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]Founder of Integrated Applied Mathematics Program, 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 (HNN) 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 active 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 on inherent properties of 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 terabytes and petabytes 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 and accurate instruction set for classification is unreasonably challenging. For this reason, we turn to a neural network to combine the computational efficiency of a computer, with the decision-making architecture of a simulated brain [1].

At a basic level, 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 and data set 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 is designed to accept and process multiple inputs simultaneously 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 an *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 classify a sample when compared to unimodal learning methods [6]

To accommodate 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 - this is visualized on the left side of Fig. (5). The feature vector input is processed by a Multilayer Perceptron (MLP) which uses repeated dense layers to transform the input, and is visualized on the right side of Fig. (5). The output of each branch is 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, multi-representation learning shows a great deal of promise and widespread applicability [4], [6], [7], [14].

II. The Neural Network

A. Structure

A neural network defines 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 transformation 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, each layer l in a network can be considered to as function, $f^{(l)}$ that accepts an array of activations from the previous layer $x^{(l-1)}$ and uses a set of parameters to return an array of modified activations for the current layer, $x^{(l)}$. Thus each layer can be generally characterized by the expression:

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

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

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

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 the set of all possible solutions that a trained network can attain [2], [3]. 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 spaces, which potentially enables the model to describe more complicated systems.

The hybrid neural network used in this instrument classification task differs from the more standard architecture in Eq. (2) in that the function chain consists of two input layers, each of which feed into their own respective layer chains, noted by f_a and f_b. Each chain transforms activations through a non-overlapping graph, and are then merged at a subsequent hidden layer to produce graph f_c. The aggregation of both graphs allows us to generate a single prediction, y^* based on the behavior of both inputs, $x_a^{(0)}$ and $x_b^{(0)}$. We detail the structure an function of each layer chain in section (III-D) and (III-C). We represent the layer-chain structure for a generalized double-input HNN, with inputs $x_a^{(0)}$ and $x_b^{(0)}$ and outputs y^* as :

$$\begin{aligned} x_a^{(0)} \rightarrow f_a^{(0)} \rightarrow ... \rightarrow f_a^{(\alpha)} \searrow & \\ & f_c^{(\gamma)} \rightarrow ... \rightarrow f_c^{(L-1)} \rightarrow y^* \\ x_b^{(0)} \rightarrow f_b^{(0)} \rightarrow ... \rightarrow f_b^{(\beta)} \nearrow & \end{aligned} \tag{3}$$

This structure allows us to consider and design each branch a and b as if they were separate neural networks. The outputs from $f_a^{(\alpha)}$ and $f_b^{(\beta)}$ can be chosen to represents the predictions of the branches respectively, which are concatenated at layer $f_c^{(\gamma)}$. This marks the entry point of a new chain, which ends at the final output of the full model. Although we may consider the each layer-chain to be it's own entity, it is more appropriate to consider the chains to be subsets of the same computational graph. This means that Eq. (3) is considered to be one hybrid neural network model in the training and validation process.

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 that 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 from a digital audio file 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 implement a classification model that uses p unqiue predictors. 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-A) and (III-B).

The outputs to a classification neural network encodes the prediction that the model has made given a particular set of inputs from a sample. For a classifier with k classes, there are k output neurons transformed such that the sum (or $L-1$ norm) of the output activations is identically 1 [2], [3]. This means the k neurons can be treated as a probability density for each of the classes, where the neuron with the highest activation value is the prediction of the sample. For classification of musical instruments, each class represents a musical instrument type. We use 37 classes of instruments encapsulating woodwinds, brass, strings, mallet percussion, and 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. Consider a sample $x \in X$ that is expected to produce corresponding label $y \in Y$. When passed into the network, x produces some prediction label denoted as 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:

$$y^{(i)} = \left[y_0, y_1, ..., y_j, ..., y_{k-1}\right] = \left[0, 0, ..., 1, ..., 0\right] \tag{4}$$

The neural network will output an equivalently sized vector, $y^{*(i)}$, which all elements sum to 1.

