Musical Instrument Classification Using a Hybrid Neural Network

Landon H. Buell[1] and Kevin M. Short[2]
[1]lhb1007@wildcats.unh.edu [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 the sound of a single note performed by some musical instrument and the task of matching the categorizing the file based on what instrument the contents 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 reliable 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 collection of inputs, x, called *features*, to be transformed into a collection of outputs, y, called *predictions* through a set of adjustable values, Θ, called *parameters* [1], [2], [14]. In machine learning audio classification, much research is devoted to engineering and developing an appropriate ensemble of features that can be used to improve classification ability [3], [7], [9], [16]. Since the nature of classification tasks and audio datasets may differ greatly from any other - each task must often develop a unique model and combination of features that will allow for the best possible performance [14]. For musical instrument classification we have produced features that will represent the contents of each audio file using two different *modalities*. Consequently, we must craft the neural network structure so that it may simultaneously accommodate these different input forms. 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 forms that are each different representations of the same data sample, at the same time. Data sets that include or can be transformed into multiple forms such as *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 2-dimensional spectrogram and a 1-dimensional vector of features from the same audio file. 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 singlemodal learning methods [6].

To account for this multimodal input, we have constructed a *hybrid neural network* (HNN) that utilizes two different input layers where each receives its prescribes input format and is handled by an appropriate layer chain. The 2D spectrogram image is processed with a Convolutional Neural Network (CNN) which uses layers of 2D convolution and 2D pooling to generate a feature map, is then flattened and transformed by repeated dense layers. This behavior is visualized on the left side of Fig. (5). The 1D feature vector 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 dense 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 its *activation* [5]. Neurons are often organized into groups called *layers*, that interact with other neurons in other layers through mathematical transformations such as matrix multiplications or convolutions [2]. Because of this interaction, each layer (l) in a network can be considered as function, $f^{(l)}$ that accepts an array (vector,matrix,tensor,etc.) 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 chain-like structure of layers that describes the flow of information and allows us to model a neural network as a directed computational graph where each node is a layer that performs an operation [1], [2]. In a feed-forward network with L layers, inputs $x^{(0)}$ and outputs $x^{(L-1)} = y^*$, 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 is often referred to as its *architecture* and chooses the *hypothesis space* of the model. This is the set of all possible functions that the network can choose as a solution [2], [3]. For deep neural networks this function chain can be dozens of layers long, contain upwards of millions of parameters, and may enable the model to cover a much more diverse set of solutions [1]. By carefully choosing the architecture to compliment the nature of the problem, we build a preference of solutions into the model which enables it to perform better [2].

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, f_a and f_b that receive inputs $x_a^{(0)}$ and $x_b^{(0)}$ respectively. Each chain uses a particular arrangement of layer to form a sub-graph that compliments each type of input data. The activations are processed and transformed into a compatible 1-dimensional format before they are combined at a designated hidden layer, f_c^{γ} and then transform again to produce the model's final prediction . 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 :

$$\begin{tikzcd}
x_a^{(0)} \arrow[r] & f_a^{(0)} \arrow[r] & ... \arrow[r] & f_a^{(\alpha)} \arrow[dr] & & & & \\
& & & & f_c^{(\gamma)} \arrow[r] & ... \arrow[r] & f_c^{(L-1)} \arrow[r] & y^* \\
x_b^{(0)} \arrow[r] & f_b^{(0)} \arrow[r] & ... \arrow[r] & f_b^{(\beta)} \arrow[ur] & & & &
\end{tikzcd} \tag{3}$$

The combination of two sub-graphs, f_a and f_b, into a third sub-graph, f_c, allows us to generate a single prediction, y^* based on the behavior of both inputs. This property means that all layers in the network are part of one aggregate computational graph, one training process, and collectively form one hypothesis space. We detail the structure and function of each layer in the neural network in section (III-D) and (III-C).

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 single 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. We expect features drawn from different samples of the same musical instruments to have very similar properties while features drawn from samples of different musical instruments to exhibit non-similar properties. In any learning algorithm, the role of feature engineering is to translate the raw time-series 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 waveform directly and instead will rely solely on these features. In a classification model that uses p unique predictors for each sample will typically combined into a single vector-like object which is 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 neurons in the output layer, each one representing a particular class. These output activations which are transformed such that the sum (or $L-1$ norm) of them is identically 1 so that they are collectively treated as a discrete probability density [2], [3]. The neuron with the highest activation value indicates the class that the models predicts in input sample originated from. For classification of musical instruments, each neurons represents a musical instrument type. We use 37 classes of instruments encapsulating woodwinds, brass, strings, mallet percussion, and synthesizers.

