CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

Feature Extraction and Machine Learning Techniques for Musical Genre Determination

A graduate project submitted in partial fulfillment of the requirements

For the degree of Master of Science in Electrical Engineering

By

Rosalind M. Davis

December 2017

The graduate project of Rosalind M. Davis is approved:

Dr. Deborah van Alphen	Date
Mr. James Flynn	Date
Dr. Xiyi Hang, Chair	Date

California State University, Northridge

ACKNOWLEDGEMENTS

I would like to thank the members of my committee, Dr. Xiyi Hang, Dr. Deborah van Alphen, and Mr. James Flynn, as well as Dr. Sharlene Katz, for their mentorship on this project as well as many others during my CSUN career.

I would also like to thank Emily Johnson for her unflagging patience and support, as well as for valiant grammar editing outside her field; and my parents, Ellen and Arnold Davis, for their invaluable support, wisdom, and encouragement throughout my education.

DEDICATION

This project is dedicated to the memory of my grandfather, Henry J. Goodwin, in gratitude for his early mentorship in math and engineering.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

Feature Extraction and Machine Learning Techniques
for Musical Genre Determination

By

Rosalind M. Davis

Master of Science in Electrical Engineering

Since 2015, the music industry has experienced a resurgence driven by online music sales and streaming, which has in turn been facilitated by very large archives of musical data. These large musical archives, however, remain challenging to search and index effectively, due to the scale of the data involved and the subjective, perceptual nature of how humans relate to music. Contemporary research in music information retrieval seeks to bridge this gap by using algorithmic analysis on features extracted from the underlying audio to automatically classify and identify perceptual features in music. This project applied three machine learning techniques (support vector classification, traditional neural networks, and convolutional neural networks) to two sets of audio features (Mel-frequency cepstral coefficients and the discrete wavelet transform) for the purposes of genre classification. Because convolutional neural networks have been used on images to great effect, the discrete wavelet transform data was used to map audio into the image domain, to leverage publicly available, pre-trained weight sets for four large, sophisticated image recognition networks. For all tasks, two subsets of a large, publicly available musical dataset were used, along with multiple training and optimization techniques. While all models were able to meet or exceed some

pre-existing benchmarks for the genre classification task, support vector classification was found to yield better results, with a best overall test set accuracy of 61%, than either traditional neural networks (51.4%) or convolutional neural networks (40.5%) on an eight-genre multi-class classification task. The application of the pre-trained image recognition networks to audio wavelet data decreased training time, but was not found to yield accuracies comparable to the accuracies those networks achieved on image data. The small size of the dataset relative to datasets in other domains, the reuse of data augmentation techniques intended for use on images, and sub-optimal feature extraction techniques are suggested as factors in the inability of the machine-learning models evaluated in this project to achieve the quality of results observed in the image domain. Audio-native augmentation techniques and the use of ensemble models present worthwhile avenues for future investigation.

1. INTRODUCTION

After falling sales for nearly two decades, the recorded music industry has experienced a resurgence since 2015, driven by the growth of streaming and online music sales [1]. This growth has been facilitated by the development of very large musical archives (e.g. Spotify, iTunes), which typically offer tracks numbering in the millions or tens of millions for paid streaming or download [2, 3]. However, the size of these archives raises other challenges due to the technological complexity of searching, indexing, and providing a human-usable interface into such large volumes of audio data [4, 5]. The surge in streaming and online music sales therefore necessitates new developments in the field of music information retrieval (MIR), the various modeling techniques and algorithmic approaches that allow large musical archives to be efficiently represented, indexed, and searched [6, 7].

The information that falls under the purview of MIR is diverse in both nature and significance. MIR may include not only objective, factual information about a track, such as the song's title and performing artist, but also subjective information such as genre and mood [8, 9], or the similarity between two songs [10]. It also may include a wide variety of information that may change across the duration of a song, such as tempo, key, and instrumentation [7, 11]. Furthermore, what precise features of music determine its various perceptual characteristics and the extent to which these perceptual characteristics are culturally universal both remain active fields of research [7, 12, 13, 14].

A common approach for MIR is to represent the underlying audio data via textual metadata, such as a song's title, album title, and artist, often in conjunction with additional short text annotations (tags) that describe the more subjective aspects of a piece of music, such as genre, mood, style, and instrumentation [15, 16]. However, [5] outlines several significant problems with this approach, especially regarding very large archives. First, when textual metadata is applied by humans, it is frequently difficult

to maintain consistency across a large archive due to the subjectivity of the individuals determining and applying the metadata [5]. Furthermore, the time cost of generating such metadata is often impracticably large: at the estimated 20–30 minutes it takes an expert to enter the metadata for a single song [5], a 30 million song archive like Spotify [2] would correspond to 10–15 million person-hours to tag by hand. By opening up such metadata entry to their user base, social media-driven music streaming and discovery services like Last.fm [15] are able to develop textual metadata with a smaller outlay of expert time. However, this democratic approach only increases the problem of variations in subjective perception amongst the many users applying the metadata [5]. Compounding this problem, because tags can be conceptualized and utilized in many different ways, individual users are not necessarily motivated to apply tags that align well with the information sought by other individual users [9].

Researchers are therefore motivated to identify new approaches for MIR that allow large volumes of music to be analyzed and processed without the time-consuming and expensive system of having humans provide often subjective or inaccurate information. Specifically, much current MIR research focuses on approaches for performing perceptual tasks, such as genre classification, algorithmically from audio features of various types [7, 8, 17]. These studies are generally focused either on specific approaches to audio feature extraction that seek to accurately encode various perceptual qualities of music [6, 7], or on the machine learning techniques by which such features can be accurately classified [6, 7, 18].

However, comparing these results across studies is not always meaningful. In recent years, very large scale *image* datasets (e.g. ImageNet, which contains 3.2 million images in 5247 classes at varying levels of granularity [19]) have allowed for dramatic improvements on image-driven tasks. However, *music* datasets of even remotely comparable size are limited, frequently by licensing concerns [20]. While some large datasets of music features have existed for several years (e.g. the Million Song Dataset

[16]), they generally do not provide the original audio, instead circumventing licensing issues by providing only the extracted features [20]. This can make it difficult to reproduce or compare study results where new features are derived from audio, one of the considerations which has driven the recent development of the Free Music Archive (FMA) dataset [20]. At just over 100,000 songs, the FMA dataset may still be smaller than image datasets like ImageNet, but it achieves a larger size than any other publicly available music dataset for which the audio is included [20]. Because of its novelty, however, benchmarking information for many machine learning tasks on this dataset remains unavailable.

This project, therefore, presents an overview investigation into three general machine learning strategies: support vector classification (SVC) models, traditional shallow neural networks, and deep learning by way of convolutional neural networks (CNNs). In all three cases, the FMA dataset is used, and both the pre-extracted features and the included audio files are applied to a single perceptual task: namely, genre classification. Two feature sets used in existing music audio research, Mel-frequency cepstral coefficients (MFCCs) and the discrete wavelet transform (DWT), were used as the inputs to these machine learning algorithms under varying training and optimization conditions, with the goal of identifying the genre of a track of music from the MFCC or wavelet features. The accuracies of the strategies thus tested were then compared. While it is not an exhaustive survey, the purpose of this project is to establish comparable benchmarks, derived under similar conditions, for several extant machine learning approaches on this recent dataset.

This paper is divided into six sections. This section provides an overview of the problem and the potential solutions that were investigated in this project. Section 2, “Background,” presents the conceptual background for, and an overview of relevant research into, the general techniques investigated in this project. Section 3, “Project Design,” gives an overview of the specific design frameworks used for each of the five

technologies investigated in this paper: MFCCs and the DWT for music feature extraction; and SVC models, traditional neural networks, and CNNs used to classify those features. Section 4, "Experiments," presents the procedure and results for the five experiments conducted to investigate these technologies. In Section 5, "Discussion," these results and their significance are analyzed, and the costs and benefits of the various technologies investigated are presented and discussed. Section 6, "Conclusion," provides a summary of results and analysis, and offers potential directions for future work.

2. BACKGROUND

While a complete overview of contemporary machine learning techniques and feature extraction methods is beyond the scope of this project, this section provides an overview of the technologies utilized in this project, along with a selection of findings from relevant research.

2.1. Feature Extraction for Music.

Originally designed for speech analysis [6], Mel-frequency cepstral coefficients are a short-time spectral feature set derived from the discrete Fourier transform [21]. They are formed by taking the frequency-scaled log of the amplitude spectrum of the discrete Fourier transform (DFT) of audio frames, then decorrelating these features by means of the discrete cosine transform, as shown in Figure 1 [21]. Through the use of the Mel scaling function, MFCCs are able to model an important nonlinearity in human auditory perception: namely that, within the approximate 20 Hz to 20 kHz band of normal human hearing [22], listeners perceive lower frequencies as more significant than higher ones [21].

Because human hearing does not perceive pitch linearly, the Mel frequency scale is often used as a preferred feature set for audio because it allows lower frequencies, which have more impact on aural perception, to be weighted more heavily than the less important, higher frequencies [21]. Because they reflect this psychoacoustic principle, MFCCs have been used extensively in audio modeling and analysis, both for speech [23, 24, 25, 26, 27] and for music [6, 10, 18, 28, 29, 30].

Specifically with regards to genre classification, MFCCs have been widely used, with significant success. MFCCs can be shown to distinguish pairs of genres (with varying success, depending on the pair of genres under question) by projecting MFCC data into a lower-dimensional space via principle component analysis (Figure 2, modeled on an example in [20]). The genre discriminatory quality of MFCCs has been

shown to extend machine learning on MFCC features. For example, Li *et al.* trained convolutional neural networks on MFCC features, achieving approximately 40% training accuracy on a six-genre classification task and approximately 90% training accuracy on a three-genre classification task [18]. However, MFCC data remains limited as to the types of perceptual audio information it can encode, particularly vis-à-vis the relationships between one temporal area in a song and another [7]. As a result, some MIR research combines MFCC features or principles with other approaches to achieve improved results [7, 18].

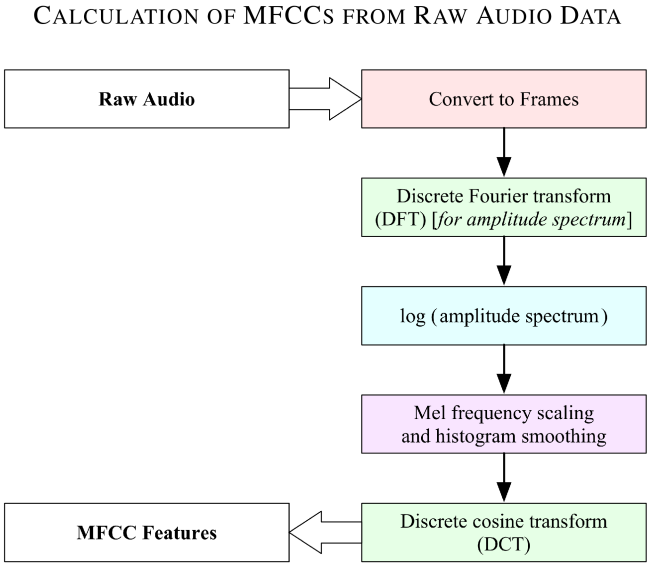


Figure 1: Process for calculating Mel-frequency cepstral coefficients (MFCCs) from raw audio data[21, Figures 1 and 2].

While MFCC data has been used widely for audio MIR, in recent years the wavelet transform has been used in many domains where multi-resolution analysis of non-stationary signals is desirable, including for audio processing, classification, and analysis [17].

While a detailed explanation of the mathematical background of the wavelet transform is beyond the scope of this project, a summary can be provided by means of the filtering properties of the transform. The discrete wavelet transform is essentially

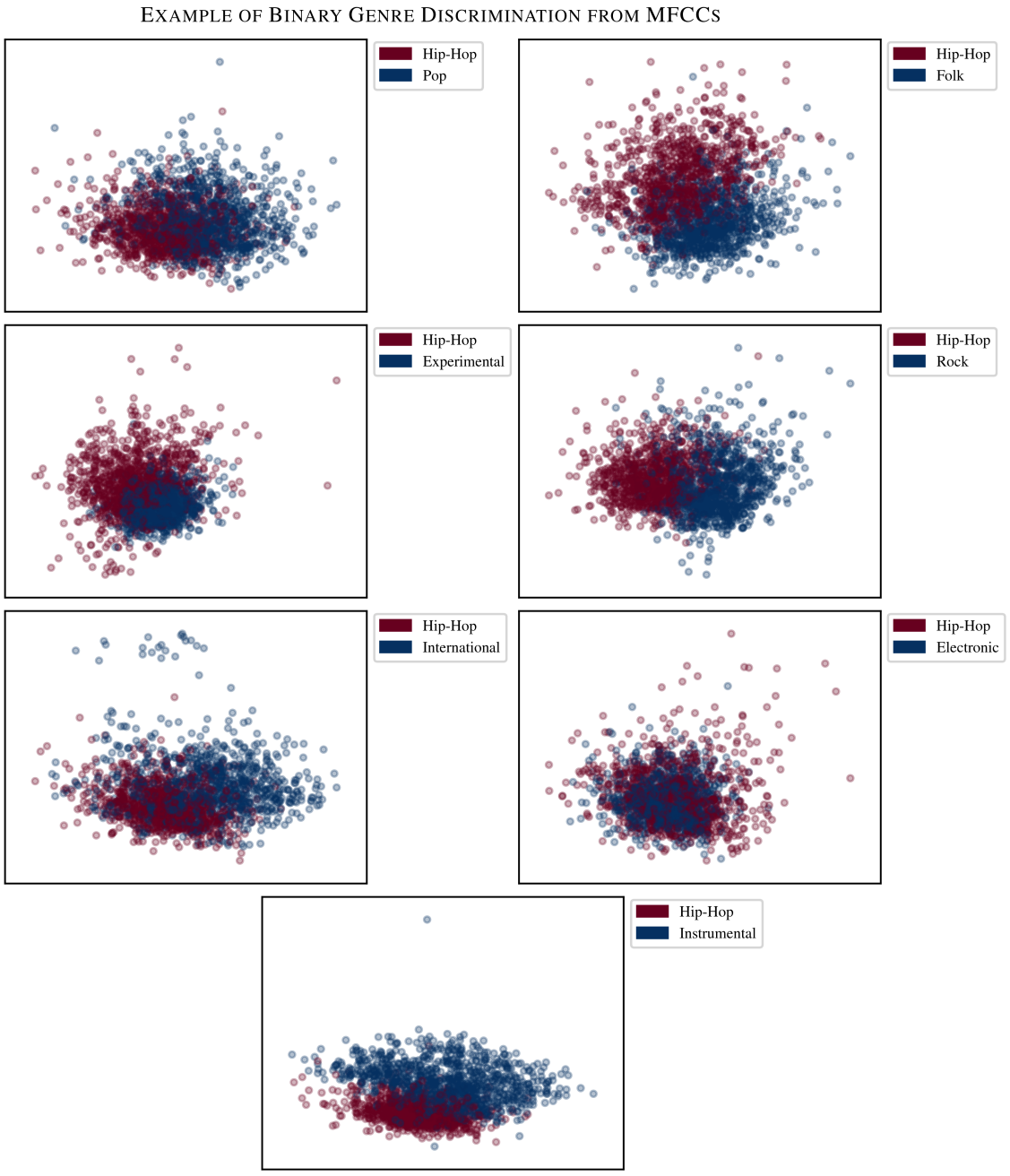


Figure 2: A 2D visualization of the genre-discriminant nature of Mel-frequency cepstral coefficients (MFCCs). In each graph, MFCC data for a single fixed genre, hip-hop, and one of the seven other genres in the FMA small dataset is projected into two-dimensional component space and then graphed, with the genre indicated by color. The extent to which MFCCs distinguish between a given pair of genres can be seen by the extent to which the two colors of data points cluster apart from each other. Note that MFCCs are not uniformly discriminant for all genres: for example, there is more overlap between the hip-hop and electronic genres than there is between hip-hop and folk.