To quantify 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]:

$$\text{CXE}[y, y^*] = -\sum_{n=0}^{k-1} y_n \ln(y_n^*) \tag{5}$$

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 $y_j^* << 1$, meaning a poor prediction, the CXE cost returns a large value and when $y_j^* \approx 1$, meaning a strong prediction, then the CXE cost returns a small value. We provide a visualization of this behavior in Fig. (1).
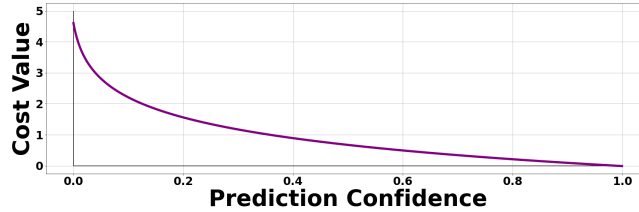

Fig. 1. Categorical Cross Entropy Cost Function given values for $y_j^* \in [0, 1]$

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 indicates a *strong* 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 show this behavior for previously unseen samples, we consider it to be a *fitted* or *trained* neural 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 held within arrays $W^{(l)}$ 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 parameter may be either impractical at best or impossible at worst. 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.(6), 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].

$$\Theta^{(i)} = \Theta^{(i-1)} + (-\alpha)\nabla_\Theta J \tag{6}$$

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. (7).

$$
\begin{aligned}
s^{(i)} &= \rho_1 s^{(i-1)} + (1 - \rho_1)\nabla_\Theta \bar{J} \\
r^{(i)} &= \rho_2 r^{(i-1)} + (1 - \rho_2)\left[\nabla_\Theta \bar{J} \odot \nabla_\Theta \bar{J}\right] \\
s'^{(i)} &= \frac{s^{(i)}}{1 - \rho_1^i} \\
r'^{(i)} &= \frac{r^{(i)}}{1 - \rho_2^i} \\
\Theta^{(i)} &= \Theta^{(i-1)} + (-\alpha)\frac{s'^{(i)}}{\sqrt{r'^{(i)}} + \delta}
\end{aligned}
\tag{7}
$$

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].

E. Layers Used in this Classification Model

As mentioned in section II-A Eq. 1), each layer in the neural network acts as it's own self-contained function that operates on the activations of the previous layer to transform them into the activations for the current layer. This function can take many different forms depending on where the layer sits in the chain, the type of information in the previous layer, and the nature of the solution that the neural network is trying to model [?], [2]. For this classification task, we employ five different layer types - (i) Dense, (ii) 2-dimensional convolution, (iii) 2-dimensional maximum-pooling, (iv) flattening, and (v) concatenation. In this section, we detail the forward-pass process of each layer and in Fig. (5) we show how the combination of these layers are used to form the multimodal architecture of this network.

1) **Dense Layer** The dense layer, also called *fully-connected* or *affine* layer is one of earliest layer types used in artificial neural networks [?], [1]. A dense layer is composed of a collection of artificial neurons, each of which contains a numerical value within it (usually a floating-point number), called the *activation* of that neuron. To compute the activation of any neuron, $x^{(l)_i}$, we take a weighted linear-combination of every neuron in the previous layer and add a constant to it, called a *bias* [?], [1], [2].

It then follows that a full *layer* of activations, is produced from the matrix-vector product of a weight-matrix $W^{(l)}$, and the previous layers of neurons $x^{(l-1)}$, and then added to a bias $b^{(l)}$ [2], [13]. We can model this transformation as :

$$x^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)} \tag{8}$$

Suppose the layer $(l - 1)$ contains m neurons, and layer (l) contains n neurons. Each object then belongs to space: $x^{(l)} \in \mathbb{R}^{(n \times 1)}$, $x^{(l-1)} \in \mathbb{R}^{(m \times 1)}$, $b^{(l)} \in \mathbb{R}^{(n \times 1)}$, and $W^{(l)} \in \mathbb{R}^{(n \times m)}$. This transformation allows for an affine-transformation of activations which enables this layer to learn roughly linear decision boundaries between adjacent layers of

neurons. This property of connecting every neuron in one layer to every other neuron in the previous layer is called *dense connectivity* which is what gives this layer its name.