C. The Cost Function

Consider a neural network that uses p features, k classes, and b samples per batch is trained by providing inputs $X \in \mathbb{R}^{(b \times p)}$, with a corresponding set of labels $Y \in \mathbb{Z}^{(b \times k)}$. 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, which will lead to poor classification performance. 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:

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

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 learning model and task, but typically a multiple - category classification tasks uses a *categorical cross-entropy* (CXE) function [1], [2]. The CXE cost for a single sample is defined as [2], [14]:

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

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

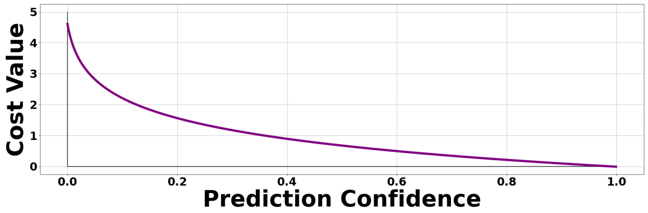


Fig. 1. Categorical Cross Entropy Cost Function given values for $y_j^* \in [0, 1]$

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 [1], [14]. Since we can quantify the difference between y^* and y with a cost function, we 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 a consistently low cost function value across multiple samples [1], [2]. Note that we optimize the cost function under the assumption that doing so will also improve the classification performance when is a concept 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 repeatedly (i.) computing the gradient vector 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). For each iteration, we compute the gradient of the cost function, $\nabla_{\Theta} J$ through a process called *backwards propagation* [1]. This algorithm 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].

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

We introduce a parameter α as the *learning rate* of the model. This scales each element in the gradient vector as t control the ”step size” taken with each iteration [1], [2], [14].

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).

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

ADAM provides a much more powerful and stable optimization algorithm at a higher computational cost [1], [13]. 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 project [1], [2].

E. Layers Used in this Classification Model

As mentioned in sec. (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. The combination and arrangement of these layers define the model's hypothesis space which is the set of functions that the network can use as a solution. For this classification task we employ five different layer types detailed below. We also describe the forward-pass process of each layer and in Fig. (5) and 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]. It is composed of a collection of artificial neurons,where the activation of any neuron, is computed through a weighted linear-combination of every neuron in the previous layer with a *bias* added as well [1], [2], [13]. We can model this transformation as :

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

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. This transformation models an multidimensional affine-transformation which enables this layer to learn roughly linear decision boundaries between adjacent layers of neurons.

2) **2-Dimensional Convolution Layer** Convolution Layers emerged from studying the brain's visual cortex, and have been used for image recognition for decades [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 of arrays A and B is given by:

$$C[i,j] = (A * B)[i,j] = \sum_{y}\sum_{v} A[i,j]B[i-u, j-v] \tag{9}$$

Where A, B, and C are also 2D arrays themselves. In a neural network, we implement this function by creating a series of K weighting matrices, called filters, each of size $m \times n$, and 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 product 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 [1].

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 activations of the k-th filter $x_k^{(l)}$ using filter $W_k^{(l)}$ and bias $b_k^{(l)}$ as [2].

$$x_k^{(l)}(i,j) = b_k^{(l)} + \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} W_k^{(l)}(i,j)x^{(l-1)}(i-u, j-v) \tag{10}$$

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 more computational efficient size.

Pooling layers are typically placed 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** A flattening layer is used to rearrange the shape of an array of activations without actually changing the values stored within them [13]. In particular, we use this layer to transform the 2D output of the convolution and pooling layers into 1D activations which can be passed into a dense layer. For an array of activations , $x^{(l-1)} \in \mathbb{R}^{(m \times n)}$, a flattening layer $f^{(l)}$ transforms them such that $x^{(l)} \in \mathbb{R}^{(1 \times mn)}$. For example, for 2D input $x^{(l-1)}$, we would return a 1D output $x^{(l)}$:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \rightarrow \begin{bmatrix} a & b & c & d & e & f \end{bmatrix} \tag{11}$$

This layer is also considered to be "non-trainable" because it does not use any adjustable parameters.