calculated by convolution between the input signal and scaled, shifted copies of a signal with specialized mathematical properties [31, 32]. This specialized signal can be described either by a **wavelet function** (or simply **wavelet**), or by its associated **scaling function**: since each of the wavelet function or scaling function can be used to derive the other, either alone is sufficient to define the transform [32]. Many groupings, or **families**, of wavelets exist, with each family derived by means of a different mathematical procedure [31, 32]. However, all wavelet families share the property that convolving a wavelet with an input signal performs a paired high-pass/low-pass filtering operation on that input signal [33]. The wavelet and scaling functions, respectively, separate out the input signal's high-frequency information (**difference coefficients**) and low-frequency information (**scaling coefficents**), which can then be analyzed or processed independently [32, Ch. 5, 8].

This filtering operation can be performed multiple times, at multiple **scales**, or filter cutoff frequencies, by time-compressing or expanding the wavelet that defines the filters [31]. By successively applying the filter, downsampling, then taking the scaling coefficients and applying the filter again, difference coefficients at multiple levels are derived [31, Ch. 3]. This process of repeatedly applying the wavelet transform to the output of the wavelet transform, known as Mallat's algorithm, is sometimes referred to as **wavelet decomposition** [31], and it performs the basis of the wavelet feature extraction used in this project.

Use of the wavelet transform for audio remains an active area of research, but several studies have found it effective for both speech and music. Tzanetakis *et al.* proposed several applications of the wavelet transform for music feature extraction, such as identifying tempo or beats per minute, and semantic classification, and found the performance of the discrete wavelet transform (DWT) to be comparable to the performance of fast Fourier transform features and MFCCs [17]. Similarly, Turner *et al.* used the DWT with a Mel frequency scale to compute the MFCCs, and found that the

DWT led to improved efficiency in a speaker recognition task when compared to the more traditional use of the DFT in the MFCC calculation [27]. Li *et al.* proposed a histogram-based combination of wavelet features as a new feature extraction technique for music, finding that this feature set lead to improvements in genre recognition over the use of MFCCs alone [11]. Li and Ogihara also achieved improved classification results by using a combination of MFCC and wavelet data with support vector classification [8]. Shen *et al.* similarly achieved improved classification results using a novel feature extraction technique designed to use a histogram of wavelet coefficients, similar to the procedure in [11], to derive semantic labels ("audio words") for songs on a section-by-section basis [7].

2.2. Support Vector Classification.

Support vector classification is a machine learning technique which uses some kind of mapping to transform input data into a high-dimensional space, and then determines an optimal hyperplane in that high-dimensional space to be used as a decision boundary distinguishing items in one class from items in another [35, 36]. SVC models are a subclass of support vector machines (SVMs), which can perform tasks other than classification (e.g. regression); for clarity, the phrase "SVC models" is used in preference to the more general "SVMs" throughout this paper.

As is suggested by the definition, SVC models lend themselves to binary classification problems, i.e., problems in which there are two possible classes (e.g. "cat" versus "dog"), and all samples belong to exactly one of the classes [35]. These models are also sometimes termed "one-against-one" classification models [36]. However, SVC models can be extended to solve more complex, one-against-many classification problems (e.g. "cat" versus "dog" versus "octopus"), either by the use of more complex models capable of considering three or more classes simultaneously, or by aggregating the probability outputs of multiple models comparing each possible pair of classes [37].

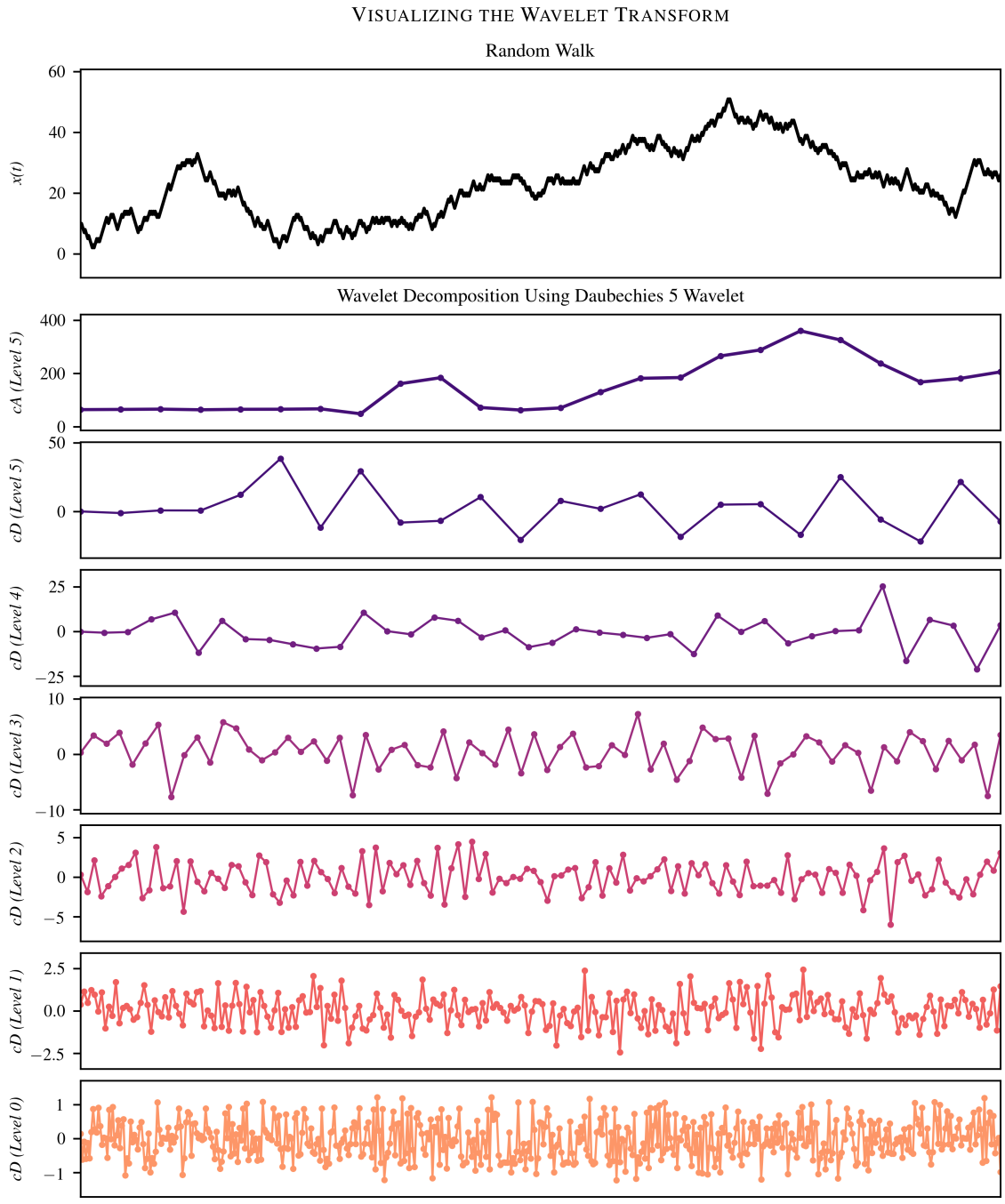


Figure 3: A visualization of the filtering view of the wavelet transform, using a random walk. The random walk undergoes 6 levels of wavelet decomposition using the Daubechies 5 wavelet [34]. At each stage of the decomposition, the scaling coefficients *cA*, containing low frequency information, from the previous stage are filtered again, while the difference coefficients *cD*, containing the high frequency information, are left unchanged [31, 32]. Every successive layer of decomposition includes a downsampling operation that halves the number of data points [31]. Note that the scaling coefficients from the final stage resemble a smoothed version of the input signal, but occur at a delay, as is to be expected of digital filtering operations.

The way the optimal decision boundary in an SVC is determined is via the SVC's **kernel function**, or simply **kernel**, which controls the non-linear mapping of input examples to the high-dimensional space in which decisions are made [38, p. 26]. In the SVC's simplest form, the kernel function is simply an inner product of any pair of training examples. However, more complex functions that use a nonlinear mapping to transform training examples before calculating the optimal decision boundary may achieve higher accuracy [36, 38].

Support vector classification enjoys widespread use in part because it is effective in situations where the number of samples in the input dataset is small relative to the dimensionality of the individual samples [36]. Because the decision function is determined by a subset of the training samples, SVC approaches are also efficient in terms of memory usage [36]. However, because of the mechanism by which SVC makes binary classification decisions (i.e., this training image is of class "octopus" rather than class "dog"), it is computationally expensive to derive a class probability output (i.e., this training image has a 53% chance of being of class "octopus," a 26% chance of being of class "cat", and a 21% chance of being of class "dog") from SVC models [36]. SVC models are therefore not appropriate for all classification tasks.

2.3. Neural Networks.

As the name suggests, neural networks were originally derived from neurological models, though contemporary neural network techniques have diverged somewhat from strictly biological models [39]. At the most basic level, neural networks are made up of some number of artificial neurons. Each artificial neuron accepts some number of inputs, applies a function to those inputs, and produces some number of outputs (Figure 4) [39]. Typically, even if all neurons implement the same function on the inputs, that function relies upon some values associated with the individual neuron (**parameters**) [39]. Neurons are generally arranged in **layers**, in which a fixed number of neurons

operate on the same set of aggregate input elements (or subsets of the same set of aggregate input elements) and produce a set of aggregate output elements [39]. During the learning process, inputs are applied, neuron parameters operate on these inputs, and an output is derived; and then an evaluation of the quality of that output is made [39]. Then, the neuron parameters are updated with the goal of improving the quality of the output on the next iteration [39].

Within this general framework, a wide variety of options control the structure and behavior of a neural network, such as how many neurons are in a layer, how many of the input elements a given neuron sees, how many of the output elements a given neuron affects, the function implemented by a given neuron, et cetera [39]. Typically, the properties of neural networks are discussed on a layer-by-layer basis [39], and while it would not be feasible to list all possible variations here, a few layer types are of particular import.

The first is the **affine layer**, the most basic layer used in neural networking, in which each neuron is connected to every neuron in the previous layer [40]. In other words, if it is desirable to process an input of 4 elements and produce an output of 3 elements, we can use an affine layer with 3 neurons, each with 4 inputs. The outputs of the 3 neurons are the 3 output elements desired.

Typically, the individual neurons in an affine layer implement $y_k = \text{act}(w_k x + b_k)$ for the kth neuron, where $w_k x + b_k$ represents a linear transformation of the input and $\text{act}(x)$ represents the **activation function** [39]. In the linear part of the formula, w_k is the vector of **weights** that should be applied to each input, and b_k is known as the **bias**: both weights and biases are parameters of the neuron that can be updated during training [39]. Multiple possible activation functions exist, but they are typically non-linear functions such as $\tanh(x)$ or $\max(x, \xi)$, where ξ is a threshold value and potentially a trainable parameter of the neuron [39]. The activation function used throughout this

project was the rectified linear unit (ReLU) activation function, $\mathrm{ReLU}(x) = \max(x, 0)$; in this case, the threshold is not a trainable parameter, but instead a constant 0 [41].

Note that as long as all neurons in a layer implement the same general function, albeit with different weights and biases, the matrix form of the function for a single neuron, $y_{1\times1} = \mathrm{act}(w_{1\times n}x_{n\times1} + b_{1\times1})$, can be easily extended to all k neurons in a single operation, $y_{k\times1} = \mathrm{act}(W_{k\times n}x_{n\times1} + b_{k\times1})$ [39]. This property makes neural networks easy to represent and evaluate efficiently with modern vectorized computation tools like NumPy [42]. This ease of matrix representation and calculation makes it possible to evaluate multiple input examples, moving through multiple successive neural network layers to the output, as a sequence of efficient matrix operations.

The power of neural networks lies in this ability to stack multiple layers one after another. Possibly the simplest form of this stacking is two affine layers atop one another, i.e. a neural network that takes m input elements and processes them through a layer of k neurons, then processes them again through a second layer of n neurons to produce n outputs (Figure 4) [39]. This conceptually simple model of a two-layer network can be shown mathematically to be capable of implementing any function [39, 41, 43]. A variation on this simple two-layer network was used in Experiments 2–5 of this project.