Formal implementations of this the affine transformation layer type introduce non-linearity into the hypothesis space by applying an *activation* function to Eqn. (8) [2]. This enables us to model the far more complicated decision boundaries found in real-world problems. For this classification network, we choose to use the *ReLU* activation function, defined as:

$$ReLU\big[x^{(l)}\big] = \max\big(0, x^{(l)}\big) \tag{9}$$

This activation function preserves only the positive portion of each activation. Does this Seem Out of place to you?

2) **2-Dimensional Convolution Layer** Convolution Layers Emerged from studying the brain's visual cortex, and have been used for image recognition [**?**], [1], [2]. The layer gets it's name due to it's implementation of the mathematical 2D *convolution* operation. The 2D discrete convolution of function or 2D array A and B is given by:

$$C[i,j] = (A * B)[i,j] = \sum_y \sum_v A[i,j]B[i-u,j-v] \tag{10}$$

We implement this function in the neural network by creating a series of K weighting matrices, called filters, of size $m \times n$, each to take the place of A above [1]. The shape of the filters in referred to as the *convolution filter size*, and step them along in two dimensions according to a set *stride size* [**?**], [2].

In the case of a 2D input array, $x^{(l-1)} \in \mathbb{R}^{(M \times N)}$, the convolution filters step through the input data and repeatedly compute the element-wise producr of each $m \times n$ filter and the local activations of the $x^{(l)}$ array. Each of the K filters generate a new array of activation. For an appropriately trained neural network, some filters may result in detecting vertical lines, horizontal lines, sharp edges, or areas of mostly one color, etc [**?**].

Just like the dense layer, we also choose to add a bias term to each output neuron, and can further modify the array by applying a non-linear activation function. for 2D convolution layer (l), given input $x^{(l-1)}$, we compute the actions of the k-th filter $x_k^{(l)}$ using filter $W_k^{(l)}$ and bias $b_k^{(l)}$ as [2].

$$x_k^{(l-1)}(i,j) = b_k^{(l)} + \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} W_k^{(l)}(i,j)x_k^{(l-1)}(i-u,j-v) \tag{11}$$

The Convolution layer allows for a few advantages over the dense layer. Among these are (i) *spare-connectivity*: not every input neuron is connected to every single output neuron, so it can be more computationally efficient, (ii) *positional invariance*: key features can be identified regardless of where the are in the input array, and (iii) *Automatic feature detection*: as the weighting kernels are trained, they update as to automatically determine dominant features in the data without explicit human instruction [**?**], [1], [2].

3) **2-Dimensional Maximum Pooling** A maximum-pooling layer returns the value of the maximum neuron activation in a local subset of neurons. In the case of 2D max-pooling, we choose a kernel size $m \times n$, and pass that kernel over the inputs according to a chosen stride size [13]. The kernel takes the largest value in that area of inputs, and then places it into the output activations array. THis operation is particularly useful for compressing the amount of data down to a computational efficient size.

Pooling layers are typically place after a layer or set of layers of convolution [**?**]. The purpose of this is arrangement is to reduce the width of the model at the pooling layer and ensure that only the dominant features with the largest activations are preserved [**?**], [1]. This layer is considered to be "non-trainable" because it does not use any adjustable parameters in it's transformation, only a fixed procedure that cannot be updated.

4) **Flattening**

5) **Concatenation**

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. Multilayer Perceptron Features

The features for this branch 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 a 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:

$$\text{TDE}_j[s] = \sqrt{\frac{1}{Q} \sum_{i=n}^{n+Q} s[i]^2} \tag{12}$$

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 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:

$$\text{ZXR}[s] = \frac{1}{2} \sum_{i=i}^{M-1} \left| \text{sign}\big(s[i]\big) - \text{sign}\big(s[i-1]\big) \right| \tag{13}$$

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:

$$\text{TCM}[s] = \frac{\sum_{i=0}^{M-1} i \big| s[i] \big|}{\sum_{i=0}^{M-1} \big| s[i] \big|} \tag{14}$$