5) **Concatenation** The concatenation layer combines the activations of two similarly-dimensional arrays by concatenating them along a desired axis. In the layer graph shown in Eq. (3) this takes the place of layer f_c^{γ}. The output of the two previous layer chains are merged here before being fed to the output layer chain. This layer is what makes the aggregation of each subgraph possible and how we are able to generate one prediction from two input modalities. For inputs $x_a^{(l-1)}$ and $x_b^{(l-1)}$ this layer will return an output $x_c^{(l)}$:

$$\begin{bmatrix} a & b & c \end{bmatrix} \& \begin{bmatrix} d & e & f \end{bmatrix} \rightarrow \begin{bmatrix} a & b & c & d & e & f \end{bmatrix} \tag{12}$$

This layer is also considered to be "non-trainable" because it does not use any adjustable parameters.

Formal implementations of the dense layer and 2D convoltional layer types introduce non-linearity into the hypothesis space by applying an *activation* function to Eq. (8) and Eq (10) [2], [13]. For this classification network, we choose to use the *ReLU* activation function, defined as:

$$ReLU\big[x^{(l)}\big] = \max\big(0, x_i)\big) \tag{13}$$

for each element $x_i \in x^{(l)}$. Successive use of affine and convolutional transformations combine with non-linear activation functions allow a neural network to model the increasingly complicated decision-boundaries that arise in real-world problems [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. Multilayer Perceptron Features

The features for this branch are derived from the time-domain and frequency-domain representations of a waveform. Each feature explored represents the values computed from and 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 are assumed to 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:

$$\text{TDE}_j[s] = \sqrt{\frac{1}{Q}\sum_{i=n}^{n+Q} s[i]^2} \tag{14}$$

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:

$$\text{ZXR}[s] = \frac{1}{2}\sum_{i=i}^{M-1} \Big|\text{sign}\big(s[i]\big) - \text{sign}\big(s[i-1]\big)\Big| \tag{15}$$

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:

$$\text{TCM}[s] = \frac{\sum_{i=0}^{M-1} i\big|s[i]\big|}{\sum_{i=0}^{M-1} \big|s[i]\big|} \tag{16}$$

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:

$$\text{ACC}_k[s] = \frac{\sum_{i=0}^{M-k-1} s[i]s[i+k]}{\sqrt{\sum_{i=0}^{M-k-1} s^2[i]}\sqrt{\sum_{i=0}^{M-k-1} s^2[i+k]}} \tag{17}$$

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]:

$$\text{MFCC}_k[B] = \sqrt{\frac{2}{R}} \sum_{i=1}^{R} \log\left(B[i]\right) \cos\left(\frac{k(i-\frac{1}{2})\pi}{R}\right) \tag{18}$$

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:

$$\text{FCM}[\widetilde{s}] = \frac{\sum_{i=0}^{M-1} i\big|\widetilde{s}[i]\big|}{\sum_{i=0}^{M-1} \big|\widetilde{s}[i]\big|} \tag{19}$$

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 spectral signature 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* (STAF) [7], [14]. An STAF is a subsets of a waveform in the time-domain. We choose each frame to contain $N = 1024$ samples, with a 768 sample overlap between adjacent frames.

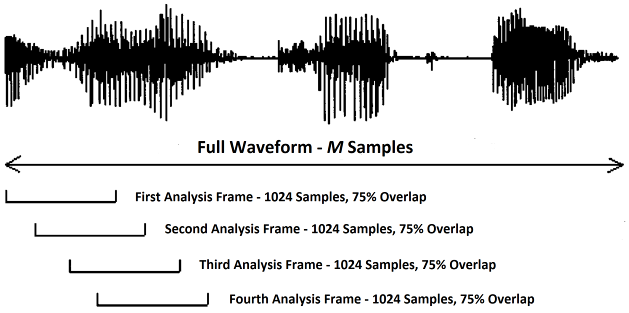
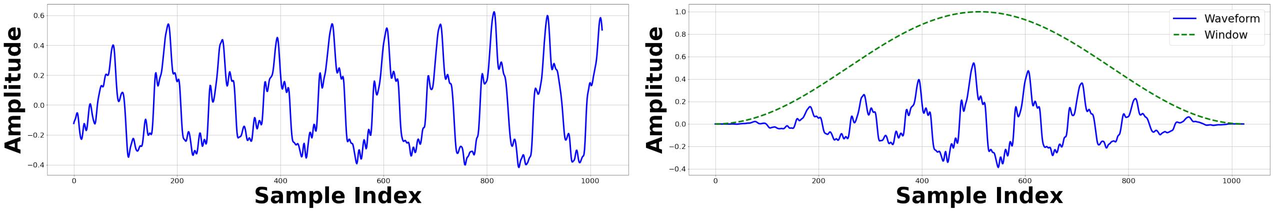

