1 The Neural Network

1.1 An Introduction to Neural Networks

The task of algorithmically matching sound files to musical instrument classes in exceedingly difficult through conventional computer programming techniques. Because of the complexity of sound waves, brain's interpretation them, the challenge arises as to how to build some sort of program that could function at a level above general procedural or explicit instructional rules. Rather than *hard-coding* a set of conditions or parameters for a classification program, we seek an architecture that allows for a computer to *learn* and change and update itself as it is presented with more data. Such an algorithm exists in the form of a *Neural Network* [2, 3, 6].

A neural network is a mathematical function that seeks to produce an approximation F^*, of some usually unknown function F . For audio recognition, F is some function, or series of functions, that occurs in the brain which allows for a listener to take the sensation of audio, and map it to a known musical instrument. For a neural network, a model must construct an approximate function F^* using a set of parameters Θ, which allows the model to map a series of inputs, x (called *features*) to a series of outputs, y (called *predictions*) [3, 4, 16].

Former YouTube video classification team lead, and current machine learning consultant, Aurelien Geron writes about the relationship between biological brains and mathematical neural networks [1]:

> Birds inspired us to fly, burdock plants inspired velcro and nature has inspired many other inventions. It seems only logical, then, to look to the brain's architecture for inspiration on how to build an intelligent machine.

The result of such an analogy is a computer program that is reminiscent of the brain.

Over the course of their lives, humans will learn to map sounds to sources almost effortlessly. Given an example, and the appropriate label, humans can do this with very reasonable accuracy over a wide array of sounds [12, 17]. We can simplify the idea of humans recognizing sound as some function or operation that occurs within the brain, accepting an input sound, and produces an output label. Similarly, a neural network can be constructed, presented with *features* of multiple, labeled sound waves and learns a set of parameters Θ that allows for the mapping to a source.

For this particular project, a neural network has been constructed to perform a *classification* task. A classification task involves mapping the input data, x to one of k possible *classes*, each represented by an integer, 0 through $k-1$. Each of the k classes represents a particular musical instrument that could have produced the sound-wave in the audio file.

1.2 The Structure of a Neural Network

A Neural Network is simply a model of a mathematical function, composed of several smaller mathematical functions called *layers* [3, 8]. Each layer represents an operation that takes some real input, typically an array of real double-precision floating-point numbers, and returns a modified array of new double-precision floating-point numbers. The exact nature of this operation can be very different depending on the layer type or where it sits within the network. It is this process of transforming inputs successively in a particular order until an output is attained [1, 8]. This output encodes the models final "decision" given a unique input.

Other than inspiration from the brain, Neural Networks are name *networks* because of their nested composition nature. The model can be represented by a linear or acyclic computational graph which successively maps exactly how the repeated composition is structured. In a *feed-forward network*, information is passes successively in one direction. For example functions could be chained together such as [3]

$$F(x) = f^{(L-1)}\left[f^{(L-2)}\left[...f^{(1)}\left[f^{(0)}\left[x\right]\right]...\right]\right] \tag{1}$$

Where each function $f^{(l)}$ represents a layer of the network model. The number of layers in a neural network is referred to as the network *depth*. The dimensionality of each layer is referred to as the network *width* [1, 8].

A network model that contains L unique layers is said to be an L-Layer Neural Network, with each layer usually indexed by a superscipt, 0 through $L-1$. Layer 0 is said to be the *input layer* and layer $L-1$ is said to be the *output layer*. The function that represents a layer (l) is given by

$$f^{(l)} : x \in \mathbb{R} \to y \in \mathbb{R} \tag{2}$$

The value of x can also be index by layer, we call the array $x^{(l)}$ the *activations* of layer l [3, 8].

The model is recursive by nature, the activations from one layer, $l-1$, are used to directly produce the activations of the next successive layer l. Thus eqn. (2) can be alternatively written as:

$$f^{(l)} : x^{(l-1)} \in \mathbb{R} \to x^{(l)} \in \mathbb{R} \tag{3}$$

The array of activations, $x^{(0)}$, is the raw input given the neural network, most commonly called *features*. Conversely, the activations $x^{(L-1)}$ are commonly called the network *output* [1, 4, 8].

1.3 Layers Used in Classification Neural Network

As stated previously, a neural network is composed of a series of functions that are called successively to transform features (an input) into a prediction (an output). As shown in eqn. (3), each function feeds directly into the next as to form a sort of computational graph [3].