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:

$$\text{ACC}_k[s] = \frac{\sum_{i=0}^{M-k-1} s[i]s[i+k]}{\sqrt{\sum_{i=0}^{M-k-1} s^2[i]} \sqrt{\sum_{i=0}^{M-k-1} s^2[i+k]}} \tag{15}$$

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]:

$$\text{MFCC}_k[B] = \sqrt{\frac{2}{R}} \sum_{i=1}^{R} \log\left(B[i]\right) \cos\left(\frac{k(i - \frac{1}{2})\pi}{R}\right) \tag{16}$$

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:

$$\text{FCM}[\widetilde{s}] = \frac{\sum_{i=0}^{M-1} i\left|\widetilde{s}[i]\right|}{\sum_{i=0}^{M-1} \left|\widetilde{s}[i]\right|} \tag{17}$$

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 first of the two modalities that we choose to represent each data sample.

B. Convolutional Network Features

The features for this branch are the entries in a standard 2D spectrogram representation of a waveform. A spectrogram maps the energy distribution of a signal as a function of time and frequency. It 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.
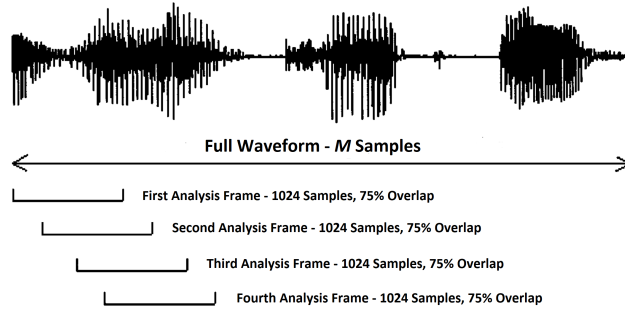

Fig. 2. Analysis frames in relation to a full waveform. This figures has been adapted from [7].

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:

$$S = \left(\mathbb{W}A^T\right) \odot \left(\mathbb{W}A^T\right)^* \tag{18}$$

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 $2048 \times k$.

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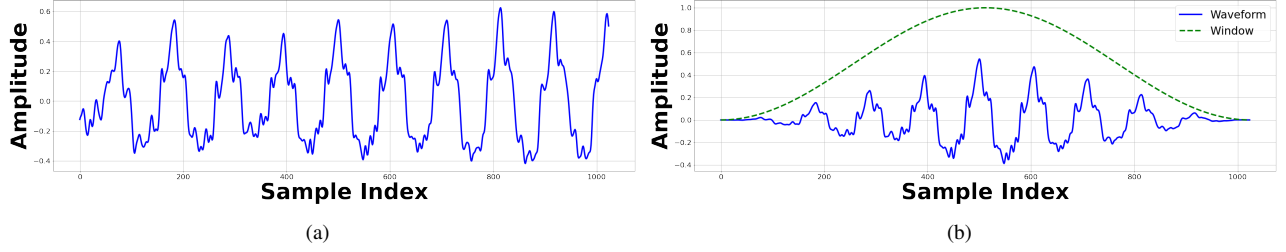
Fig. 3. (a) A standard time-series analysis frame made up of 1024 samples and (b) The same frame with a standard Hanning Window Applied to it. This sample is taken from a bowed violin playing an $A4$ note.

