time Domain Envelope** (TDE) - The TDE is a method of approximating the energy in the full or subset of the time-series waveform of a signal. We divide the waveform into 5 non-overlapping analysis frames and compute the RMS energy of the waveform in each section. This allows for an approximation of the amplitude envelope of the time-domain signal [14]. For a signal s, the RMS energy in the j-th frame containing Q samples is given by:

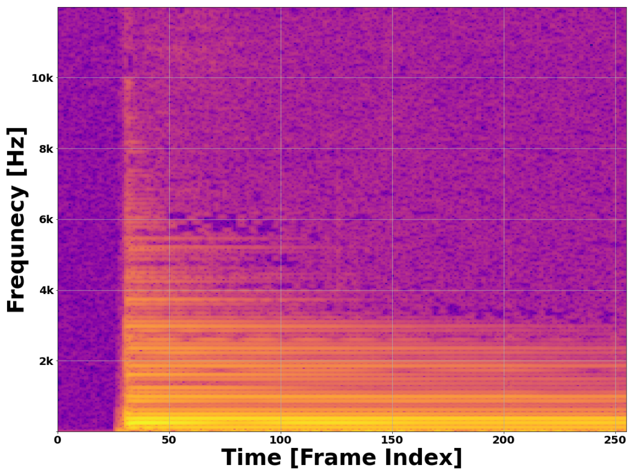
$$\text{TDE}_j[s] = \sqrt{\frac{1}{Q}\sum_{i=n}^{n+Q} s[i]^2} \tag{8}$$

These features allow us to characterize the time-evolution of energy in a signal. Instruments with heavy attacks, and sudden decays show large TDE values in early frames, and low TDE values in later frames. Instruments with longer sustains will show consistently decaying TDE values throughout the duration of the file.

2) **Zero Crossing Rate** (ZXR) - The ZXR of a waveform measures how many time that a signal crosses it's equilibrium value, often normalized per unit time. This feature is commonly used to differentiate speech from music



(a)



(b)

Fig. 4. Spectrograms of (a) an alto saxophone playing an $A4$ and (b) an acoustic guitar playing a $B2$

because speech often has a more jagged and less periodic structure, giving it a characteristically higher ZXR value [4], [14], [16]. We adapt this feature to compute the zero-crossing rate over the full waveform. The zero crossing rate for a waveform s with M samples is given by:

$$\text{ZXR}[s] = \frac{1}{2}\sum_{i=i}^{M-1}\left|\text{sign}\big(s[i]\big) - \text{sign}\big(s[i-1]\big)\right| \tag{9}$$

Where $\text{sign}(x)$ returns $+1$ if $x > 0$, -1 if $x < 0$ and 0 if $x = 0$. In generally, the zero crossing rate can also be used as a rough frequency measurement, where instruments that can produce higher fundamental frequencies often have higher ZXR values, where instruments that can produce lower fundamental frequencies have lower ZXR values [16].

3) **Temporal Center of Mass** (TCM) - The TCM of a waveform computes approximately where in time, the energy of a waveform is centered around, and can conveniently summarize characteristics of the transient response into a single scalar value. The temporal center of mass of a signal s with M samples is given by:

$$\text{TCM}[s] = \frac{\sum_{i=0}^{M-1} i\big|s[i]\big|}{\sum_{i=0}^{M-1}\big|s[i]\big|} \tag{10}$$

For instruments with heavier attacks and short decay and release times, such as plucked strings or percussion, we expect the energy of the waveform to be very early on, thus providing a very low TCM value. For instruments with shorter attacks, with long sustain and release times, such as bowed strings or woodwinds, we expect the energy of the waveform to be more spread out, thus have a higher TCM value.

4) **Auto Correlation Coefficients** (ACC) - ACC's are rough estimates of the signal spectral distribution. They are computed by taking the dot product of a signal with a time-expedited variant of itself, then the normalized to lie between 0 and 1 [14]. We can compute any number of ACC's and it's value will change according to the index chosen. In practice, it is common to compute the first K ACC's, with the k-th ACC for a signal s, with M samples is given by:

$$\text{ACC}_k[s] = \frac{\sum_{i=0}^{M-k-1} s[i]s[i+k]}{\sqrt{\sum_{i=0}^{M-k-1} s^2[i]}\sqrt{\sum_{i=0}^{M-k-1} s^2[i+k]}} \tag{11}$$

Dotting this signal with the time-shifted version of itself allows us to measure periodicity in time-space.