Full Waveform - *M* Samples
First Analysis Frame - 1024 Samples, 75% Overlap
Second Analysis Frame - 1024 Samples, 75% Overlap
Third Analysis Frame - 1024 Samples, 75% Overlap
Fourth Analysis Frame - 1024 Samples, 75% Overlap

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 $k \times N$, such that each row is a frame. We apply a Hanning window of length N to each row of A which helps to eliminate 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 for a higher frequency-space resolution [12]. The spectrogram, S, is produced by computing:

$$S = \left(\mathbb{W}A^T\right) \odot \left(\mathbb{W}A^T\right)^* \tag{20}$$

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 data set rarely have fundamental frequencies that extend beyond 6 kHz. Similarly, we rarely find much energy or characteristic behavior above the 12 kHz 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. Since the audio files were all sampled at 44.1 kHz, this nearly quarters the number of rows in the spectrogram. We have also choosen to zero pad or crop the matrix S so that there are exactly $k = 256$ columns. This combination of choices drastically reduces the width and complexity of the model at the entry layer saving significant computational resources. 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.

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 dense layer types explored in sec. (II-E.1). 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 in each.

FINISH THIS!

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 it's own 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

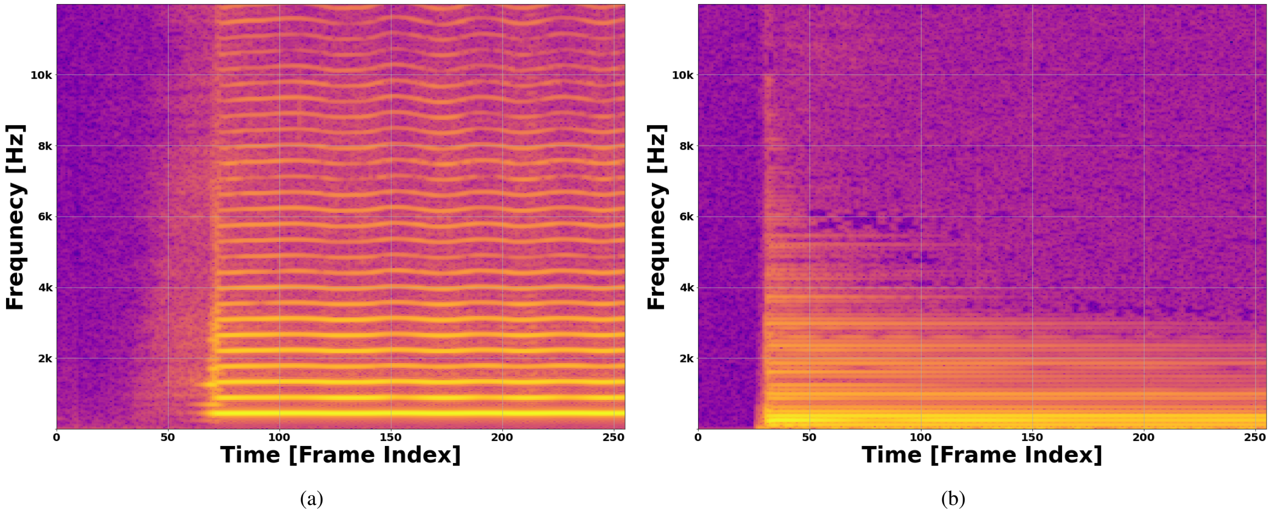


Fig. 4. Spectrograms of (a) an alto saxophone playing an $A4$ and (b) an acoustic guitar playing a $B2$

VGG-16 network architecture [1]Add VGG citation!. 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 dropped 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

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, on K unique models.

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 collection of learned parameters, Θ that represent a possible outcome if trained on the given subset [1], [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.