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 CNN branch of the HNN. This is the second of the two modalities that we choose to represent our data sample.
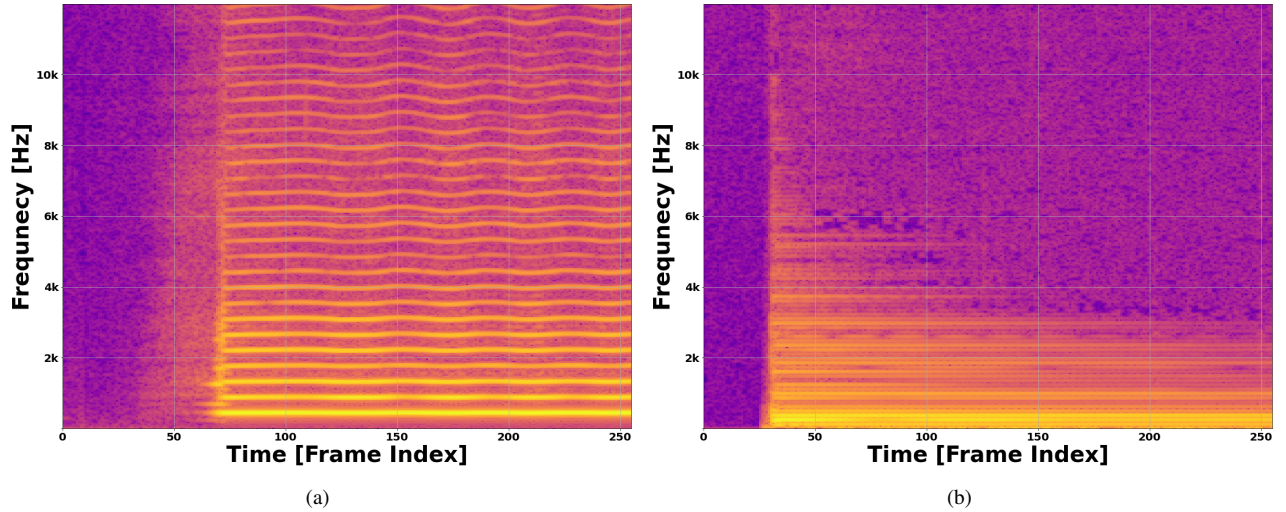

(a)   (b)

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

C. The Perceptron Branch

The perceptron branch is an pictured on the left side of Fig. (5) and is an implementation of multilayer perceptron (MLP) neural network. The MLP processed input by transforming activations through repeated layers of neurons, modeled as vectors of floating-point numbers. Each neurons in a given layer by a linear combination of every neuron in the previous layer. Practical implementations introduce the addition of a constant value to each neuron (called a *bias*), and some form or non-linear activation function [1], [2]. Transformations of this type are called *dense* layers, which when compounded can model that highly non-linear decision boundaries than arise in real world problems [3].

The input to this branch is a vector of features, described in section (III-A), which are transformed through two dense layers, with 64 neurons each. Passing through these dense layers allows these inputs to be handled just like any other multilayer perceptron model, which have shown experimental success in the last few decades. We choose to implement one here, enabling the training process to learn parameters that can effectively minimize the cost function and sucessfully classify inputs. The output of the last dense layer is provided to the concatenation layer, which merges it with the output of the convolution branch.

D. The Convolution Branch

The Convolution branch is pictured on the left side of Fig. (5) and is an implementation of a convolutional neural network (CNN), connected to a multilayer perceptron (MLP). This branch accepts 2D image-like input in the form of a spectrogram, described in section (III-B). The order and organization of the layers in this branch of the model was inspired from Oxford's *VGG-16* network architecture [1]. VGG-16 model uses successive 2D convolutions to identify characteristic features, followed by 2D pooling layers to reduce the size of the activations. Unlike VGG-16, our model does not use padding to keep the image resolution after each convolution layer, and instead the resolution of the image is dropeed with every layer.

The dominant structure in our convolution branch is the *convolution layer group* with combines three layers in succession:

1) **2D Convolution Layer** - 32 features feature maps, 3×3 kernel size, 1×1 kernel stride, ReLU activation function.
2) **2D Convolution Layer** - 32 features feature maps, 3×3 kernel size, 1×1 kernel stride, ReLU activation function.
3) **2D Maximum Pooling Layer** - 3×3 kernel size, 1×1 kernel stride, Identity activation function