The challenge with neural networks, despite their representational power, comes in training them effectively. For ease of discussion, it should here be noted that for classification tasks, the final layer of the neural network is a **classifier**, which takes the arbitrary number of numeric outputs from the previous layer and produces an activation pattern that identifies the class to which an input example belongs [44]. As previously mentioned, during the training process, the quality of the output is evaluated in some way, and then that quality evaluation is used to update the weights. This quality evaluation is done by way of a **loss function** that evaluates the discrepancy between the actual output and the desired output: the goal of training is to minimize this

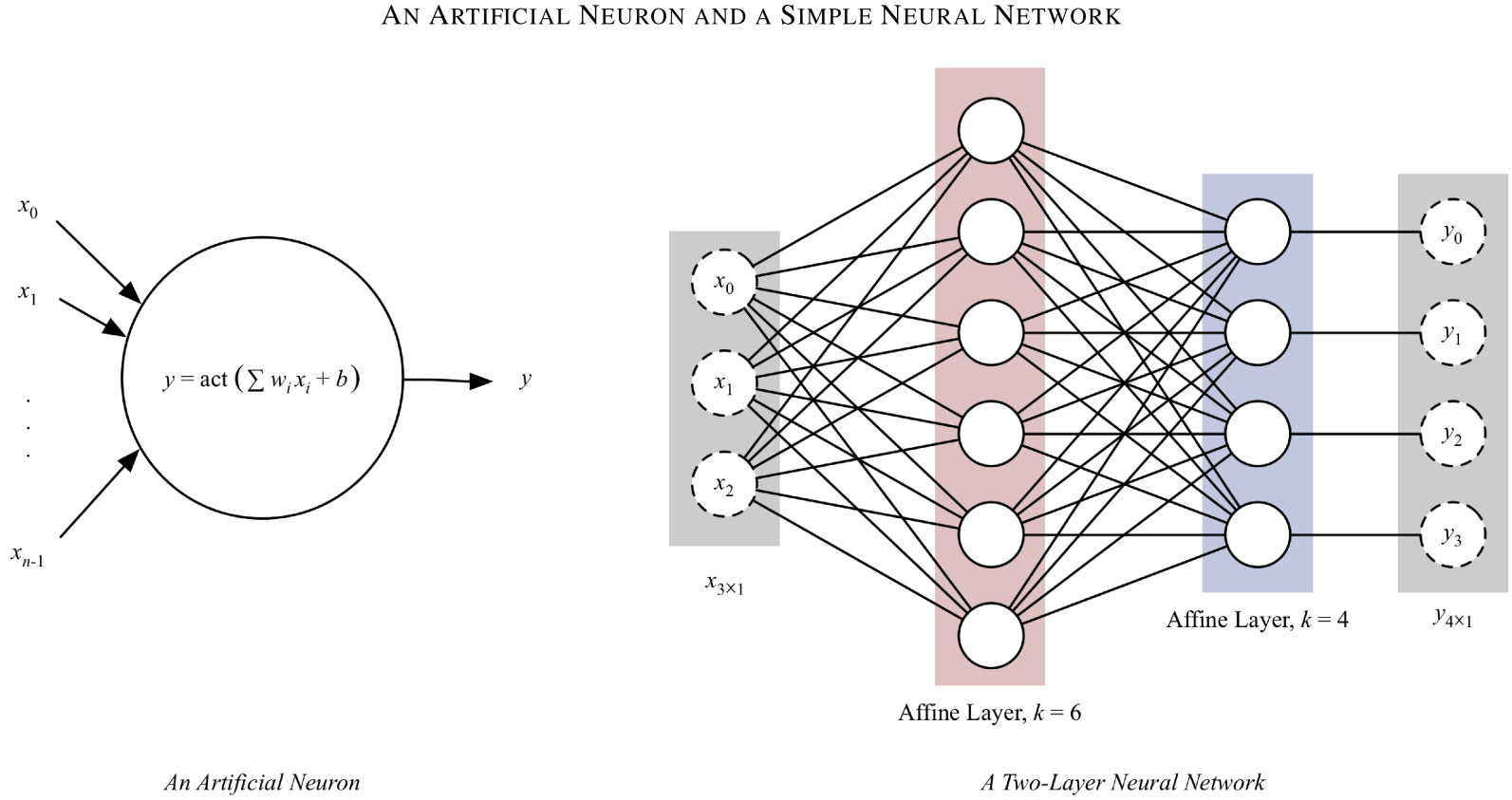


Figure 4: A basic artificial neuron (left) and a simple two-layer neural network (right) [39, 41]. The artificial neuron accepts a vector input of length n, multiplies it by its associated n weights, and adds a bias, then applies a non-linear activation function, producing a single output [39, 41]. In an affine layer, k such neurons operate on the same input, producing a k-element vector output [39]. Multiple affine layers may be stacked, with each accepting the output of the previous layer as its input [41]. The final affine layer's size must match the size of the desired output, but the size of any intermediate (**hidden**) layers is arbitrary and can be adjusted to suit the application [39, 41].

loss [44]. For multi-class classification, the most commonly used types of loss function are the *hinge loss*, which evaluates whether the score for a particular class is higher than all the other class scores by some threshold value, and the *cross-entropy loss* as determined by the **Softmax function**, where the loss for the ith training example is given by $L_i = -\log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}})$ for all f_j in the vector of class scores [39, 44]. The Softmax function allows the network to evaluate the cross-entropy between the class probabilities predicted by the neural network and the correct probabilities, in which the correct class has probability 1 and all other classes have probability 0 [44]. The Softmax cross-entropy loss was used for all neural networks in this project.

The updates to network parameters, then, are made through a process known as **stochastic gradient descent**, or SGD [39]. In standard SGD, the gradient of the loss function is calculated, and weights are updated by a fixed amount in the direction of maximum gradient towards low loss [39]. The goal is for the network to **converge** upon a global minimum in the loss function [41]. The amount by which these updates are made is called the **learning rate**, and is what is known as a **hyperparameter**: i.e., a variable that is set by the person training the neural network and that controls some aspect of the training procedure [39, 45]. Frequently, determining the best value for the learning rate and other hyperparameters is a major challenge in developing neural networks, as the optimal values depend heavily upon both the architecture of the network and the nature of the machine learning task under consideration [45, 46, 47].

There are several common problems that can be encountered during the training process, such as networks that oscillate or otherwise fail to converge on a loss minimum, networks that converge on a local minimum rather than a global minimum, and networks that in fact converge so well that they fit the noise in a training data set, rather than the true underlying data [41, 45]. This last problem, **overfitting**, is particularly a problem with large networks with many parameters operating on small datasets with few examples [48], and was an issue at several points in this project.

These problems, however, can be countered or reduced via several different techniques. **Dropout** is a method of controlling overfitting by ignoring a random subset of neurons in any given training step [41, 48]. This essentially turns a single large neural network into a much larger ensemble of small neural networks which share parameters, and in which each network has been trained for fewer training steps [41, 48]. The smaller number of training steps in any given subnetwork decreases the overfitting, while the overall ensemble model can still achieve high accuracy due to extensive training [41]. Dropout is a specific form of **regularization**, which, in the general sense, encompasses mechanisms to control the magnitude of the weights that are used inside neurons, since large weights tend to cause certain inputs to have a disproportionate affect on the output [49]. **L2 regularization**, for example, adds a term based on the square magnitude of the weights to the loss function, encouraging the neural network to use all its inputs evenly rather than relying heavily on only a few [49]. Both dropout and L2 regularization were used in multiple places in the models used in this project.

Another technique for improving training performance used in this project is **batch normalization**, which performs a statistical transform, similar to whitening, on each batch of training data [50]. This reduces the impact of the weights (typically random) that are set at model initialization and helps the network to converge faster [49, 50]. Batch normalization was also utilized in multiple places in the models used in this project.

To address other common training difficulties, many variations on the standard SGD procedure seek to optimize the learning process, rather than the underlying model [45]. Three were used in this project: the first, **learning rate decay**, gradually decreases the learning rate as training progresses, to allow fast learning early on while allowing the network to navigate small local changes in loss around a minimum [45, 51]. The second technique, known as **momentum**, uses the analogy of a ball rolling across the surface of the gradient, and dampens oscillations in the gradient de-

scent process by introducing a factor similar to the coefficient of friction, aiding SGD convergence [45, 51]. Both of these strategies are used as refinements on top of the general SGD procedure. The final training optimization strategy used in this project, **RMSprop**, is an adaptive update procedure that can be used as an alternative to SGD, which uses batch-based moving averages to normalize the gradient during the update process [45, 51, 52]. This has been shown to achieve a balance of rapid learning and good model convergence [45, 51].

2.4. Deep Architectures.

While these simple neural network structures have great representational power, they have been excelled in recent years, particularly in the field of computer vision and image recognition, by deep networks [53]. It has been found that the success of two-layer neural networks lies less in their theoretical universality than it does in the fact that they are particularly well-suited for modeling the statistical properties of the data humans would like to use them on [41]. It has also been observed experimentally that deeper networks yield better results, even though two-layer networks have this property of universality [41]. However, traditional neural networks with more than two affine layers tend to be difficult to train [41]. Deep learning has, therefore, relied on the development of novel layer configurations that can be combined many layers deep to build powerful models with higher empirical accuracy than simple affine models [54].

In a **convolutional neural network**, the deep architecture investigated in this project, layers are made up of blocks that implement convolution-based transformations on input data that is arranged in a grid, such as an image file [54, 55]. From a signal processing standpoint, these layers can be viewed as implementing a filterbank, with some trainable transfer function, that performs a set of spectral manipulations on the input data just as any more traditional filterbank would do [54]. In convolutional neural networks, these **convolutional layers** are often used in conjunction with **pooling layers**,

which take a local average of some kind and optionally reduce the dimensionality of the input via a downsampling operation [54, 55]. Pooling layers introduce a degree of **location invariance**, i.e., they reduce the network's sensitivity to the input grid being shifted slightly [55]. The usefulness of this invariance can be illustrated with a simple example from the image recognition domain: if a network is being trained to identify a cat, it is desirable for it to identify both a cat in the bottom right of the image and a cat in the middle of the image. Pooling also tends to reduce overfitting [54].

Much as wavelet decomposition successively separates high- and low-frequency data, convolutional neural networks tend to learn low-level, general features in early levels and high-level, detail features in later levels [56]. While the similarity between the filter view of convolutional neural networks (Figure 5) and the filter view of the wavelet transform (Figure 3) was not directly investigated in this project, it is here noted as a point of interest, as well as a potential direction for future research.

Convolutional neural networks have achieved many state-of-the-art results in image recognition, across several different data sets [53]. In part this is because, unlike traditional affine networks that accept only vector data for their inputs, convolutional neural networks are designed to accept 3D inputs (i.e., two-dimensional images with multiple color channels), and handle blocks of input data in the context of the positional relationships of elements in that block [39, 54, 55]. In other words, typically a two-dimensional image is passed through a flattening operation on its way into an affine layer, which removes valuable information about what pixels are above and below a given pixel in the image, but convolutional neural networks are designed to accept and exploit this grid-based data [54, 55]. While individual weights on the inputs to neurons in an affine layer have no intrinsic relationship to one another, in a convolutional layer, an individual convolutional filter operates on contiguous blocks of the input [54]. This architectural reality of convolutional neural networks matches the un-

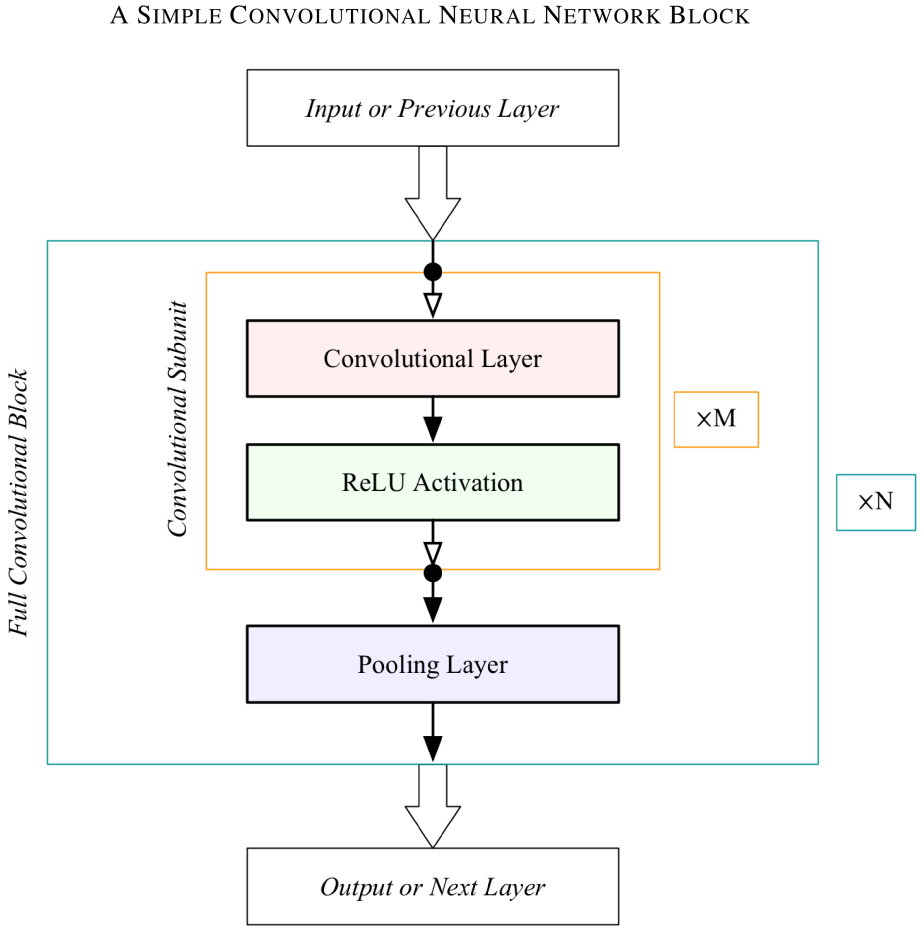


Figure 5: A simple convolutional neural network block [54]. The pooling layer is optional. In some implementations, including Keras [57], the ReLU activation may be combined with the convolutional layer and treated as a single discrete layer. In the simplest case, only one convolutional subunit is used, and the full convolutional block is not repeated [54]. In more complex architectures, the convolutional subunit may be repeated M times within a given convolutional block, and convolutional blocks may be stacked N deep [54].

derlying reality of the image data that they are intended to process, and makes CNNs particularly suited for image-based tasks [54, 55].

However, convolutional networks typically have a large number of parameters, and convolution operations are computationally expensive, making the training of convolutional neural networks time and resource intensive [56]. This high resource cost associated with convolutional neural networks has been mitigated somewhat by increasing access to graphics processing units (GPUs) capable of massively parallel computation at high speed [39, 54]. In this project, Google Cloud Engine [58] was utilized with an NVIDIA GPU to accelerate neural network training tasks.

A primary line of investigation in this project was the extent to which convolutional neural networks trained on images can learn to "see" not just native image data, like a picture of a dog, but image representations of data in a different specific domain: in this case, an image of the spectral content of music audio. This idea is derived from the principle known as **transfer learning** [56], in which convolutional neural networks trained on a particular image dataset are used as general visual feature extractors for a different image based task. Transfer learning was used by Razavian *et al.* to obtain state-of-the art results for several computer vision tasks [59]. However, the application of image-trained networks to data outside the native image domain has been limited.