Typically, a layer function can be divided into two portions: (i.) a Linear transformation, with a bias addition, and (ii.) an element-wise activation transformation. This activation function typically is a non-linear function which allows for the modeling of increasingly complex decision-boundaries. Step (i.) is usually in the form of a matrix-vector equation:

$$z^{(l)} = W^{(l)} x^{(l-1)} + b^{(l)} \tag{4}$$

Where $W^{(l)}$ is the *weighting-matrix* for layer l, $b^{(l)}$ is the *bias-vector* for layer l, $z^{(l)}$ are the *linear-activations* for layer l and $x^{(l-1)}$ is the final activations for layer $l-1$. Step (ii.) is usually given by some *activation function*:

$$x^{(l)} = \sigma^{(l)}\left[z^{(l)}\right] \tag{5}$$

Where $x^{(l)}$ is the final activations for layer l and $z^{(l)}$ is given in equation (4). $\sigma^{(l)}$ is some activation function, which is often use to enable the modeling of more complex-decision boundaries.

Below, we discuss and describe the types of layer functions that are used to produce the classification model in this project.

1.3.1 Dense Layer

The Linear Dense Layer, often just called a *Dense* Layer for short, was one of the earliest function types used in artificial neural network models. A dense layer is simply composed of a layer of *artifical neurons*, each of which holds a numerical value within it, called the *activation* of that neuron. This idea was developed back from McCulloch and Pitts' work [9], and was expanded upon by Frank Rosenblatt in 1957 [1].

We model a layer of neurons as a vector of floating-point numbers. Typically, it is required that the array be one-dimensional. Suppose a layer (l) contains n artificial neurons. We denote the array that hold those activations as $x^{(l)}$ and is given by:

$$\vec{x}^{(l)} = \Big[x_0, x_1, x_2, \dots, x_{n-2}, x_{n-1}\Big]^T \tag{6}$$

The activation of each entry is given by a linear-combination of activations from the previous layer, as outlined in eqn.(4) and eqn.(5).

Suppose the layer $l-1$ contains m neurons. Then the weighting-matrix, $W^{(l)}$ has shape $m \times n$, the bias-vector $b^{(l)}$ has shape $m \times 1$. Thus for a dense layer l, the exact values of each activation is given by [1, 8]

$$\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}^{(l)} = \sigma^{(l)} \left(\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,m-1} \\ w_{1,0} & w_{1,1} & \dots & w_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n-1,0} & w_{n-1,1} & \dots & w_{n-1,m-1} \end{bmatrix}^{(l)} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{m-1} \end{bmatrix}^{(l-1)} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix}^{(l)} \right) \tag{7}$$

or more compactly:

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

Eqn. (8) is generally referred to as the *dense layer feed-forward equation* [3].

1.3.2 2-Dimensional Convolution Layer

1.3.3 2-Dimensional Maximum Pooling Layer

1.3.4 1-Dimensional Flattening Layer

A flattening layer is used to compress an array with two or more dimensions down into a single dimension. For a flattening layer l, multidimensional activations in layer $l-1$ are compressed down into a single dimensional array. We can use function notation to express this as:

$$f^{(l)} : x^{(l-1)} \in \mathbb{R}^{(M \times N \times \dots)} \to x^{(l)} \in \mathbb{R}^{(MN \dots \times 1)} \tag{9}$$

The numerical value of each activation is left unchanged. For a layer with activation shape$N \times M$, the resulting activations are reshaped into $NM \times 1$ as shown in eqn (9).

Flattening Layer are most commonly used to prepare activations for entry into dense layer or series of dense layers. For example, 2D or 3D images are typically processed with 2D convolution, which may output a 2D or 3D array of activations. Those values are then flattened to 1 dimensions, which can then be passed into dense layers for further processing.

1.3.5 1-Dimensional Concatenation Layer

1.4 Training a Neural Network

A neural network's job is to produce an approximation of some function F, which we call F^*. The model must have a procedure for updating the parameters in Θ to allow for a reasonable approximation of F. To better understand this, we turn to Tom Mitchell's explanation of a learning algorithm [3, 11]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if it's performance at tasks in T, as measured by P, improves with experience E.

Without any external intervention, a model must update itself to improve it's performance at a give task a new information is presented to it. To do this, the model must be constructed with a training procedure in mind.

If we consider the set of parameters θ as a the concatenation, of each layer's weighting matrix and bias vector such that:

$$\theta = \left\{W^{(0)}, b^{(0)}, W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, ..., W^{(L-2)}, b^{(L-2)}, W^{(L)}, b^{(L-1)}, \right\} \tag{10}$$

We can consider each parameter is simply a float-point number. Each value for each number contribute to the output of the function. In increasing complicated neural networks, there can be upwards of hundreds of thousands,or even millions of elements in θ making a neural network a functions that exist is a very high dimensional parameter-space.