The combination of layers in this order allows for the convolutions to detect characteristic features in the spectrogram, while the pooling allows for minor positional in variance in those features. Additionally, the output of the layer group is smaller is size, causing each layer's activation to take up less memory, and allowing only the most prolific features of each input to survive. At the end of the final layer group, the remaining 2D image contain only a few pixels, and the activation are flattened, by reorganizing each sample such that it is a 1D array. The 1D array is appropriately shaped for processing in the branche own MLP structure. The output of the last dense layer of this MLP is provided to the concatenation layer, which merges it with the output of the Perceptron branch.
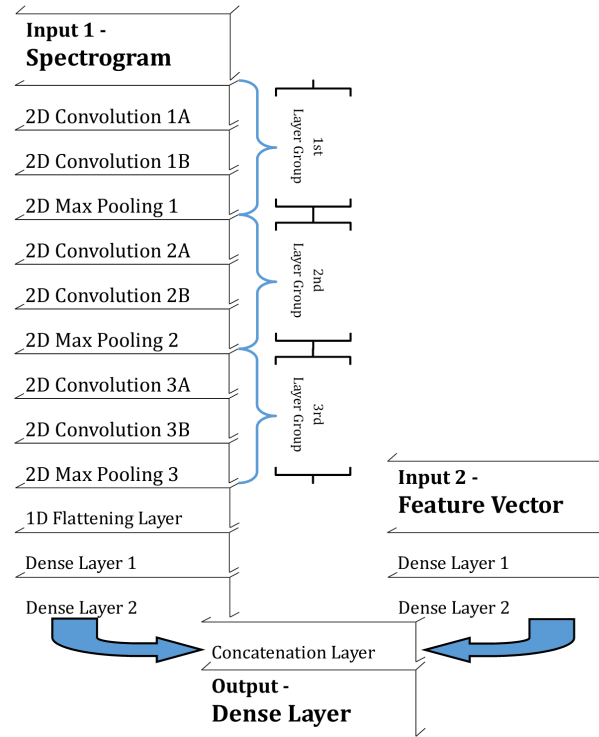
E. Multirepresentation Learning

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

The architecture of the HNN enables us to perform classify the contents of digital audio files as musical instruments through *multimodal* learning. The combined architecture, using Google's *Tensorflow* library for Python allows the model to be trained conveniently as a single object [13]. This means that although the HNN is composed of two separate branches, all parameters and layers in the network are included with each back-propagation pass. This HNN contains 319,365 trainable parameters.

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]. The training subset does contain corresponding labels, but they are withheld from the trained model such that it can make unbiased predictions. 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:

$$\text{Accuracy} = \frac{TP + FN}{TP + TN + FP + FN} \tag{19}$$

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

- **Precision Score** - is the ratio of correctly identified 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:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{20}$$

  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 identified 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:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{21}$$

  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:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{22}$$

  $F1$ 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 the entries of 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. Cross Validation 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 bypass the concatenation layer, and are 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 MLP models individually. Cross validation was performed on each model using a different set of random splits from the same data set. The reported scores are averged over all samples in each training data subset.
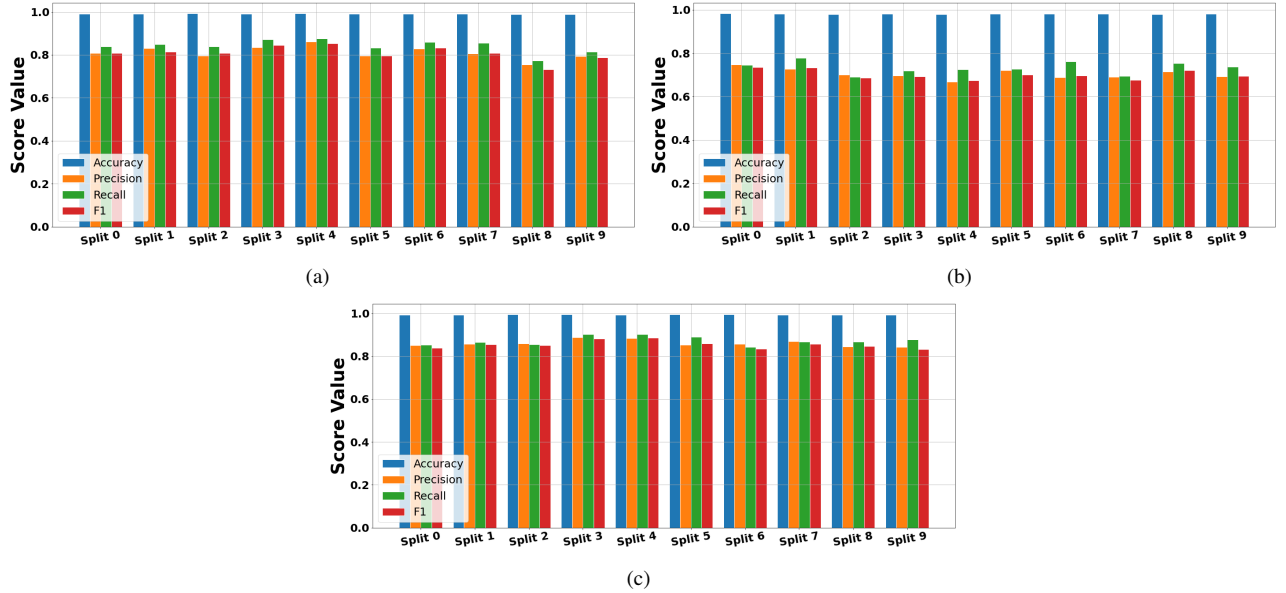