5) **Mel Frequency Ceptrum Coefficients** (MFCC) - In the MFCC computation process, a frequency spectrum is passed through overlapping triangle filter branks that are spaced according to the Mel scale [12]. This is done by calculating the dot product between the power spectrum of a signal and each of R Mel filter banks, which yields an approximation of energy in that frequency band. These are called Mel Filter bank energies (MFBE's). Each MFCC is found by computing the inverse discrete cosine transform of the log of the MFBE's. For R filter banks, the k-th MFCC is given by [14]:

$$\text{MFCC}_k[B] = \sqrt{\frac{2}{R}}\sum_{i=1}^{R}\log\big(B[i]\big)\cos\left(\frac{k(i-\frac{1}{2})\pi}{R}\right) \tag{12}$$

Where $B[i]$ is the i-th MFBE value. MFCC's allow for the investigation of periodicity in frequency-space, which identifies phenomena of overtones, echoes, etc [12], [14].

6) **Frequency Center of Mass** (FCM) - FCM computes approximately where in the frequency spectrum, the energy of a waveform is centered around. It is calculated by treating the power spectrum of a signal (or analysis

frame) as a 1D discrete mass distribution. The FCM of a power spectrum $\widetilde{s}$, containing M samples is found by:

$$\text{FCM}[s] = \frac{\sum_{i=0}^{M-1} i\left|\widetilde{s}[i]\right|}{\sum_{i=0}^{M-1}\left|\widetilde{s}[i]\right|} \tag{13}$$

For instruments with lower ranges and fundamental frequencies such as basses, bassoons or cellos, we find a consistently low FCM. For instruments with higher ranges and fundamental frequencies such as flutes, bells, or oboes, we find a consistently large FCM.

Each of these features represents an entry in a 1×24 *feature-vector* that describes each sample. This object conveys important characterisitcs of each digital audio in a list-like fashion. It is presented as input to the multilayer perceptron branch of the HNN. This is the second of the two modalities that we choose to represent each data sample.

C. The Convolution Branch

I will develop the math/behavior/functionality for the CNN in this section

D. The Perceptron Branch

I will develop the math/behavior/functionality for the MLP in this section

E. Multirepresentation Learning

The architecture of the HNN enables us to perform classify the contents of digital audio files as musical instruments through *multimodal* learning. How do I wrap up this section?

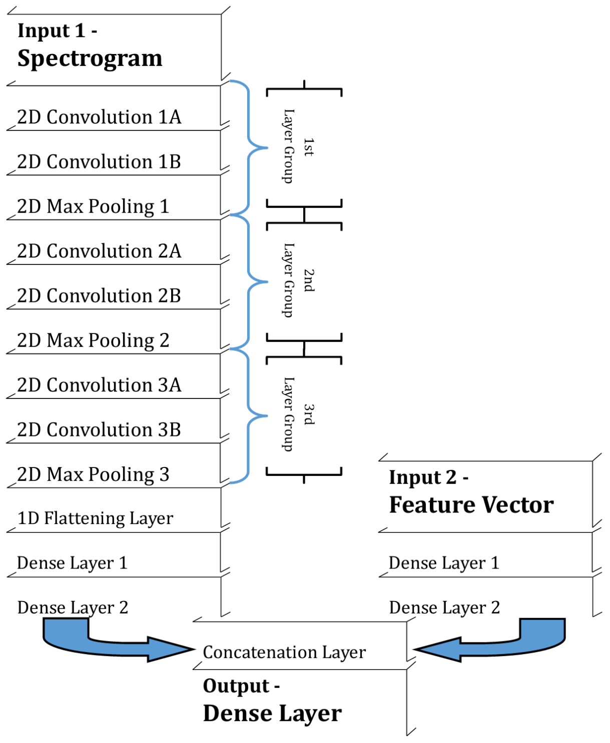


Fig. 5. Architecture for Hybrid Neural Network model, implemented with Tensorflow [13]