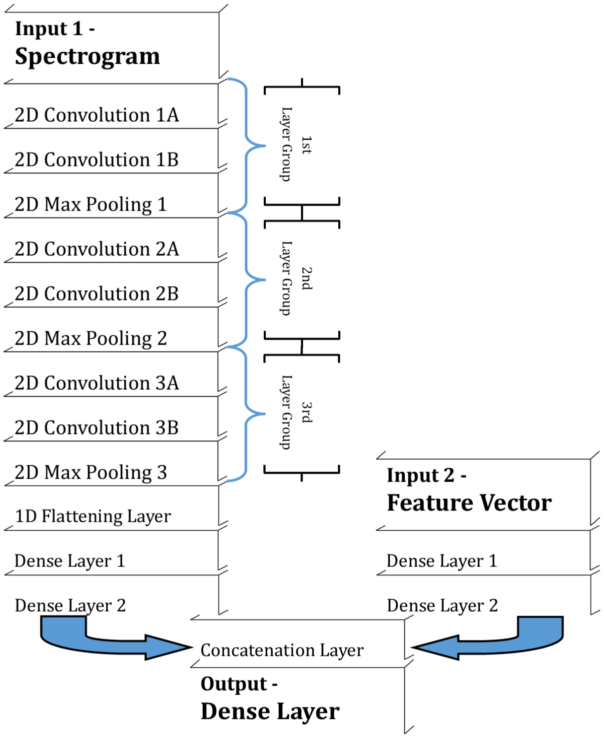


Fig. 5. Architecture for Hybrid Neural Network model, implemented with Tensorflow [13]

Across each fold, we expect to observe a reasonably high, as well as consistent performance. A model that does not perform well on all of the splits indicates a poor choice or architecture or features, and a model that performs inconsistently across the splits indicates a model that suffers from high-variance, where small changes in initial conditions can drastically affect it's classification abilities [3]. When we see consistent and strong performance across multiple splits, we develop confidence that we have a model that can train effectively, and generalize appropriately [1].

B. Performance Metrics

We choose the following metrics to evaluate the performance of our HNN. All metrics are defined 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 choices made. This is especially true when classes have non-uniform representations in the data set. The accuracy score of a classification algorithm is given by:
$$\text{Accuracy} = \frac{TP + FN}{TP + TN + FP + FN} \tag{21}$$
Accuracy is bound between 0 and 1 with a higher score indicating better performance.
- **Precision Score** - is the fraction of retrieved samples that were relevant. This means that the higher the score, the more *specific* a model is when categorizing. The precision score of a classification algorithm is given by:
$$\text{Precision} = \frac{TP}{TP + FP} \tag{22}$$
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 fraction of relevant samples that were retrieved. The higher the score, the *sensitive* a model is to a particular category. The recall score of a classification algorithm is given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{23}$$

  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:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{24}$$