3. PROJECT DESIGN

This project consisted of a total of five separate experiments, utilizing two overlapping datasets and two different feature sets for each dataset. Both feature sets contain spectral information extracted from music audio. These feature sets were then used as training data for either SVC models or neural networks designed for classification tasks, and the results were compared. In all cases, the goal of either SVC or neural network analysis was to accurately classify each example in the dataset as belonging to one of eight musical genres.

3.1. Dataset.

The Free Music Archive (FMA) dataset [20] formed the base dataset for all experiments in this project. The FMA dataset provides both extracted features and MP3 audio for a total of 106,574 music tracks released under a Creative Commons license. It is split into four subsets: a *small* dataset containing 8,000 30-second or shorter audio tracks in 8 genres (1000 tracks per genre), a *medium* dataset containing 25,000 30-second or shorter tracks in 16 genres, a *large* dataset containing 106,574 30-second or shorter tracks in 161 genres, and the *full* dataset containing 106,574 full-length tracks in 161 genres.

This project used the small dataset, referred to throughout this paper as "FMA small," in Experiments 1, 2, 3, and 4. It also used a constructed dataset, referred to throughout this paper as "FMA extended," in Experiments 1 and 5. To construct the FMA extended dataset, all tracks in the large dataset provided by [20] with a genre that was one of the eight genres in the small dataset were used. Because the small dataset is a balanced subset of the large dataset [20], all examples in the FMA small dataset also appear in the FMA extended dataset.

For reproducible results, [20] also provides pre-selected training, validation, and test splits containing 80%, 10%, and 10%, respectively, of the dataset. These recom-

Dataset	Split	Genre							
		Electronic	**Experimental**	**Folk**	**Hip-hop**	**Instrumental**	**International**	**Pop**	**Rock**
small	train	800	800	800	800	800	800	800	800
	validation	100	100	100	100	100	100	100	100
	test	100	100	100	100	100	100	100	100
extended	train	7,662	8,557	2,275	2,910	1,579	1,124	1,815	11,394
	validation	871	966	229	319	191	137	313	1,324
	test	839	1,085	229	323	309	128	204	1,464

Table 1: Breakdown of the contents of the two datasets (FMA small and FMA extended) used in this project. Note that while the small dataset is balanced, with the same number of samples in each genre in each split (train, validation, or test), the extended dataset is not.

mended splits were used for both the FMA small and FMA extended datasets in this project. The small dataset split information was used exactly as provided by [20]. For the extended dataset, the split information from the large dataset provided by [20] was used for all tracks included in the extended data set (i.e., all those tracks with an appropriate genre). While this resulted in the train:validation:test ratio varying slightly from the 80:10:10 ratio fixed for the small dataset, this variation was not large and it was decided that reproducibility was more significant for this particular project.

The FMA small dataset is designed to be balanced by genre, i.e., it contains exactly the same number of samples for each genre in each split [20]. Specifically, it contains 1000 tracks per genre, and when using the training, validation, and testing splits provided, this results in a consistent 800 training samples, 100 validation samples, and 100 testing samples for each genre. This dataset is considered small for deep learning applications because, for all models investigated in this project, there are many times as many parameters per model (Table 2) as there are samples per class [60]. This situation tends to cause problems with overfitting [55, 61]. Multiple strategies for improving deep learning results with small data sets exist, such as regularization [62], k-fold cross-validation [55], and data augmentation [63], all operating in conjunction with careful hyperparameter and activation function selection [64]. The effects of data augmentation on the small dataset were investigated in Experiment 4 and the results of [62] and [64] guided decisions throughout; however, other approaches were beyond the scope of this project.

By contrast, the large dataset (and the FMA extended dataset constructed from it) contains many more samples, but is not balanced by genre (Table 1). For example, the extended dataset's training split contains over ten times as many "rock" samples as it does "international" samples. This varying number of samples across different classes is in general undesirable because it can lead to the large classes (e.g. "rock") overwhelming the small classes (e.g. "international") in the classifier [65]. However, in

general, an increased number of samples is anticipated to give improved results because it reduces the gap between the number of parameters and the number of samples, and thus reduces the probability of overfitting [55, pp. 110–113]. While multiple strategies at both the data and algorithmic levels exist for coping with unbalanced data sets [65], in the interests of limiting the number of factors being investigated concurrently, the only strategy that was used in this project was data augmentation, as it was also used on the small data set. The effects of the larger, but unbalanced, FMA extended dataset, both with and without data augmentation, were investigated in Experiments 1 and 5.

3.2. Features.

Two feature sets were used in this project. The first was the Mel-frequency cepstral coefficients (MFCCs) provided for each track as part of the FMA dataset [20].

Because of the previously-mentioned indications that the DWT may offer an alternative, or augmentation, to traditionally-calculated MFCC features for music and general audio analysis tasks ("Feature Extraction for Music," p. 5), the second feature set used in this project was a form of the DWT calculated from the compressed audio (MP3) files provided with the FMA dataset using the PyWavelets package [34]. Because the wavelet features were intended for use with convolutional neural networks that had been pre-trained for image recognition (Experiments 3, 4, and 5), the wavelet data extracted with PyWavelets was saved as image (PNG) files using Matplotlib [66]. These images contain time data on the horizontal axis and frequency data on the vertical axis, allowing, in theory, the time relationship of various spectral qualities to be identified, as was key to the results in [7].

All wavelet images generated from the audio files for this project contain time on the x axis and frequency on the y axis. While the DWT implementation used in this experiment did not use the Mel function to adjust the frequency scale, it did use an oc-

tave (doubling frequency) framework [11, 17] for the frequency axis to achieve a log-base frequency scale.

Because this project used models pre-trained on the multi-channel color images in the ImageNet dataset [19], color wavelet images were constructed by mapping the single-value wavelet data output by PyWavelets to a perceptually uniform, three-channel colorspace with the Matplotlib "magma" colormap [67]. These data can be considered to be informationally equivalent to the same data saved as a grayscale image [67], but the colorized version was selected because convolutional neural networks trained for image recognition are known to contain filters sensitive to color content [68], i.e., the pre-trained networks can be expected to be utilize color data as part of the feature extraction process.

A randomly-selected sample of generated wavelet PNG files from each of the eight genres in the FMA small dataset appears in Figure 6.

It is important to note that the MFCC feature set and the wavelet feature set are of different dimensionality, i.e., for any given example in the dataset, the wavelet feature set contains many more values than does the MFCC dataset. For any given example in the dataset, the MFCC features are in the form of a 140-element vector. In contrast, for any given example in the dataset, the wavelet features are in the form of a $256 \times 256 \times 3$ image, corresponding to 196,608 elements per example.

The pre-trained neural network models used in Experiments 2, 3, 4, and 5 all have minimum size requirements for input images (Table 2), which make the raw MFCC features provided by [20] inappropriate as inputs to these models.

Additionally, while support vector classification is in theory effective on high-dimension inputs [36], the wavelet feature set was found to be too large to practically hold in RAM, making the wavelet features inappropriate as inputs to the support vector classifiers used in Experiment 1.

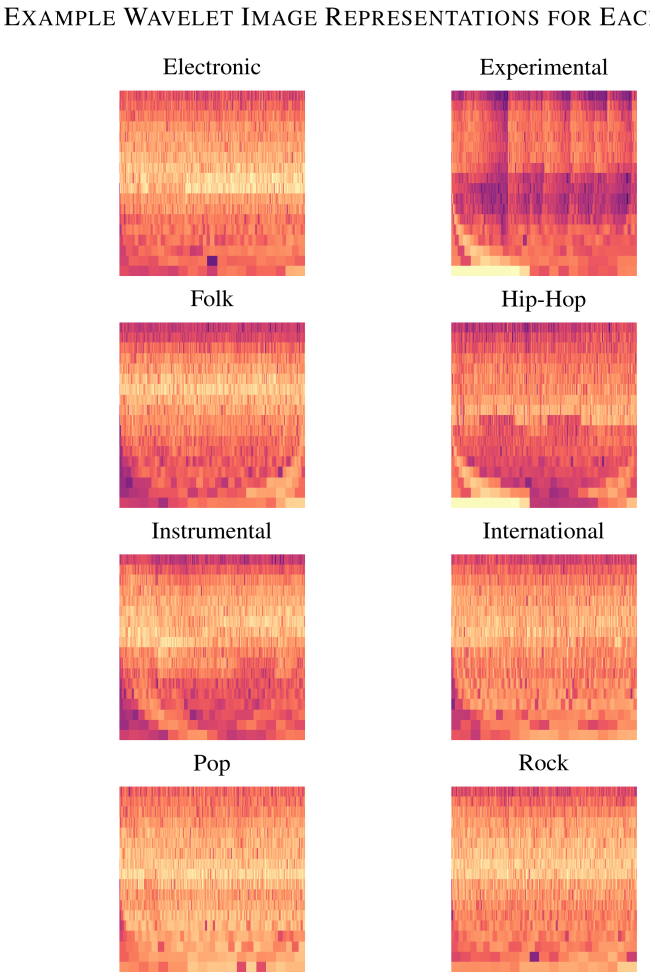


Figure 6: Randomly-selected examples of wavelet images, for each genre in the FMA small dataset. These images were generated using the Daubechies 5 wavelet and the PyWavelets module [34].

Therefore, only the models appropriate to the size of the feature set were used in this project: the lower-dimensionality MFCC features with the SVC models in Experiment 1, and the higher-dimensionality wavelet features with the neural networks in Experiments 2–5.

3.3. Preprocessing.

Before training the SVC models, the MFCC data was first preprocessed by subtracting the mean and scaling to unit variance, and the training examples were shuffled, using scikit-learn's `preprocessing` module [69, 70].

When training the neural networks on wavelet images, the data was first preprocessed by subtracting the mean across the training split for the appropriate dataset. Much like the wavelet features themselves, this mean can be treated as a $256 \times 256 \times 3$ color image. A visual representation of the means for the FMA small and FMA extended data sets appears in Figure 7.

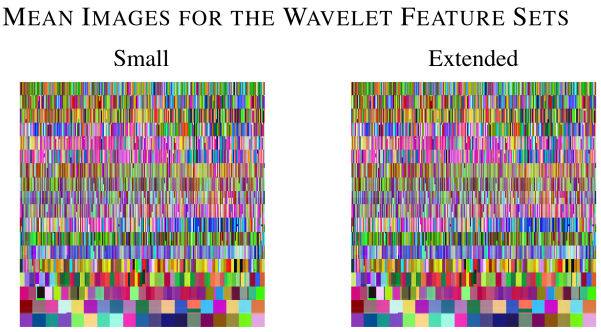


Figure 7: Calculated mean images for the wavelet feature sets, for the FMA small dataset (left) and the FMA extended dataset (right).

3.4. Support Vector Classifiers.

A total of four SVC models were trained on the MFCC features in this project. These models were instantiated and trained with scikit-learn [70], with varying kernel function type. The kernel determines the mapping of input data into the high-dimensional space (Hilbert space) in which classification decisions are made [35, 70, 71]. In essence, the kernel function $k(x_i, x_j)$ corresponds to taking the inner product of two examples x_i and x_j, mapped into the Hilbert space by means of a mapping function, $\phi(x)$, i.e. $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ [71]. In matrix form, this becomes $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ [36]. The mapping function and the kernel function then become interchangeable, and the simpler can be calculated as an analogue of the more complex [71].

3.4.1. Linear SVC.

The first and simplest SVC kernel used was the linear kernel type, which corresponds to using the mapping $k(x_i, x_j) = \langle x_i, x_j \rangle$ for two input examples x_i and x_j to determine the kernel [36].

3.4.2. Polynomial SVC.

An SVC model with a polynomial kernel was also used, corresponding to the mapping $k(x_i, x_j) = \gamma(\langle x_i, x_j \rangle + r)^d$ [36]. The default values of $\gamma =$ `'auto'` $(= \frac{1}{n_{features}})$, $r = 0$, and $d = 3$ were used [36].

3.4.3. RBF SVC.

The default scikit-learn kernel type is the *radial basis function*, or RBF kernel, which uses a Gaussian of the form $k(x_i, x_j) = \exp(-\gamma \left\| x_i - x_j \right\|^2)$ for the kernel [36]. The default value of $\gamma =$ `'auto'` $(= \frac{1}{n_{features}})$ was used [36].

3.4.4. Sigmoid SVC.

The final kernel type used was the sigmoid kernel. The sigmoid kernel corresponds to the mapping $k(x_i, x_j) = \tanh(\gamma \langle x_i, x_j \rangle + r)$ [36]. The default values of $\gamma =$ `'auto'` $(= \frac{1}{n_{features}})$ and $r = 0$ were used [36].

Scikit-learn SVC models use one-against-one classification inside each model, and so implement one-against-many classification by means of multiple one-against-one models, each distinguishing between one possible class pair [36]. For n classes, this produces a total of $\frac{n(n-1)}{2}$ internal one-against-one classifiers [36], or 23 classifiers for the 8 classes in the FMA small and FMA extended datasets. After training, the full SVC classifier can then use the aggregate output of these internal classifiers to predict a single most-likely class for a given input sample.

3.5. Neural Networks.

A total of five neural network models (one simple shallow network and four convolutional networks) were trained on the wavelet features in this project.

3.5.1. Two-Layer Neural Network.

The first model used was a simple, non-convolutional network [55, pp. 164–172] with two affine layers, with a batch normalization layer, an ReLU activation layer,

and a dropout layer in between, and a Softmax activation function used for classification at the output. This model is referred to throughout this paper as the **two-layer network**, due to the number of affine layers present. Its architecture was based on a model suggested by the course materials for CS231n at Stanford University [72, 73] and implemented with Keras [57]. The two-layer network was both tested independently and used as the classifier for the four pre-trained convolutional architectures (Inception V3, Xception, ResNet50, and VGG16) used in this project.

The other four networks tested were all convolutional models with weights that had been pre-trained on the ImageNet classification problem [19], with a final top–1 image recognition accuracy between 71% and 79% ([74], summary provided in Table 2). The Keras package allows these models to be quickly instantiated and their pre-trained ImageNet weights loaded at instantiation [74]. A classifier appropriate to the task at hand can then be added at the output, and the model can be further trained, either as a whole or in part, to fit the desired classification task [74]. In this case, the two-layer network was used as the classifier.