IV. Experimental Results

A. Resampling

In the machine learning workflow, it is standard practice to divide a full data set into a *training* subset and a *testing* subset [3], [14]. The training subset is used to optimize the parameters, Θ, in the model as to minimize the cost function, J, for the data set. The testing subset is used to evaluate how well the neural network performs on data that it has never interacted with [1]. We can expand this idea to a common resampling method called *K-Folds Cross Validation*, or X-validation for short [3]. X-validation divides a data set into K equally-sized folds. The first $K-1$ folds are used to train the model, and the last fold is used to evaluate how well the network performs on unseen samples. This process repeats of all K folds.

The result of X-validation is K neural network models that have all been trained and evaluated on overlapping subsets of the full data set. Each model contains a set of learned parameters, Θ that represent a possible outcome if trained on the given data set [3]. By repeating this and comparing the performance across each fold, we understand how the chosen architecture and hyper-parameters influence the stability of the performance of the network. Across each fold, we expect to observe a reasonably high, as well as consistent performance. This is indicates a model that can train effectively, and generalize appropriately. X-validation is critical to ensuring that a model is behaving as expected.

B. Performance Metrics

We choose the following metrics to evaluate the performance of our HNN. All metrics are define in relation to *true-positive* (TP), *true-negative* (TN), *false-positive* (FP), and *false-negative* (FN) predictions [1], [3]. These metrics are computed for each of K cross validation folds. We visualize the results of each split in Fig.(6.

- **Accuracy Score** - is the ratio of correct predictions to total predictions made by the model. While accuracy is convenient for quick estimates of performance, it does not provide much statistical information on the nature of predictions made of the model. This is especially true when classes have non-uniform representations in the data set. The accuracy score of a classification algorithm is given by:

$$\text{Accuracy} = \frac{TP + FN}{TP + TN + FP + FN} \tag{14}$$

Accuracy is bound between 0 and 1 with a higher score indicating better performance.

- **Precision Score** - is the ratio of correctly elements, to all relevant elements in the data set. The higher the score, the *specific* a model is to a particular category. The precision score of a classification algorithm is given by:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{15}$$

This metric determines how many of the selected items are relevant to the problem and is bound between 0 and 1 with a higher score indicating better performance.

- **Recall Score** - is the ratio of correctly elements, to all relevant elements in the data set. The higher the score, the *sensitive* a model is to a particular category. The recall score of a classification algorithm is given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{16}$$

This metric determines how many of the relevant items have been selected and is bound between 0 and 1 with a higher score indicating better performance.

- **F1 Score** - is the harmonic mean of the precision and recall scores. The higher the score, the more sensitive and specific a model is to a particular category. A high score indicate a model with both high precision and recall values. The $F1$ of a classification algorithm is given by:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{17}$$

$F1$ score is bound between 0 and 1 with a higher score indicating better performance.

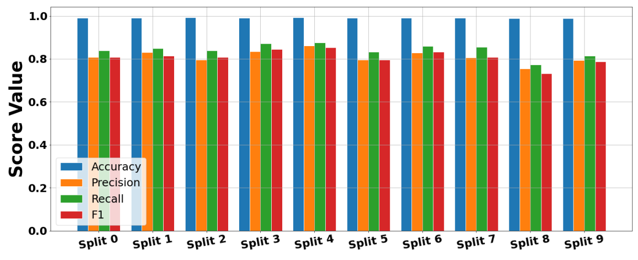
C. Confusion Matrices

A confusion matrix is a square array that counts the number of times a sample was predicted to be in a given class. For a k-categories classifier, the confusion matrix,C has shape $k \times k$ where $C_{i,j}$ is the number of times a sample known to be in class i was predicted to be in class j [1]. A value added to the main-diagonal indicates a *correct* classification, while a value on the off-diagonal indicates an *incorrect* classification. We expect a well-performing classifier model to produce a diagonally-dominant confusion matrix.