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

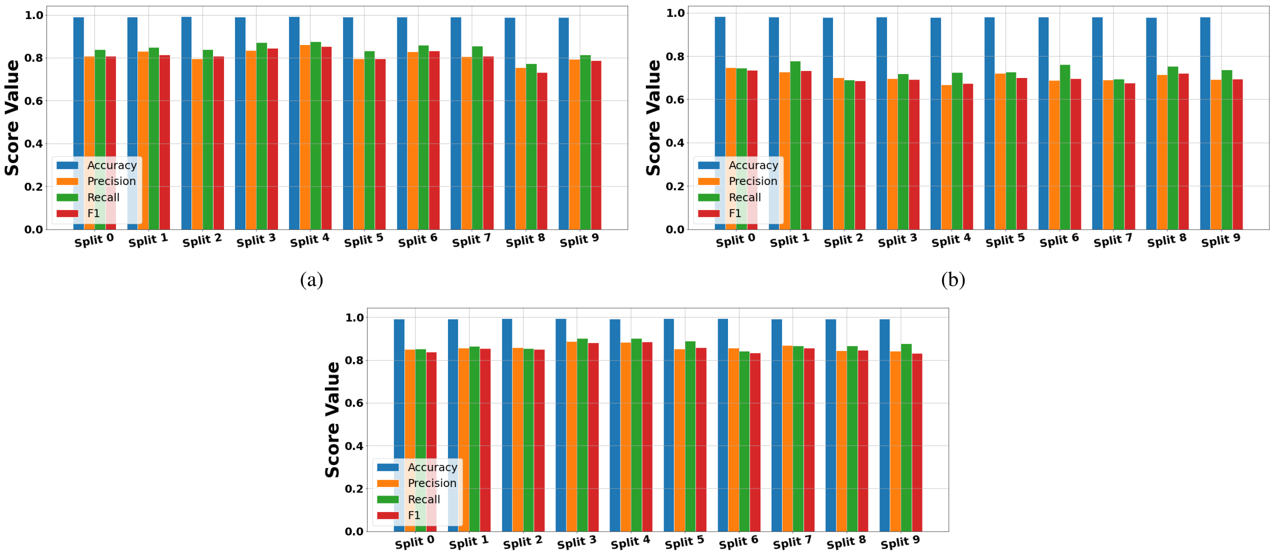
C. Confusion Matrices

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.



(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

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.

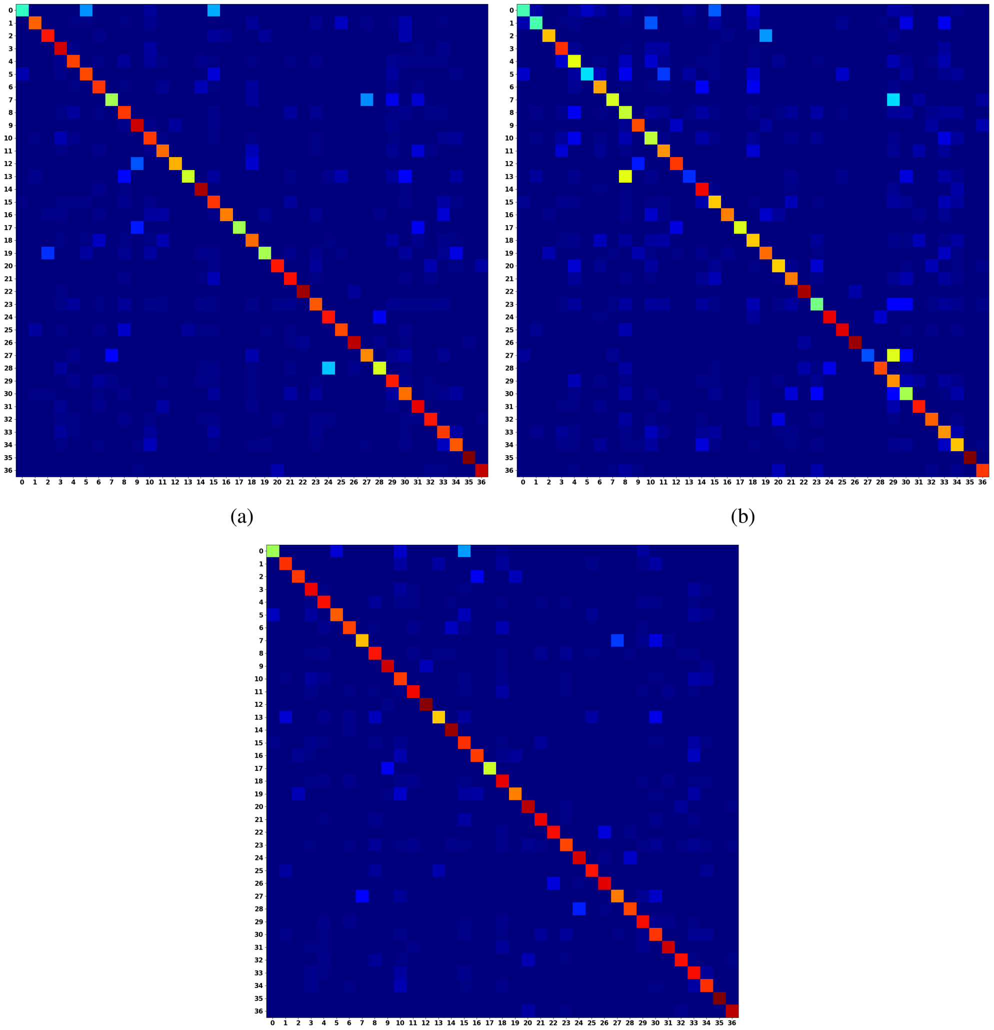
We also choose to examine and compare the normalized confusion matrices of the three modal variants. In Fig. (7-a) we see a confusion matrix for the CNN + MLP, which shows a strong main diagonal, indicated by the dark reds, oranges, and yellows which indicated a high-level of correct classifications. 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.