Fig. 6. Performances of the (a) CNN + MLP with spectrogram input, (b) MLP with feature-vector input, and (c) the HNN with spectrogram and feature-vector input

We observe the classification scores for each of the three model variants in Fig. (6). The CNN + MLP variant in (6-a) shows metric scores consistently averaging close to 0.8. with a few splits above or below this value. Similarly, the MLP variant in (6-b) shows scores all consistently close to 0.7. This indicates that each input mode and complementary architecture allows for the optimization of a set of parameters to enable reasonable classification performance on unseen samples. Both models do exhibit fluctuation in the range of scores between each fold, which indicates that each model is vulnerable to small changes in initial conditions.

The HNN architecture variant in (6-c) shows improved performance when compared to either the CNN or MLP variant alone. Where the single input mode models shows performance scores averaging around 0.8 and 0.7, in (6-a) and (6-b) respectivly, the HNN boasts a higher average, around 0.84 for both precision and recall. We also observe a lower variance between the performances of each split, indicating a more stable architecture that is less susceptible to changes in initial condition.

We also choose to examine and compare the normalized confusion matrices of the three modal variants. In Fig. (7-a) we see a confusion matrix for the CNN + MLP, which shows a strong main diagonal, indicated by the dark reds, oranges, and yellows which indicated a high-level of correct classifications. We do observe indexes such as $C[0, 5]$, $C[0, 15]$, $C[7, 27]$, and $C[28, 24]$ that contain unusually large entries indicated by the bright blue color. In Fig. (7-b) we see a weaker diagonally dominant matrix

for the MLP, which displayed even more misclassified samples, spread through the matrix. Note how each confusion matrix represents the average of 10 confusion matrices produced from cross validation.

Finally, we observe the normalized confusion matrix of the HNN model in Fig. (7-c). This matrix is the most-diagonally dominant, showing only a few commonly confused classes. The smaller number of mis-classifications in this model indicates a greater performance, and a generally stronger classifier. We observe $C[0, 15]$, $C[7, 27]$, and $C[27, 7]$ to have unusually large entries also indicated by the bright blue pixels. Consulting the list of instrument classes reveals that class 0 and 15 represent *alto flutes* and *flutes* respectively. Classes 7 and 27 represents *bass trombones* and *tenor trombones*. These pairs of instruments are similar in build and in timbre, meaning that they likely have similar properties in feature space, and that a human listener may also show difficulty differentiating them.