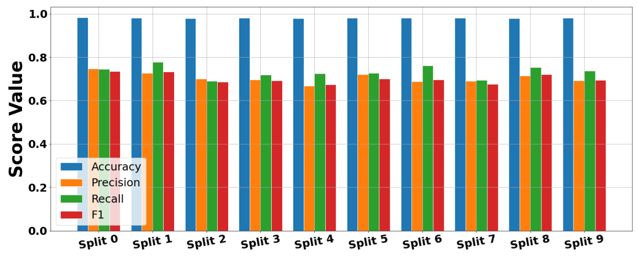
A standard confusion matrix does not account for any non-uniformity in the number of samples in each class in the data set. To combat this, we divide each row of the confusion matrix by the sum of that row. This has the effect of "normalizing" the confusion matrix according to how often class appears in a data set. Just like the metrics in section (IV-B), we construct the confusion matrix for each fold of cross validation, and then normalize it according to class occurrence. We average the K matrices together and visualize the result in Fig. (7.

D. Results

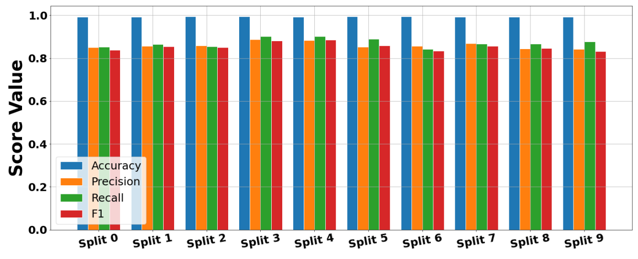
We have chosen to run a $K = 10$ fold cross validation program on our model, and compute the value of the each of the four metrics described in section (IV-B). We have also run X-validation on two similar models that represent either mode of the network. The first variant uses just a 2D spectrogram input, fed through an identical CNN + MLP as shown on the left of Fig. (5). The activations are passed directly to the output layer where a prediction is made using this mode alone. Similarly, the second variant uses just a 1D feature-vector input, fed through an identical MLP as shown on the right side of Fig.(5). The activations are again passed directly to the output layer where a prediction is made using this mode alone. This allows us to produce an approximate comparison of how the HNN compares to the CNN or just the MLP models.



(a)



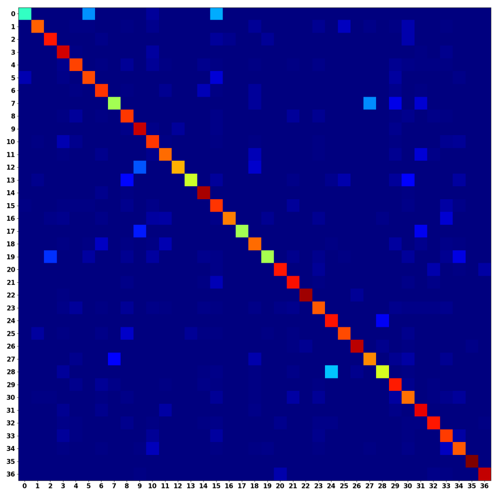
(b)



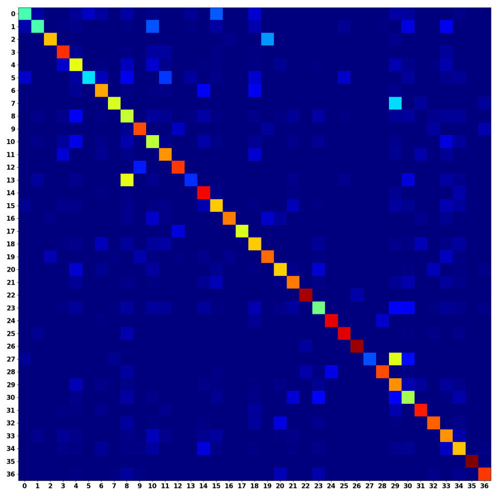
(c)

Fig. 6. Performances of the (a) CNN + MLP with spectrogram input, (b) MLP with feature-vector input, and (c) the HNN with spectrogram and feature-vector input