Last

1.5 Chosen Model Architecture

The success of a neural network algorithm is enormously dependent of the strength of the chosen features, rather than the quantity of available data [1, **?**, 7]. We have derived an appropriate set of inputs arrays *[See Section on Features]*, and constructed a neural network architecture to compliment these features.

The full classification neural network used for this project consists of two distinct entry points. This means that rather than presenting the network with one set of input data, X, we present the network with two different arrays, X_1 and X_2. Both arrays are a produce of the same audio file sound wave, and thus share a common training label. Each of the two entry points feed into their own, non-related set of layer functions. The second-to-last layer, the outputs from both neural networks are concatenated into a single dense layer, which is then passed into the output layer, which contains k neurons, one of reach unique class [8, 1]. A visual representation of this architecture can be found in fig. (1.5).

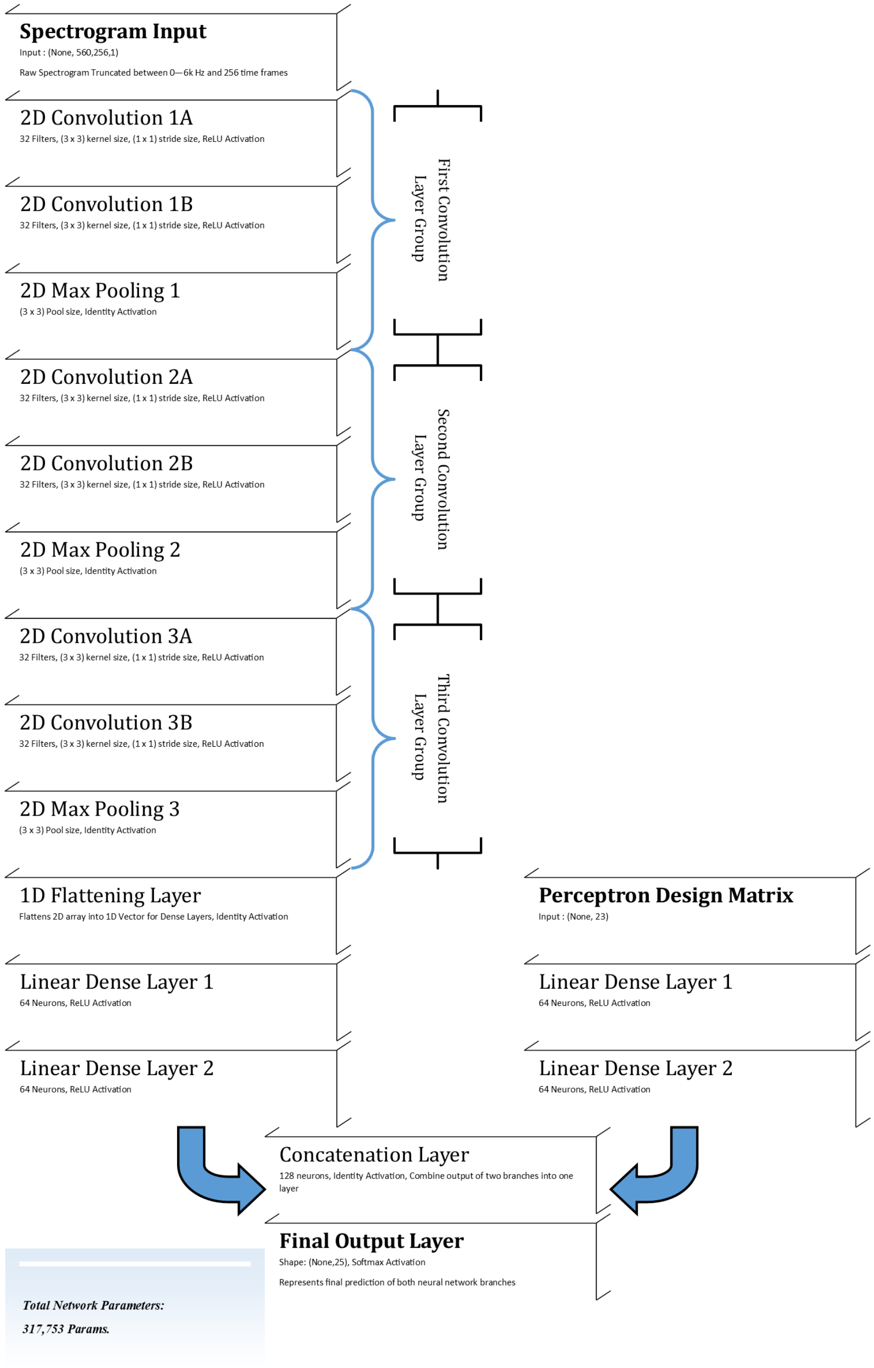

Figure 1: The developed architecture of the audio file classification neural network. The Left branch process an image-like input, the right branch processes a vector-like input. The activations are then merged, and then a single output is produced