3.5.2. VGG16.

The simplest convolutional architecture investigated in this project was VGG16 [75]. The VGG architectures, which can be constructed in different sizes, stack units of multiple convolutional layers of varying sizes, interspersed with pooling operations [75]. At the most basic level, the convolutional layers are made up of a varying number of small (1×1 and 3×3) convolutional blocks which fully cover a very large area of the layer input [75]. These high-coverage convolutional layers lead to VGG architectures containing a very large number of parameters even for a small number of depthwise layers: the built-in implementation of VGG16 provided with Keras, for example, contains 138,357,544 parameters across 23 total layers, over five times as many parameters as (e.g.) Keras's built-in implementation of ResNet50, which contains 25,636,712 pa-

rameters across 168 total layers ([74], summarized also in Table 2). However, VGG architectures are known to achieve high accuracy on image classification tasks: an ensemble of VGG models of varying sizes (VGG16 and VGG19) achieved state-of-the-art results on the 2014 ImageNet challenge [75].

VGG-type architectures have been found to be adaptable to a wide variety of deep learning tasks [75, 76, 77, 78]. However, they have a very large number of parameters and thus require a large amount of both processing power and memory to be trained and used, and more complex models have been found to yield better results for image classification (Table 2). The other convolutional models used in this project attempt to address these performance concerns and high resource costs associated with VGG-type models.

3.5.3. Inception-Based Models.

Two of the pre-trained models used, Xception and Inception V3, derive from the Inception architecture proposed in [79], which sought to model a sparse network structure, previously shown to be optimal for computer vision tasks, while leveraging the dense, highly connected underlying architecture of efficient modern computational libraries. To accomplish this, Szegedy *et al.* proposed an "Inception module" that performed both 1×1 convolutions on small-scale, highly clustered information and 3×3 and 5×5 convolutions on larger-scale, less densely clustered information at each level, alongside a 3×3 max pooling operation [79, Figure 2]. These Inception modules also preceded the larger convolutions (3×3 and 5×5) with 1×1 convolutions for dimensionality reduction, to keep the number of parameters and overall computational complexity of the model within bounds [79]. This allowed Szegedy *et al.* to achieve improved accuracy relative to state-of-the-art results on image classification and detection tasks, at a lower memory and computational cost [79].

3.5.4. Inception V3.

The original Inception architecture was not directly used in this project. Instead a subsequent version also by Szegedy *et al.*, Inception V3, was used. Inception V3 achieved a reduced number of parameters and decreased computational complexity, as compared to the original Inception architecture, via the factorization of the larger convolutional blocks into smaller, stacked convolutional units that could be calculated more efficiently, used alongside a pooling operation with a larger stride [80]. This model also utilized some additional improvements to the structure and training process, namely additional batch normalization, label smoothing, and the use of an improved optimizer [80].

3.5.5. Xception.

The second Inception-derived model used in this project was Xception [81]. In designing this model, Chollet *et al.* expanded the idea in [80] that larger, more expensive convolution operations (3×3, 5×5, et cetera) could be factorized into smaller, stacked convolution operations that were, overall, less computationally expensive [81]. The Xception architecture relies on stacks of convolutional blocks similar to the maximal factorization of this kind, where first a **depthwise convolution** is performed, handling each color channel independently, and a **pointwise** (1×1) **convolution** follows [81]. These **depthwise separable convolutions** rely on the assumption that correlations across color channels and correlations across spatial locations can be considered separately [81].

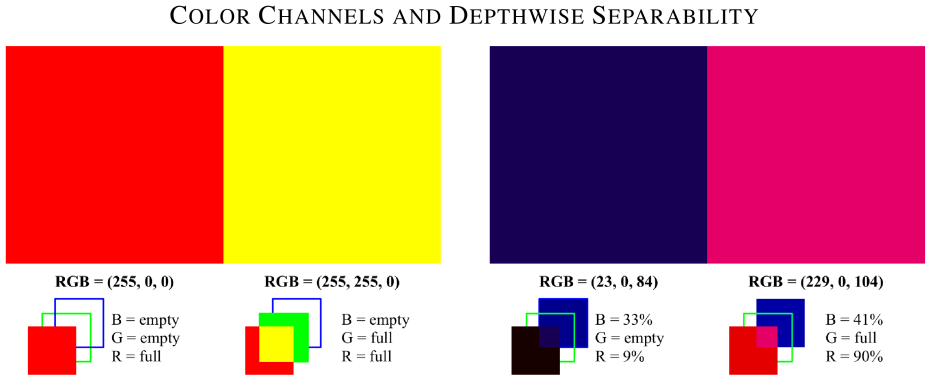


Figure 8: A visualization of depthwise separability, for native image data (left) and for colormapped wavelet data (right). On the left, a red square and a yellow square are shown with the by-channel breakdown of these colors. Across the border between the red square and the yellow square, the red channel remains the same and the green channel changes by 100%. In native image data, where any color is possible, borders of this type, where the behavior of the individual color channels are independent, are possible. In colormapped data, however, not all colors are possible, and the colormap is designed to alter color channels in conjunction with one another. On the right, squares of a shade of purple and a shade of pink from the `magma` colormap used in this project are shown. Across the border between the squares, the red channel and blue channel both change proportionally to the distance between these two colors in the colormap. Colormapped images are constrained such that all regions of differing colors will demonstrate a similarly proportional relationship across their borders, with the magnitude of change in each channel determined by the distance between those two colors in the colormap.

3.5.6. A Caveat Regarding Depthwise Separability.

It should be noted that this assumption of depthwise separability makes intuitive sense for native image data, where all colors are possible and channel to channel (e.g. red to blue, or red to green) changes and spatial (pixel location to pixel location) changes are not intrinsically related to one another. However, it is less likely that such an assumption is useful in the artificial, color-mapped world of the wavelet data used in this project. For example, the RGB tuplet (255, 0, 0) represents bright red. The RGB tuplet (255, 255, 0) represents bright yellow. A bright red rectangle and a bright yellow rectangle next to each other have identical red-channel values across the boundary between the rectangles, but very different green-channel values across the boundary (Figure 8). In this case, the green channel distinguishes between pixels whose red channel might otherwise suggest similarity. In another instance (a magenta rectangle and a blue rectangle), it might be the red channel that suggested the difference. While arbitrary color boundaries are common in native image data (e.g. a fire truck and a school bus parked next to a lake), the use of the "magma" colormap in the generation of wavelet images causes color channels to vary in conjunction with one another to produce a perceptually uniform colormap [67]. This introduces an artificial correlation across color channels that may impact the performance of Inception-based models (Figure 8).

3.5.7. ResNet50.

The final convolutional architecture used in this project was ResNet50. ResNet50 is a 50-convolutional layer architecture that was designed to improve training accuracy in deep networks by adding some shallower connections that run through the architecture parallel to its deep connections [82]. Specifically, some layer outputs not only form the input to the layer immediately following them in the network stack, but are also

Model Name	Model Size	Accuracy	Parameters	Layer Depth	Minimum Input Size (Width × Height × Channels)
Xception	88 MB	0.790	22,910,480	126	71×71×3
InceptionV3	92 MB	0.788	23,851,784	159	139×139×3
ResNet50	99 MB	0.759	25,636,712	168	197×197×3
VGG16	528 MB	0.715	138,357,544	23	48×48×3

Table 2: Statistics for pre-trained Keras models, with top-1 accuracy when run on ImageNet[74], sorted by model size. Note that while the top-5 accuracy is also often provided as a meaningful metric for ImageNet, which has hundreds of classes[19], top-5 accuracy was not used for this project because the datasets used have only 8 classes[20], so top-5 accuracy is likely to be deceptively high.

taken as-is and summed with the output of a subsequent, deeper layer [82, Figures 2 and 3]. These **shortcut connections** allow ResNet (residual) architectures to be optimized more easily than comparable non-residual architectures [82].

While combinations of the ResNet and Inception concepts exist, such as the InceptionResNetV2 architecture proposed by [83] and available via Keras [74], these models were not investigated as part of this project.

3.5.8. Initialization and Training.

For this project, in each of Experiments 3–5, an instantiation of the two-layer network initialized with random weights was used as the classifier for each of the four pre-trained models and additional training was performed to refine the network's performance. The two-layer network, initialized with random weights, was also trained as an independent model. Whenever the two-layer network was being trained alone, either as an independent network or as the classifier at the output of one of the pre-trained models, an RMSprop optimizer with a relatively high learning rate was used. However, whenever the layers embedded in the convolutional networks were being trained, a slower, standard SGD optimizer was used to prevent large updates from overwriting the ImageNet pre-training.

The full structure of all models used in this project can be found in Appendix A.

4.1. Overview.

Five experiments were conducted in this project. In Experiment 1, support vector classification with varying kernel types was used to classify the MFCC features for the FMA small and FMA extended datasets. These values were then used as internal benchmarks for later experiments. In Experiment 2, cross-validation with a short training time was used to obtain good hyperparameters for the RMSprop optimizer used to train the classifier. In Experiment 3, all five neural network models were trained for 30 epochs on the FMA small dataset, without the use of data augmentation. In Experiment 4, all five neural network models were trained for 30 epochs on the FMA small dataset, with a varying amount of horizontal shift added to augment the training data and increase training-time variation. In Experiment 5, all five neural network models were trained for 30 epochs on the FMA extended dataset, both with and without data augmentation.

4.1.1. The 6G200E Benchmark.

For all five experiments, the base benchmark used for genre classification was the training accuracy achieved by Li *et al.*, who, in 200 training epochs, achieved a training accuracy of approximately 40% (or an error of approximately 60%) using a single convolutional neural network with three convolutional layers to classify MFCC data from a six-genre dataset [18]. In the interests of space, this approximate 40% accuracy base benchmark is marked “6G200E” in graphs of experimental results throughout this project.

A few caveats regarding comparisons to this study, however, must be mentioned before proceeding. First, the dataset used in [18] was not the FMA dataset [20] used

in this project, but a subset of the GTZAN dataset [84], which is significantly smaller, with only 100 tracks per genre. Benchmarking data using the FMA dataset is presently limited because [20] is still in pre-publication. Second, the approximate 40% accuracy achieved after 200 epochs in [18] is a *training* accuracy, not a *validation* accuracy, and because of the tendency of neural networks to overfit on the training data, particularly when using small data sets [55, 61], this benchmark is therefore not directly comparable to the benchmarks derived in Experiment 1, below. The expectation, however, is that a validation accuracy would be lower than a training accuracy, in keeping with the definition of overfitting [55], so accuracies exceeding this value would qualify as meaningful results. Finally, the final overall accuracy achieved by Li *et al.* was substantially higher than the 40% achieved by their CNN alone: namely, they achieved a final accuracy over 84% using a combination of tree classifiers and majority voting [18]. However, because this ensemble model was substantially more complex than any of the single models tested in this project, that value provides a less useful benchmark for the purposes of this project.

4.1.2. Memory Management.

It should also be noted that, due to the resource limitations of the free trial-available virtual machines available on Google Cloud Engine [58] and some known issues with memory fragmentation and leaks in Keras's TensorFlow backend [85, 86], it was desirable to frequently **checkpoint** the learning process and allow Keras to clear memory by stopping and restarting training after a certain number of training steps [85] and by clearing the backend in between model runs [86]. The memory problems these techniques addressed tended to increase with both increasing size of the model and increasing size of the dataset, making more frequent checkpointing necessary.

For Experiments 3 and 4, it was found to be sufficient to stop and restart training after some number of whole epochs. However, for Experiment 5, the larger dataset

meant that a single epoch contained more training steps, and it was found that on the larger models (especially VGG16), an individual epoch was too long to run without checkpointing the training process mid-epoch to clear fragmented memory. For code simplicity and reusability, therefore, Experiment 5 was run on the basis of quarter-epochs: i.e., the code was written to consider approximately a quarter of the training data in each "epoch" (actually a quarter-epoch) but to run for four times as many "epochs" (actually quarter-epochs). The primary effect of handling the extended dataset in this manner is that the results shown for Experiment 5 contain four times as many data points per run as do Experiments 3 and 4 (one per quarter-epoch, or four total per epoch). Outside of this explanation, all graphing and analysis in this paper identifies *only* full epochs as epochs, and specifies "quarter-epoch" when necessary. However, the accompanying Python code in places uses "epoch" to interface correctly with Keras, when in truth quarter-epoch would be more accurate. These instances are heavily commented in the accompanying Python code.

4.2. Experiment 1: MFCC Benchmarking.

In Experiment 1, newly-instantiated SVC models using each of the four kernel options (linear, polynomial, RBF, and sigmoid) were fit to the training split of the FMA small dataset, after subtracting the mean and adjusting to unit variance. The four resulting trained SVC models were then used independently to predict the classes for the validation split of the FMA small dataset.

A similar training and prediction operation was performed for the FMA extended dataset: the mean was subtracted from the training split and the variance was adjusted to 1. The preprocessed data was then used to train newly-instantiated SVC models using each of the four kernel options, and the four resulting trained SVC models were then used independently to predict the classes for the validation split of the FMA small dataset.

The final classification accuracies of the trained models, evaluated on both the training split and the validation split, are graphed in Figure 9.

These SVC models could, in theory, also be trained on the wavelet features for either or both dataset sizes. However, because of the large size of the wavelet feature set even for the FMA small dataset (8000 images at 256×256×3 pixels per image, corresponding to 1,572,864,000 total 8-bit pixel values, or nearly 1.6 GB, excluding overhead), persistent errors were encountered when attempting to load the wavelet feature set into memory for processing by scikit-learn. Therefore, the SVC models instantiated with scikit-learn were trained *only* on the MFCC data, which is over a thousand times smaller than the wavelet data.

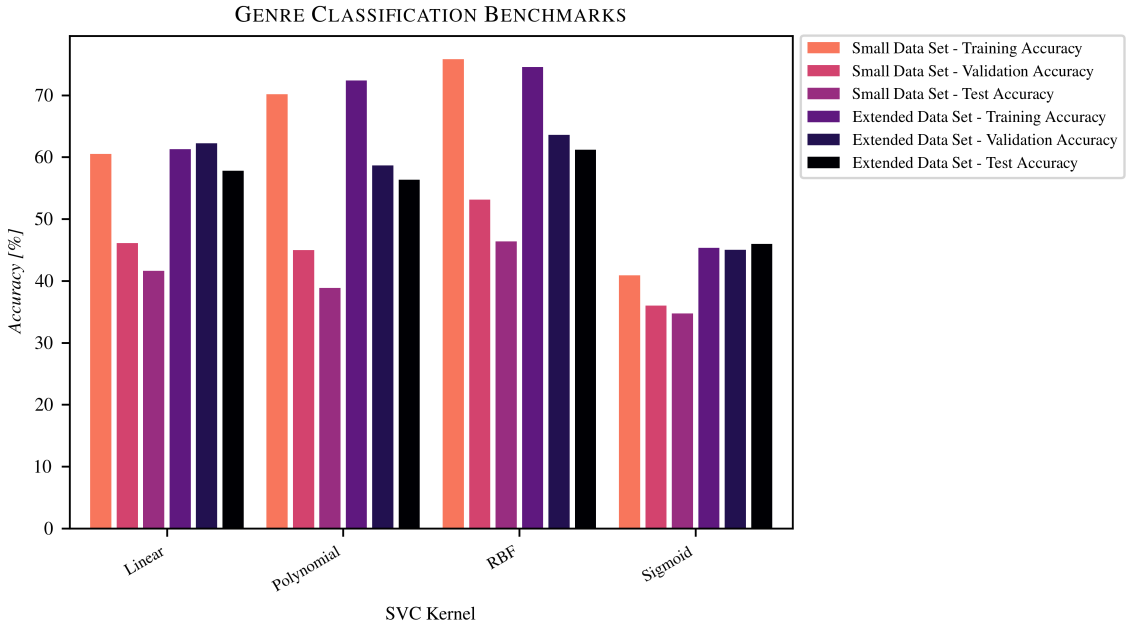


Figure 9: Results of genre classification benchmarking experiment using support vector classification on the Mel-frequency cepstral coefficients for the specified FMA data subsets.