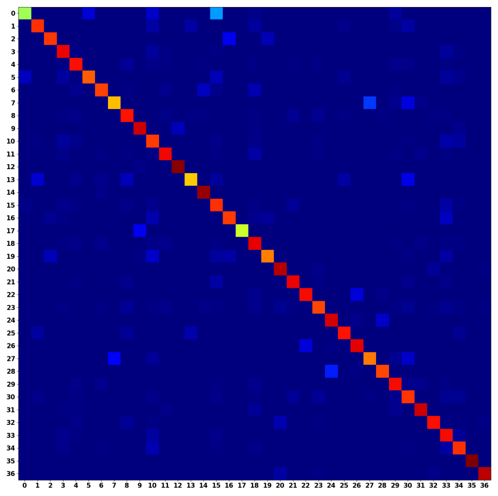
Explain those plots



(a)



(b)



(c)

Fig. 7. Performances of the (a) CNN + MLP with spectrogram input, (b) MLP with feature-vector input, and (c) the HNN with spectrogram and feature-vector input

Explain those plots too

V. Discussion and Conclusion

Multimodal learning itself shows great promise to machine learning algorithms in numerous domains. For data sets that lend themselves to multiple representations of a single sample, the ability to use those different modes allows for a more comprehensive representation of a sample. Across full data set, this will enable a model to generate a more stable set of parameters for classification which has the potential to yield better performance. Multimodal learning does not limit itself to classification of digital audio, but instead opens the door to many genres of machine learning. Finish this!

The design of this Hybrid Neural Network enables the input of two non-compatible modal representations of a sample to be process and combined to form a single output prediction. From the results of cross validation, we conclude that our HNN demonstrates better and more consistent performance at musical instrumental classification when compared to that of either the CNN or MLP models. By transforming each input sample into multiple representations of itself, and providing those modes to a single network with two entry layers, the HNN learns a more robust and stable set of parameters that can more thoroughly describe each class.

References

[1] Geron, Aurelien. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly, 2017.

[2] Goodfellow, Ian, et al.*Deep Learning*. MIT Press, 2017.

[3] James, Gareth, et al. An Introduction to Statistical Learning with Applications in R. Springer, 2017

[4] Khan, M. Kashif Saeed, and Wasfi G. Al-Khatib. "Machine-Learning Based Classification of Speech and Music." Multimedia Systems, vol. 12, no. 1, 2006, pp. 55–67., doi:10.1007/s00530-006-0034-0.

[5] Levine, Daniel S. Introduction to Neural and Cognitive Modeling. 2nd ed., Routledge, 2000.

[6] Li, Yingming, and Ming Yang. "A Survey of Multi-View Representation Learning." Journal of LateX Class Files, vol. 14, no. 8, Aug. 2015.

[7] Liu, Zhu, et al. "Audio Feature Extraction and Analysis for Scene Segmentation and Classification." Journal of VLSI Signal Processing, vol. 20, 1998, pp. 61–79.

[8] McCulloch, Warren S., and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, 1943, pp. 115–133.

[9] Mierswa, Ingo, and Katharina Morik. "Automatic Feature Extraction for Classifying Audio Data." Machine Learning, vol. 58, no. 2-3, 2005, pp. 127–149., doi:10.1007/s10994-005-5824-7.

[10] Ngiam, Jiquan, et al. "Multimodal Deep Learning." 2011.

[11] Philharmonia Symphony Orchestra home page- *https://philharmonia.co.uk/*

[12] Sahidullah, Goutam S. "Design, Analysis and Experimental Evaluation of Block Based Transformation in MFCC Computation for Speaker Recognition." 18 Nov. 2011.

[13] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[14] Virtanen, Tuomas, et al. *Computational Analysis of Sound Scenes and Events.* Springer, 2018.

[15] University of Iowa Electronic Music Studios home page- *http://theremin.music.uiowa.edu/*

[16] Zhang, Tong, and C.-C. Jay Kuo. "Content-Based Classification and Retrieval of Audio." *Advanced Signal Processing Algorithms, Architectures, and Implementations VIII*, 2 Oct. 1998, pp. 432–443., doi:10.1117/12.325703.