1.5.1 The Spectrogram Branch

The spectrogram branch is pictured on the left side of fig. (1.5). A spectrogram is a representation of a sound wave, or signal by representing energy distributions as a function of *time* and *frequency* [17, 12, 5]. Details on the creating of the spectrogram can be found in the *features* section of this document.

The input layer of the spectrogram accepts a 4-Dimensional array. The axes, in order of indexing, represents (i) the size of the *mini-batch* of samples, (ii) the pixel width of each sample, (iii) the pixel height of each sample, and (iv) the number of channels in each sample. As a model hyper-parameter, we have chosen each batch to contain 64 samples. We have also chosen to truncate time and frequency axes (ii & iii) to contain 560 frequency bins, and 256 time-frames. Each image also contains a single channel, which make it gray-scale when visualized. We can denote the 4D shape of the input object into this branch as:

$$X_1 \in \mathbb{R}^{(64 \times 560 \times 256 \times 1)} \tag{11}$$

Any other shape will be rejected by the model, and an error is raised.

After the input layer assert the appropriate shape, the array X_1, which is a collection of 64 spectrograms, is passed into the first of three *Convolution Layer Groups*. These layer groups are inspired from the *VGG-16 Neural Network* architecture [citation needed]. Each convolution layer group in composed of three individual layers:

1. A 2-Dimensional Convolution layer, using 32 filters, a 3×3 kernel, a 1×1 step size, and a ReLU activation function,
2. A 2-Dimensional Convolution layer, using 32 filters, a 3×3 kernel, a 1×1 step size, and a ReLU activation function,
3. A 2-Dimensional Maximum Pooling layer using a 3×3 pooling size, and an Identity activation function

The Convolution layers convoluted over the middle two axes of the data (over space and time in the spectrogram).

By grouping layers in this structure, each of the 32 filters passes over a 3×3 area, and then again. The result is

1.5.2 The Perceptron Branch

The Perceptron branch is picture on the right side of fig. (1.5), notice that it is considerably smaller in side and complexity than it's neighbor. Rather than accept an image-like input, the perception simply takes a vector-like input of properties of a larger data set - in this case, each waveform. We call these properties *features* [1, 5, 16]. Details on the creating of these features can be found in the *features* section of this document.

The input layer of the perceptron accepts a 2-Dimensional array. The axes, in order of indexing, represent (i) the size of the *mini-batch* of samples, (ii) the number of features for each sample. We use the same model hyper-paramater of 64 samples per batch, and have developed 23 unique clssification features that have been derived from time-space and frequency-space representations of the audio file data. We can denote the 2D shape if the input object into this branch as:

$$X_2 \in \mathbb{R}^{(64 \times 23)} \tag{12}$$

Again, any other shape will be rejected by the model, and an error is raised.

1.5.3 The Final Output Branch

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] Geron, Aurelien. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd ed., O'Reilly, 2019.

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

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

[5] 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.

[6] Levine, Daniel S. *Introduction to Neural and Cognitive Modeling*. 3rd ed., Routledge, 2019.

[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] Loy, James , *Neural Network Projects with Python*. Packt Publishing, 2019

[9] 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.

[10] 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.

[11] Mitchell, Tom Michael. Machine Learning. 1st ed., McGraw-Hill, 1997.

[12] Olson, Harry E. *Music, Physics and Engineering*. 2nd ed., Dover Publications, 1967.

[13] Peatross, Justin, and Michael Ware. *Physics of Light and Optics*. Brigham Young University, Department of Physics, 2015.

[14] Petrik, Marek. "Introduction to Deep Learning." Machine Learning. 20 April. 2020, Durham, New Hampshire.

[15] Short, K. and Garcia R.A. 2006. "Signal Analysis Using the Complex Spectral Phase Evolution (CSPE) Method." AES: *Audio Engineering Society Convention Paper*.

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

[17] White, Harvey Elliott, and Donald H. White. *Physics and Music: the Science of Musical Sound*. Dover Publications, Inc., 2019.

[18] 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.