For both the FMA small dataset and the FMA extended dataset, the RBF kernel outperformed the other three kernels, with a test accuracy of 46% on the small dataset and 61% on the extended dataset. By contrast, the sigmoid kernel underperformed the

other three kernels, achieving only 35% test accuracy on the small dataset and 46% validation accuracy on the extended dataset.

With the exception of the linear kernel on the extended dataset, the training accuracies exceeded the validation and test accuracies, often by a large margin. This problem was particularly an issue with the polynomial kernel, across both datasets, and particularly an issue on the small dataset, across all kernels. For example, the training accuracy for the polynomial kernel on the small dataset exceeded the validation accuracy by 25 percentage points and the test accuracy by 31 percentage points. This suggests that either the polynomial kernel, or the default order of 3 used for the polynomial kernel, yields a poor mapping for these data.

The best and worst *validation* accuracies (for the RBF and sigmoid kernels, respectively) for the appropriate dataset were also used as benchmarks for Experiments 3, 4, and 5, and therefore also appear on training history graphs for those experiments. The validation accuracies were used as benchmarks on these graphs in preference to the test accuracies because the training history graphs contain only training and validation history data for these experiments. The corresponding test accuracy results are presented separately, in Tables 3, 4, and 5.

4.3. Experiment 2: Optimizer Cross-Validation.

In Experiment 2, the two-layer network alone was trained on the FMA small dataset for a single epoch, using varying values for the RMSprop hyperparameters: learning rate, learning rate decay, ρ/`rho`, and ε/`epsilon`, to find a "good" optimizer to use for training the two-layer network (the classifier) in later experiments. Specifically, the goal was to identify the best optimizer within all possible combinations of a limited but plausible range of hyperparameters tested. The values tested for each hyperparameter were suggested by the Stanford CS231n course materials [45, 46, 47] and extended and refined following repeated experimentation.

A total of 300 optimizers were thus examined. The best optimizer achieved a validation accuracy of 30.4% in a single epoch at learning rate, `lr`$= 0.01$; learning rate decay, `decay`$= 0.01$; `rho`$= 0.85$; and `epsilon`$= 1 \times 10^{-10}$.

A visualization of how varying hyperparameters affect the final validation accuracy can be found in Figure 10.

While this variation was by no means exhaustive and the ability to visualize the optimization problem two-dimensionally is limited, a few trends do stand out. First, considering the "Rho vs. Learning Rate" plot (Figure 10, middle left), the higher the value of ρ, the higher the learning rate had to be for equivalent performance—up to a point. Note that mean performance decreased for increasing ρ at the highest learning rate tested. However, the best performance overall occurred at moderate ρ and high learning rate, illustrating the limitations of this graphical approach for evaluating performance across four variable hyperparameters. Second, the presence of both particularly dark and particularly light values in many of the graphs (e.g. "Rho vs. Learning Rate Decay", "Log Epsilon vs. Learning Rate Decay", Figure 10) suggests that there are pairwise combinations of individual hyperparameters that are particularly bad or particularly good, regardless of the value of the other hyperparameters.

In Experiments 3, 4, and 5, whenever an RMSprop optimizer was used, it was this best RMSprop optimizer found via cross-validation (i.e., an RMSprop optimizer with `lr`$= 0.01$, `decay`$= 0.01$, `rho`$= 0.85$, and `epsilon`$= 1 \times 10^{-10}$). This optimizer was used whenever the two-layer network (the classifier) was being trained alone.

4.4. Experiment 3: Small Dataset Without Augmentation.

In Experiment 3, each of the five neural network models were trained for 30 epochs on the training split of the FMA small dataset, after subtracting the mean and adjusting to unit variance.

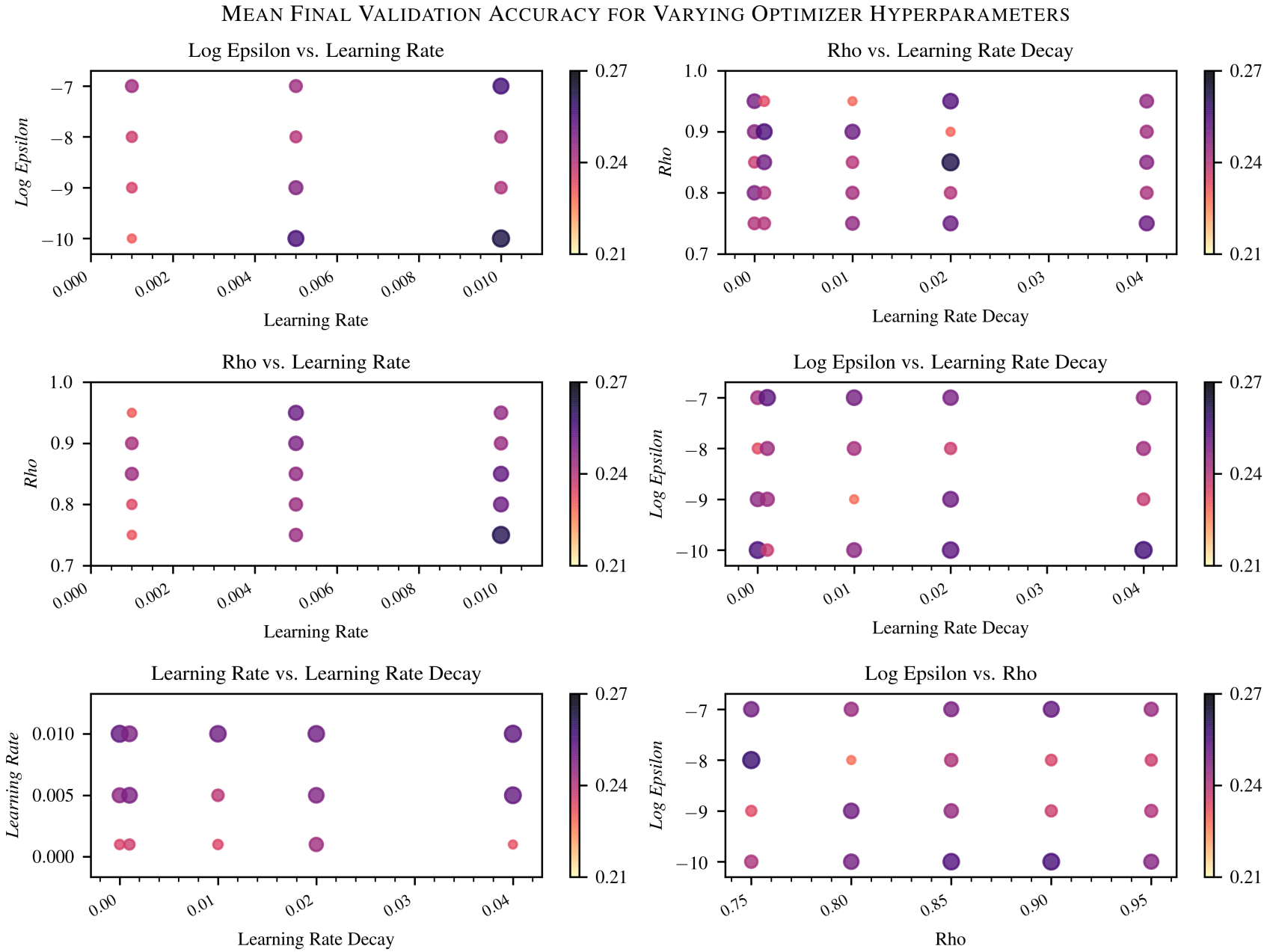


Figure 10: Final validation accuracy after 1 training epoch, for each pair of optimizer hyperparameters, using an RMSprop optimizer. Since a total of four hyperparameters were being altered during cross-validation, each graph shows the mean results for the multiple training runs at the specified values, during which the other two hyperparameters were varied.

For the two-layer neural network by itself, no changes were made during the 30 training epochs, and the best RMSprop optimizer found in Experiment 2 was used throughout. For all the pre-trained models, however, training was divided into six five-epoch phases. In the first phase, only the classifier was trained, while all layers of the pre-trained model were **frozen**, i.e., the Keras training process was not permitted to update their weights. The best RMSprop optimizer found in Experiment 2 was used for this training phase. In each subsequent phase, some number of layers in the pre-trained model were unfrozen, so that the Keras training process could update their weights along with the weights of the classifier. For training phases 2–6, in which updates were made to the pre-trained layers, an SGD optimizer with a low training rate was used, to prevent large updates that would overwhelm the pre-trained weights. This general training process was adapted from [63].

Keras provides detailed epoch-by-epoch information during training, namely the training loss, the validation loss, the training accuracy, and the validation accuracy at the end of each epoch. This detailed training and validation history for Experiment 3 appears in Figure 11 (the two-layer network), Figure 12 (Inception V3), Figure 13 (Xception), Figure 14 (ResNet50), and Figure 15 (VGG16). A comparison of the validation accuracy throughout training for all five models appears in Figure 16 . Because the validation accuracy fluctuates somewhat even late in training, both the final validation accuracy (at the end of Epoch 30) and the final-phase mean validation accuracy (for Epochs 26–30) appear in Table 3.

In terms of validation accuracy, ResNet50 and VGG16 outperformed both the 6G200E model [18] and the sigmoid SVC (the worst of the SVC models) on the validation set by the end of 30 training epochs. However, no models achieved the performance of the RBF SVC, and the other 3 models (the two-layer network alone, Xception, and Inception V3) failed to exceed even the lowest benchmark, with only Inception V3 performing comparably to the sigmoid SVC model. It should also be reiter-

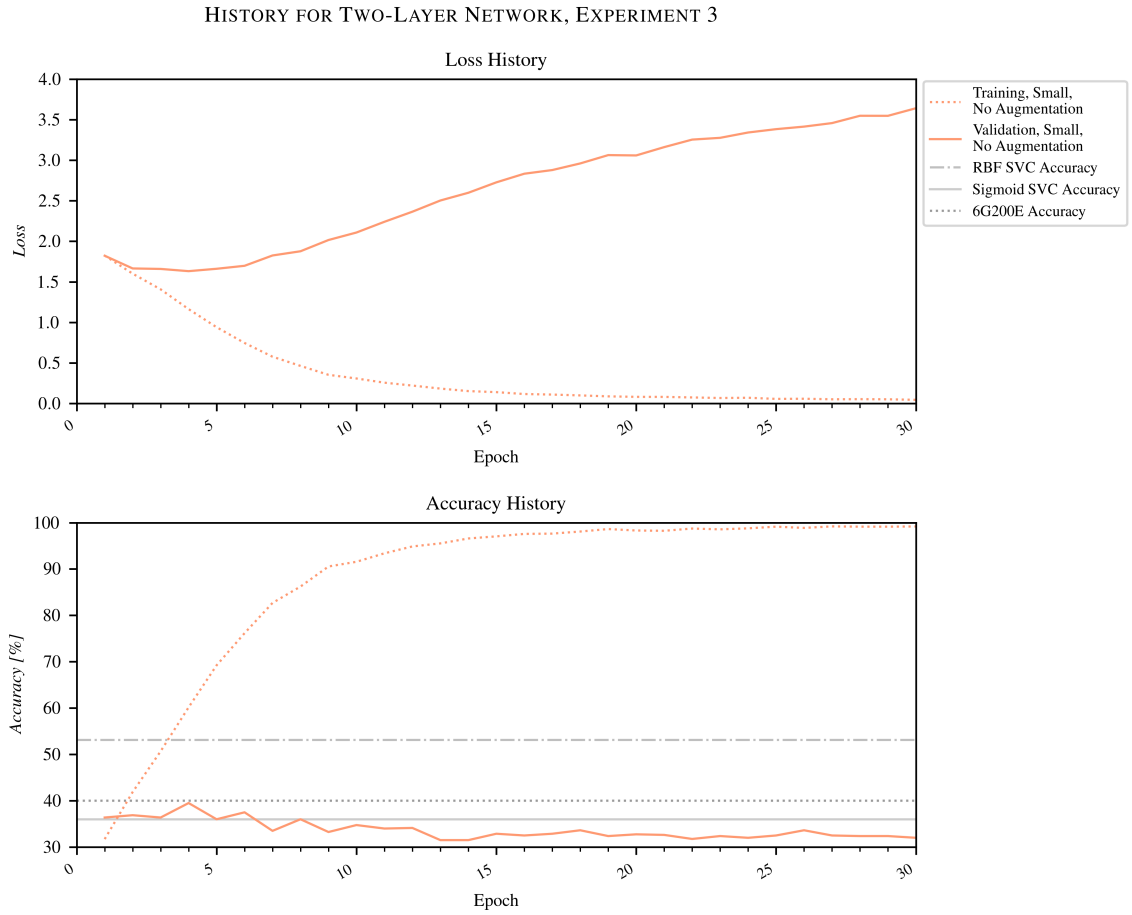


Figure 11: Loss and accuracy history for two-layer network (the classifier), using the FMA small dataset, with no data augmentation. When the two-layer network is training alone, no changes are made between the six 5-epoch training phases. Overfitting can be seen by Epoch 5, after which the training loss and accuracy continue to improve, but the validation loss and accuracy continually deteriorate. This problem is characteristic of small datasets.