E. Practical Example

As a practical demonstration of the classification ability of the Hybrid Neural Network, we choose to test the model on an unlabeled digital audio file, ”*zcombo.wav*”. This audio file contains a waveform that is generated through

I'll need your help to introduce the chaotic Synth files, and address how they are generated, and why we use and ”care” about them
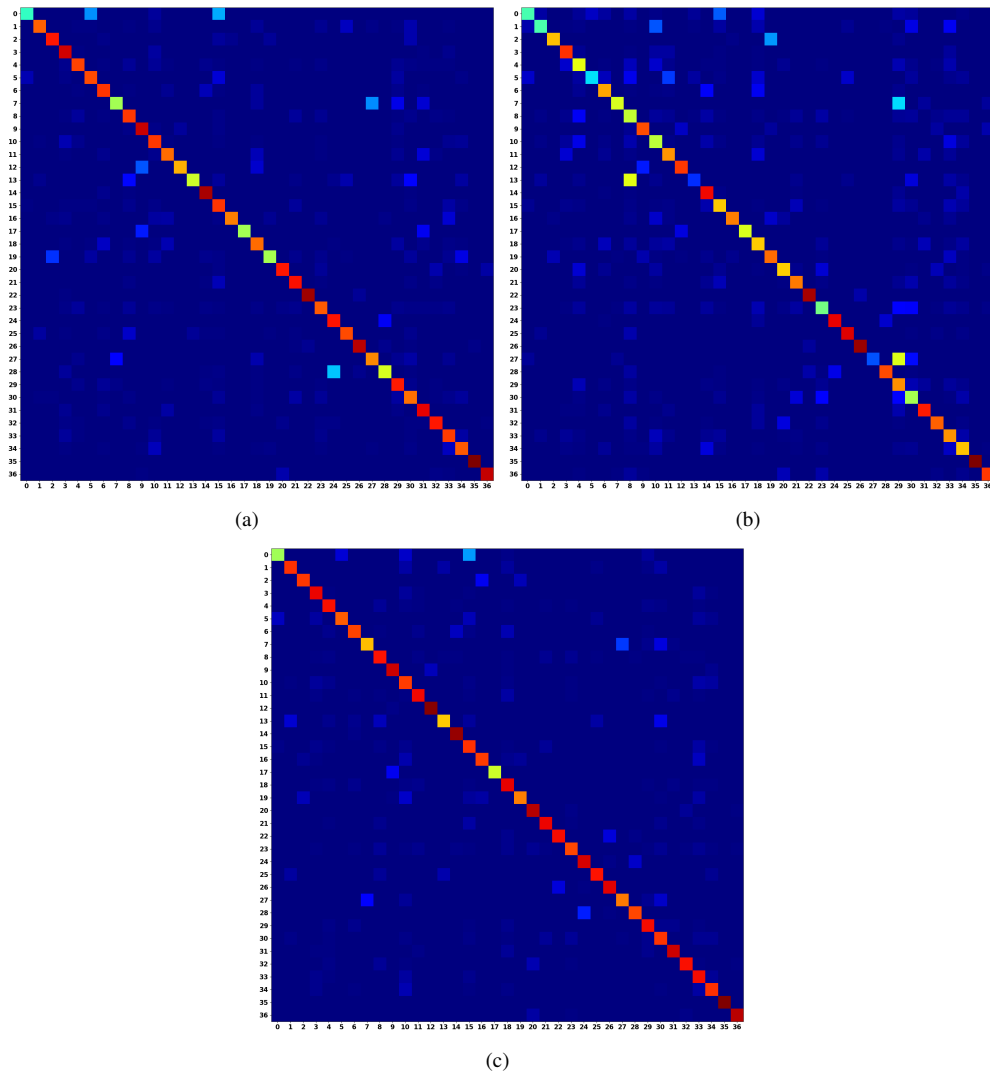

(a)

(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

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 data. 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 such a medicine, finance, and other media classification.

The results of cross validation indicate that we have constructed a hybrid neural network that accept two input modes, and can effectivly classify the contents of a digital audio file as one of 37 different musial instruments. The design of this HNN enables the input of two non-compatible modal representations of a sample to be processed and combined to form a single output prediction. We find that this model demonstrates better and more consistent performance at musical instrumental classification when compared to that of either the constituent CNN or MLP models. By transforming each input sample into multiple representations of itself, the HNN learns a more robust set of parameters that can more thoroughly differentiate each class. While some common misclassfications are present, many of them are due to the related nature of musical instruments, which may also cause similar confusion in human listeners.

I sure cannot write a conclusion to save my life

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.