Model	Final Validation Accuracy	Final Test Set Accuracy
Two-layer network	32.0%	30.1%
Inception V3	32.1%	29.9%
Xception	36.3%	33.8%
ResNet50	42.3%	38.0%
VGG16	44.4%	39.1%

Table 3: Statistics for all models trained for 30 epochs on the FMA small dataset, without data augmentation. Both the final validation accuracy (at Epoch 30) and the final accuracy of the trained model on the test set are provided. While the final test and validation accuracies of VGG16 were the highest of the five models, its overall performance is roughly comparable to ResNet50, in part because VGG16 exhibits significant variation late in training. This variation can also be seen in Figure 16 and suggests that VGG16 may be more vulnerable than ResNet50 to the precise epoch at which training is terminated.

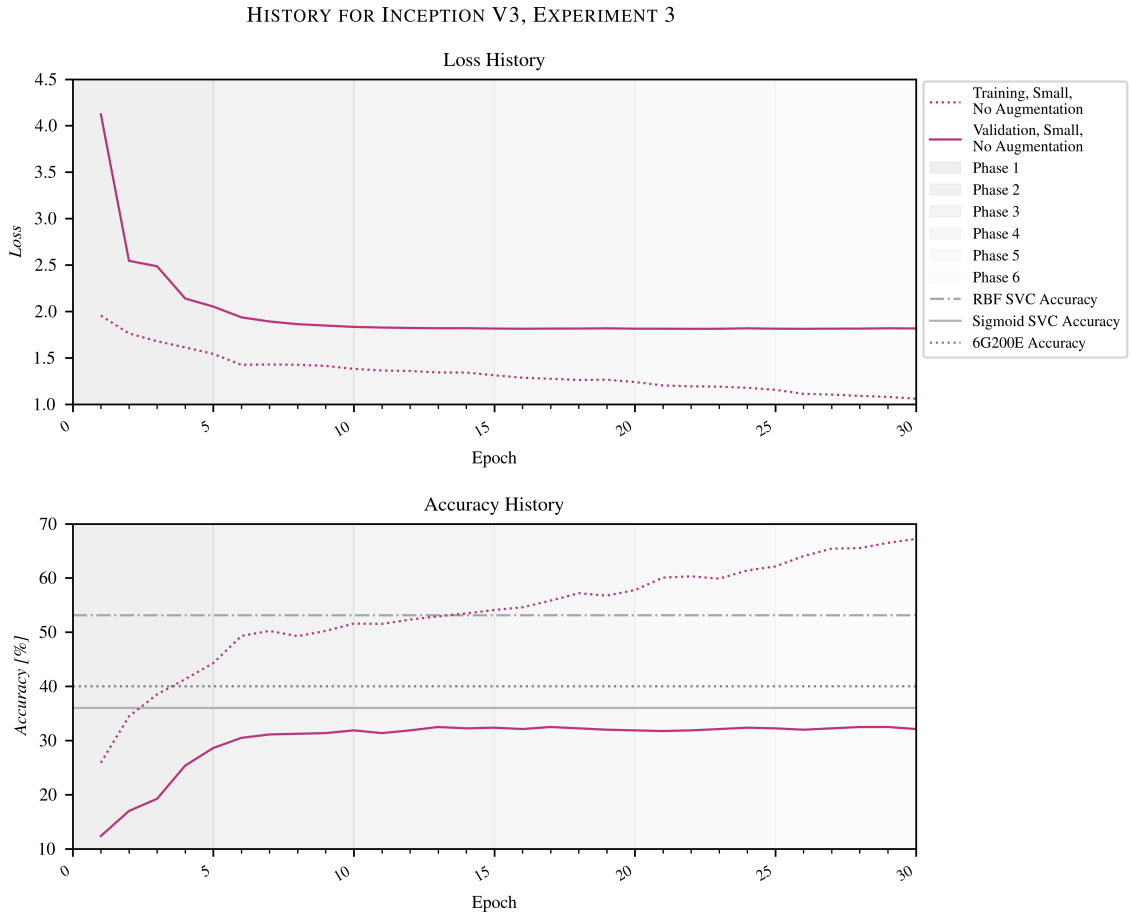


Figure 12: Loss and accuracy history for Inception V3, pre-trained on ImageNet, followed by the classifier. In this experiment, Inception V3 was trained on the FMA small dataset, with no data augmentation. When training Inception V3, every five epochs a new section of the model was unfrozen for training. Phase 1 trained the classifier alone, Phase 2 trained layers 249 and above, Phase 3 trained layers 232 and above, Phase 4 trained layers 229 and above, Phase 5 trained layers 200 and above, and Phase 6 trained layers 187 and above. The full list of model layers can be found in Appendix A. Early overfitting is suggested by the increasing size of the gap between training and validation data after Epoch 10.

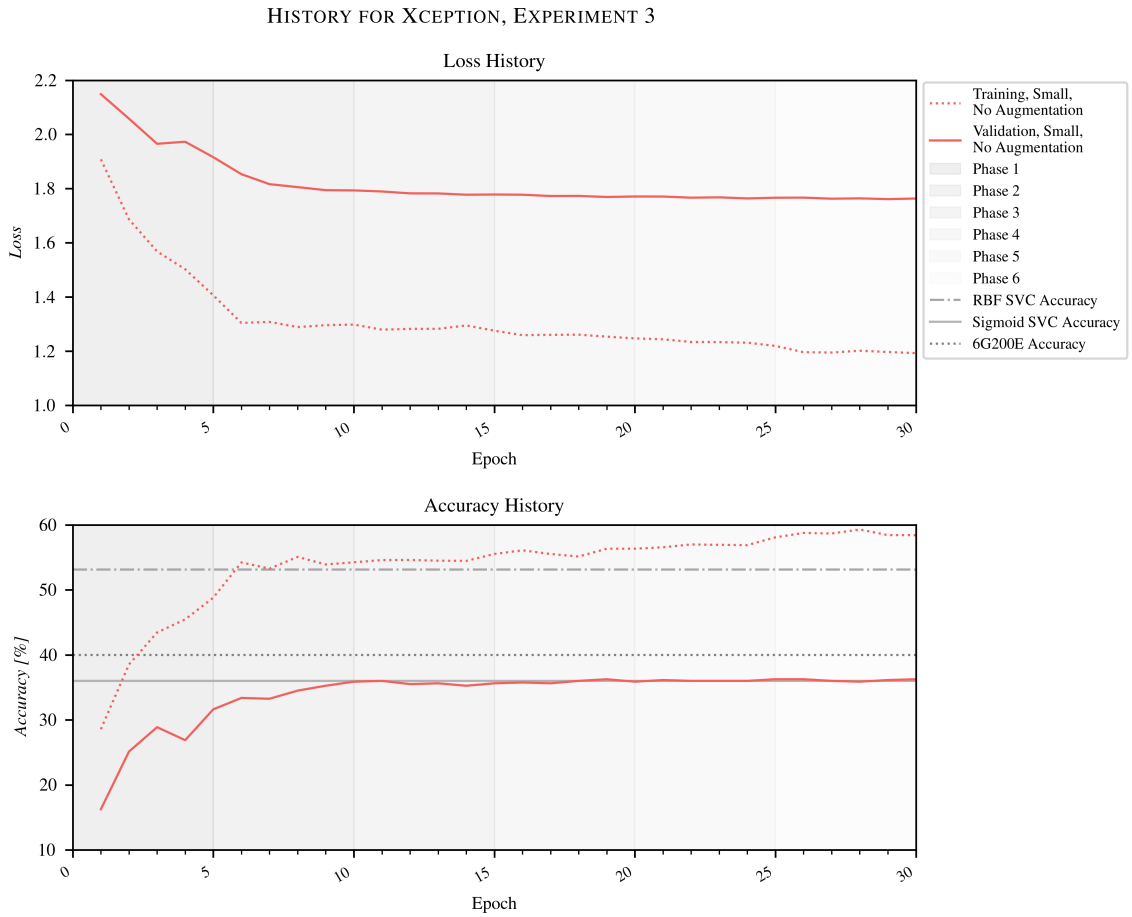


Figure 13: Loss and accuracy history for Xception, pre-trained on ImageNet, followed by the classifier. In this experiment, Xception was trained on the FMA small dataset, with no data augmentation. When training Xception, every five epochs a new section of the model was unfrozen for training. Phase 1 trained the classifier alone, Phase 2 trained layers 122 and above, Phase 3 trained layers 105 and above, Phase 4 trained layers 95 and above, Phase 5 trained layers 85 and above, and Phase 6 trained layers 75 and above. The full list of model layers can be found in Appendix A. While Xception does not demonstrate the clear overfitting seen when training the classifier alone (Figure 11), the large difference in performance between training and validation accuracies seen here is likewise typical of the small size of the dataset.

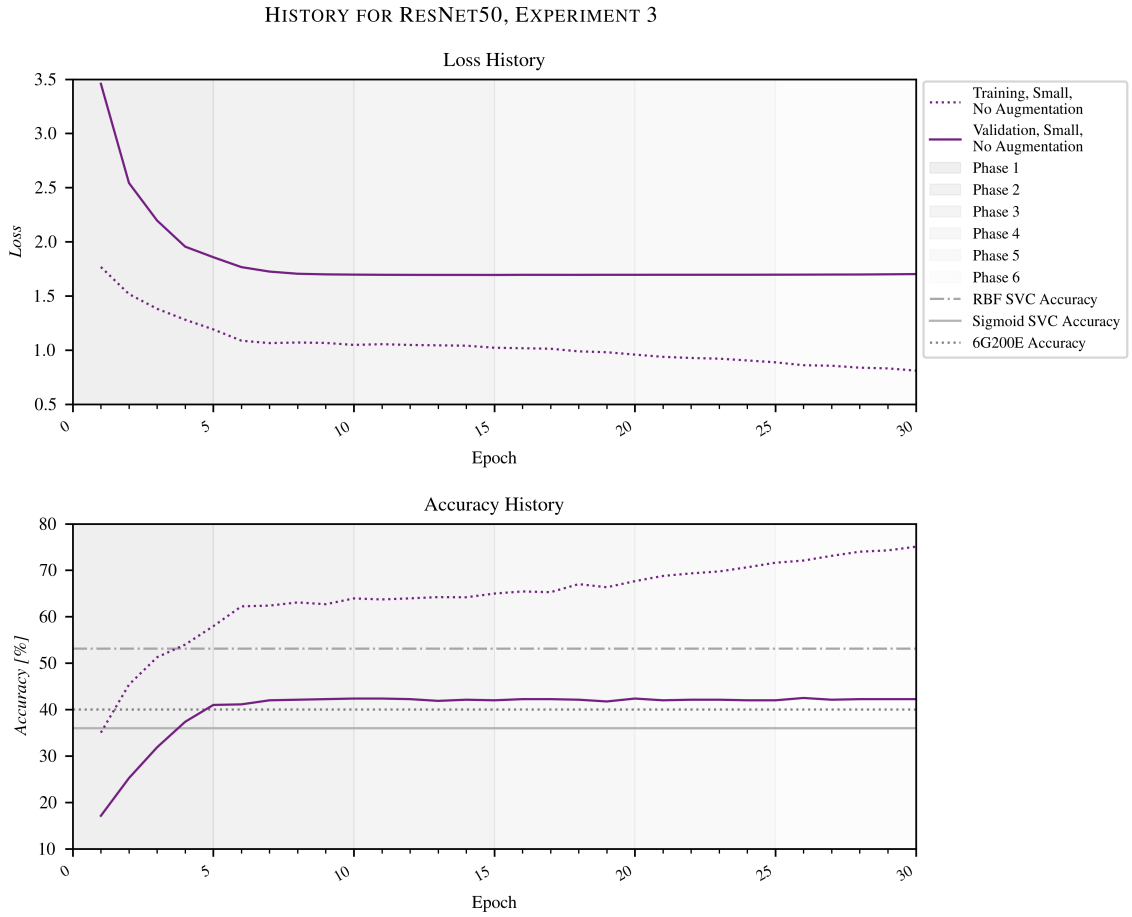


Figure 14: Loss and accuracy history for ResNet50, pre-trained on ImageNet, followed by the classifier. In this experiment, ResNet50 was trained on the FMA small dataset, with no data augmentation. When training ResNet50, every five epochs a new section of the model was unfrozen for training. Phase 1 trained the classifier alone, Phase 2 trained layers 161 and above, Phase 3 trained layers 151 and above, Phase 4 trained layers 139 and above, Phase 5 trained layers 129 and above, and Phase 6 trained layers 119 and above. The full list of model layers can be found in Appendix A. Early overfitting is suggested by the increasing size of the gap between training and validation data after Epoch 15.

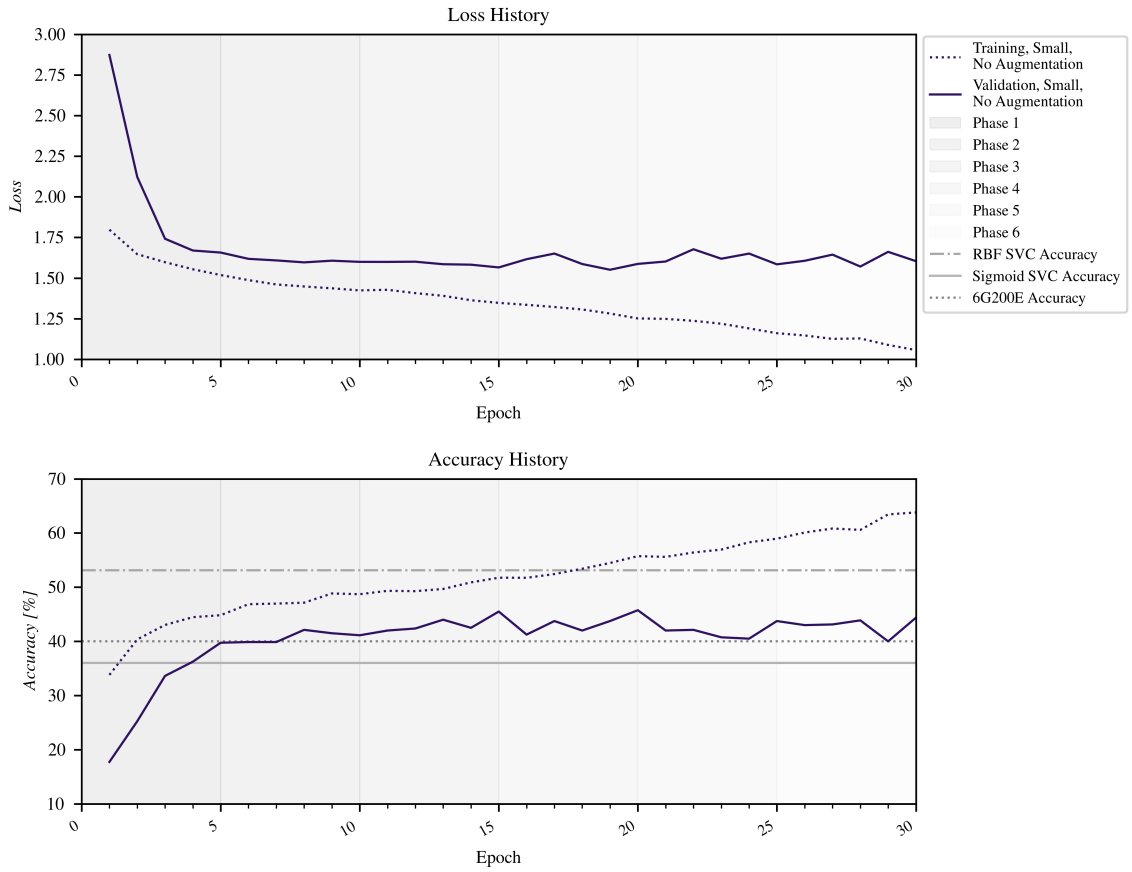


Figure 15: Loss and accuracy history for VGG16, pre-trained on ImageNet, followed by the classifier. In this experiment, VGG16 was trained on the FMA small dataset, with no data augmentation. When training VGG16, every five epochs a new section of the model was unfrozen for training. Phase 1 trained the classifier alone, Phase 2 trained layers 17 and above, Phase 3 trained layers 15 and above, Phase 4 trained layers 13 and above, Phase 5 trained layers 11 and above, and Phase 6 trained layers 8 and above. The full list of model layers can be found in Appendix A. Early overfitting is suggested by the increasing size of the gap between training and validation data after Epoch 15.

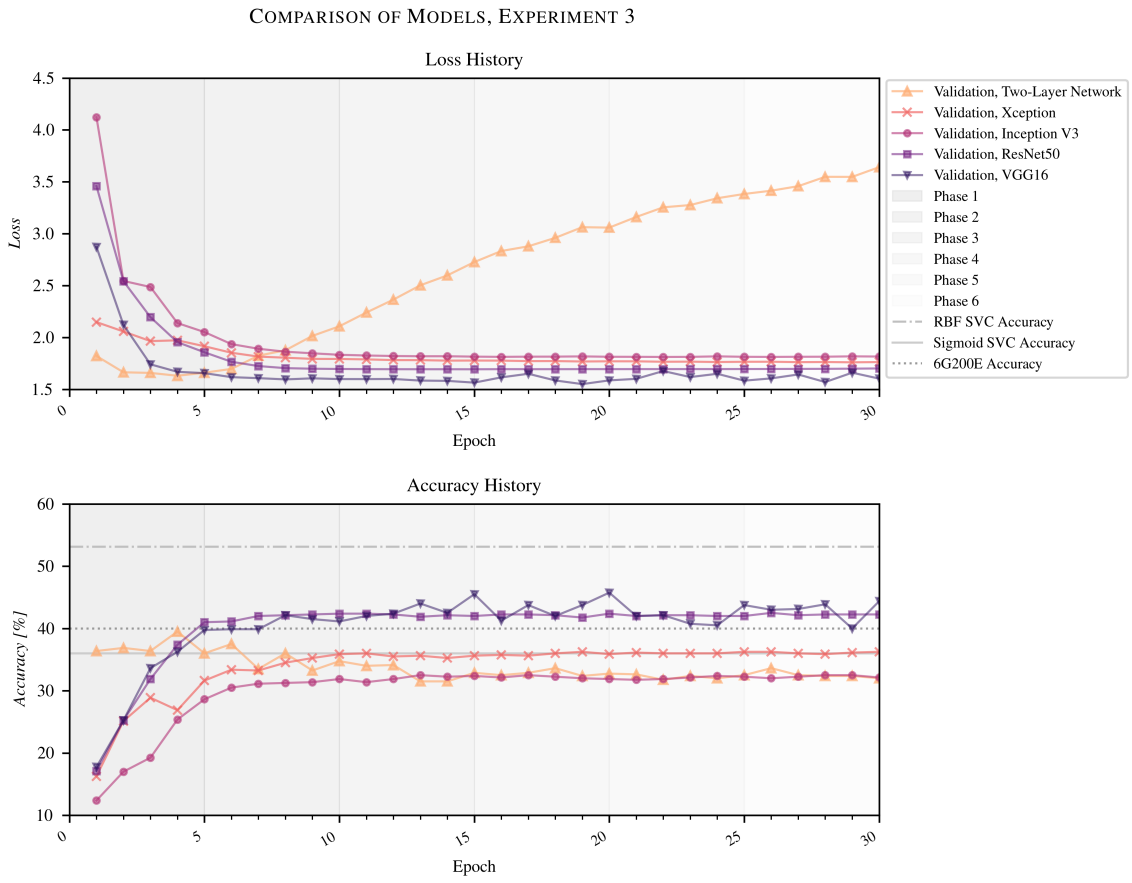


Figure 16: A comparison of the validation loss and accuracy for all five models, run on the small FMA dataset with no data augmentation. The vertical axis for the accuracy history has been zoomed to more clearly show the variation between the models. Note again the strong overfitting on the two-layer network (the classifier) alone, clearly visible in the increasing validation loss. While some signs of overfitting were present for the other networks, this effect was most significant when the classifier was run alone.

ated that VGG16 demonstrated more fluctuation in accuracy late in training than did ResNet50 (Figure 16, suggesting that it may be more vulnerable to small variations in training duration.

Overall, VGG16 and ResNet50 demonstrated the best performance on the FMA small dataset without data augmentation, both near 38% or 39% final test set accuracy. These accuracies were comparable to the polynomial SVC model (39%), but did not achieve the accuracies of the better linear (42%) or RBF (46%) SVC models.

4.5. Experiment 4: Small Dataset With Augmentation.

In Experiment 4, each of the five neural network models were trained for 30 epochs on the training split of the FMA small dataset, after subtracting the mean and adjusting to unit variance, as in Experiment 3. However, in Experiment 4, the Keras `ImageDataGenerator` module was also used to augment the data by shifting the wavelet images horizontally by a random amount [63]. While other types of data augmentation exist, such as zoom, horizontal flip, and rotation [87], they were designed for use with native image data, and were not used in this project due to their implicit meaning, were they to be used on the wavelet images. For example, an image of a cat flipped horizontally would still be, to humans, visually recognizable as a cat. However, flipping one of the wavelet images used in this project horizontally would be equivalent to time-reversing the audio clip from which the wavelet image was derived. It is not clear that the genre of a time-reversed clip would be identifiable to a human listener. Likewise, the wavelet transform generating these images was designed to cover the whole human auditory spectrum from 20 Hz to 20 kHz [22, p. 80]. Vertical shift or zoom would cut off some of this frequency content, and would therefore implicitly alter the spectrum encoded in the wavelet image.

The amount of horizontal shift was controlled by a *data augmentation factor* of either 0.33, 0.5, or 0.66. This value corresponds to a maximum permissible horizontal

shift, i.e., a shift of 33% or less, 50% or less, or 66% or less [87]. This has an effect much like windowing the audio clip: since the source audio clips in the FMA dataset are 30 seconds long [20], this corresponds to taking sections of the audio clip that are 20 to 30 seconds long (for a data augmentation factor of 0.33), 15 to 30 seconds long (for a data augmentation factor of 0.5), or 10 to 30 seconds long (for a data augmentation factor of 0.66), with both the length and the start time selected at random by the Keras data generator. However, because the neural networks expect consistently sized input images, for shifted data, the pixels that would otherwise have been left empty due the windowing implicit in the shift operation were filled by means of the `'nearest'` `fill_mode` option, i.e., the pixel data at the edge of the selected window was extended out to the full 256-pixel width of the image, as necessary. Preliminary testing during development suggested that this lead to better results than the `'constant'` (with the constant value set to 0) or `'wrap'` options, but exhaustive testing of the various methods for extending images during data augmentation was beyond the scope of this project. The `'reflect'` option was not considered, for the same conceptual reasons that a horizontal flip was not used for data augmentation: it is not clear that time reversal of part of a song's spectrum would not alter the perceived genre of that song to a human listener.

As in Experiment 3, when training the two-layer neural network by itself, no changes were made during the 30 training epochs, and the best RMSprop optimizer found in Experiment 2 was used throughout. Also as in Experiment 3, for all the pre-trained models, training was divided into six five-epoch phases, using the best RMSprop optimizer found in Experiment 2 to train the classifier on the features extracted by the pre-trained model in phase 1, then an SGD optimizer with a low training rate for the remaining 5 phases, in which parts of the pre-trained model were successively unfrozen for updates.

The detailed training and validation history for Experiment 4 appears in Figure 17 (the two-layer network), Figure 18 (Inception V3), Figure 19 (Xception), Figure 20 (ResNet50), and Figure 21 (VGG16). A comparison of the validation accuracy throughout training for all five models appears in Figure 22. Because the validation accuracy fluctuates somewhat even late in training, both the final validation accuracy (at the end of Epoch 30) and the final-phase mean validation accuracy (for Epochs 26–30) appear in Table 4.

As can be seen in Table 4 and Figure 22, increasing the value for data augmentation does not necessarily lead to better results. While the best overall test set results were obtained at 66% augmentation using ResNet50, ResNet50 showed the least effect from data augmentation overall; and the next best overall result was achieved with the two-layer network and 33% augmentation.

With data augmentation, by the end of 30 training epochs, three of the models (the two-layer network, ResNet50, and VGG16) consistently outperformed the 6G200E model in terms of validation accuracy, and all models met or exceeded the performance of the sigmoid SVC (the worst of the SVC models). However, no models achieved the performance of the RBF SVC models in terms of either validation or test accuracy. The two-layer network showed the greatest overall improvement due to the introduction of data augmentation, a 9.3 percentage point improvement in final test set accuracy between no data augmentation and a data augmentation factor of 0.33. In contrast to Experiment 3, where the two-layer network demonstrated severe overfitting, when data augmentation is added, the two-layer network (the classifier operating alone) competes with ResNet50 and VGG16 and in fact consistently outperforms two of the convolutional neural networks, Xception and Inception V3. Inception V3, in fact, underperforms the other models (including the classifier alone) for all four values of data augmentation.

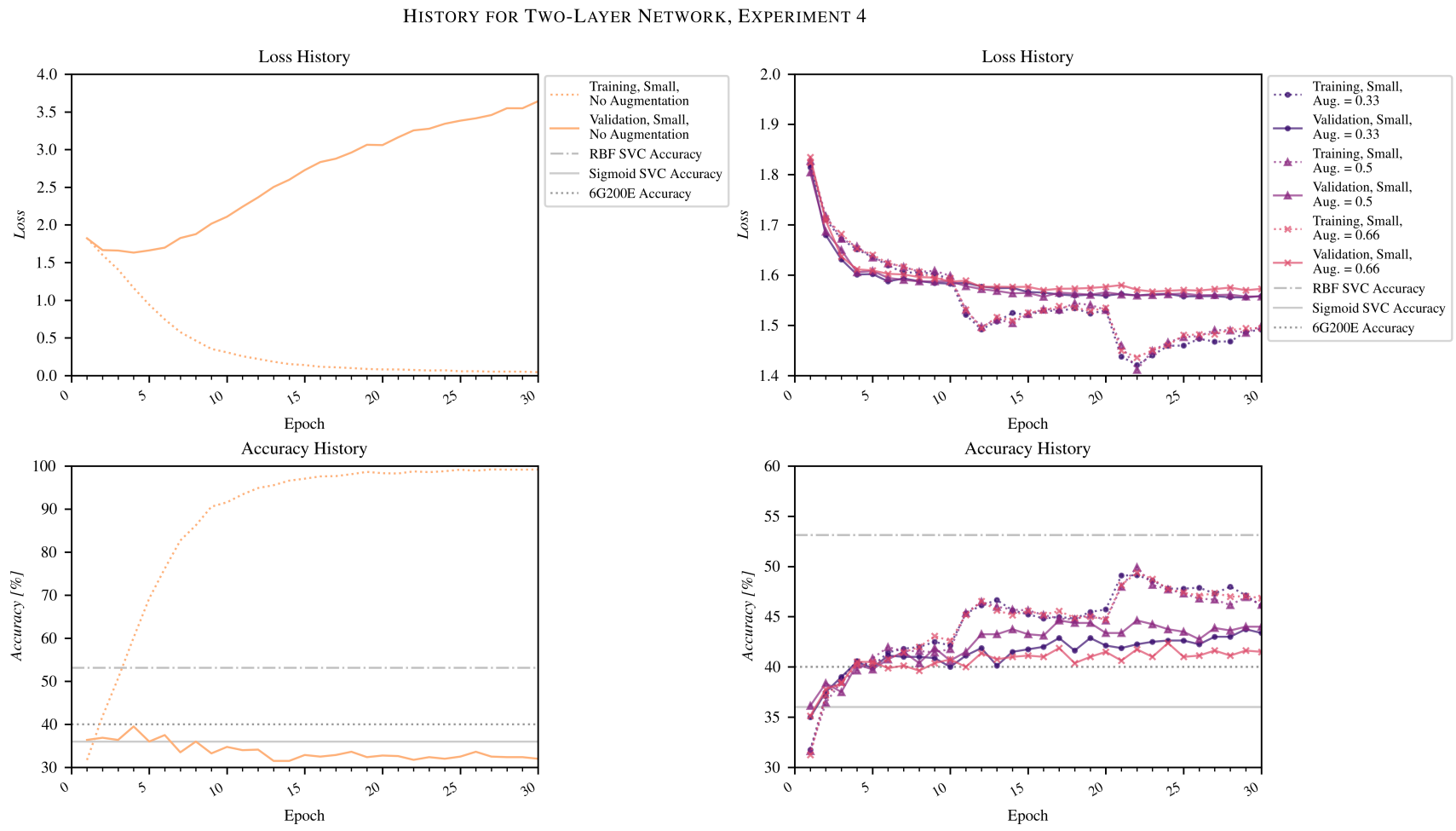


Figure 17: Loss and accuracy history for two-layer network (the classifier), using the FMA small dataset, contrasting performance with no data augmentation [left] and with data augmentation of 0.33, 0.5, and 0.66 (up to 33%, 50%, and 66% horizontal shift, respectively) [right]. As in Experiment 3, for the classifier training alone, no changes are made between the six training phases. Note that less overfitting occurs with data augmentation than does for the model without data augmentation. However, even with data augmentation, a gap in performance between training and validation data can still be observed. This is characteristic of early overfitting and is typical of a small dataset.

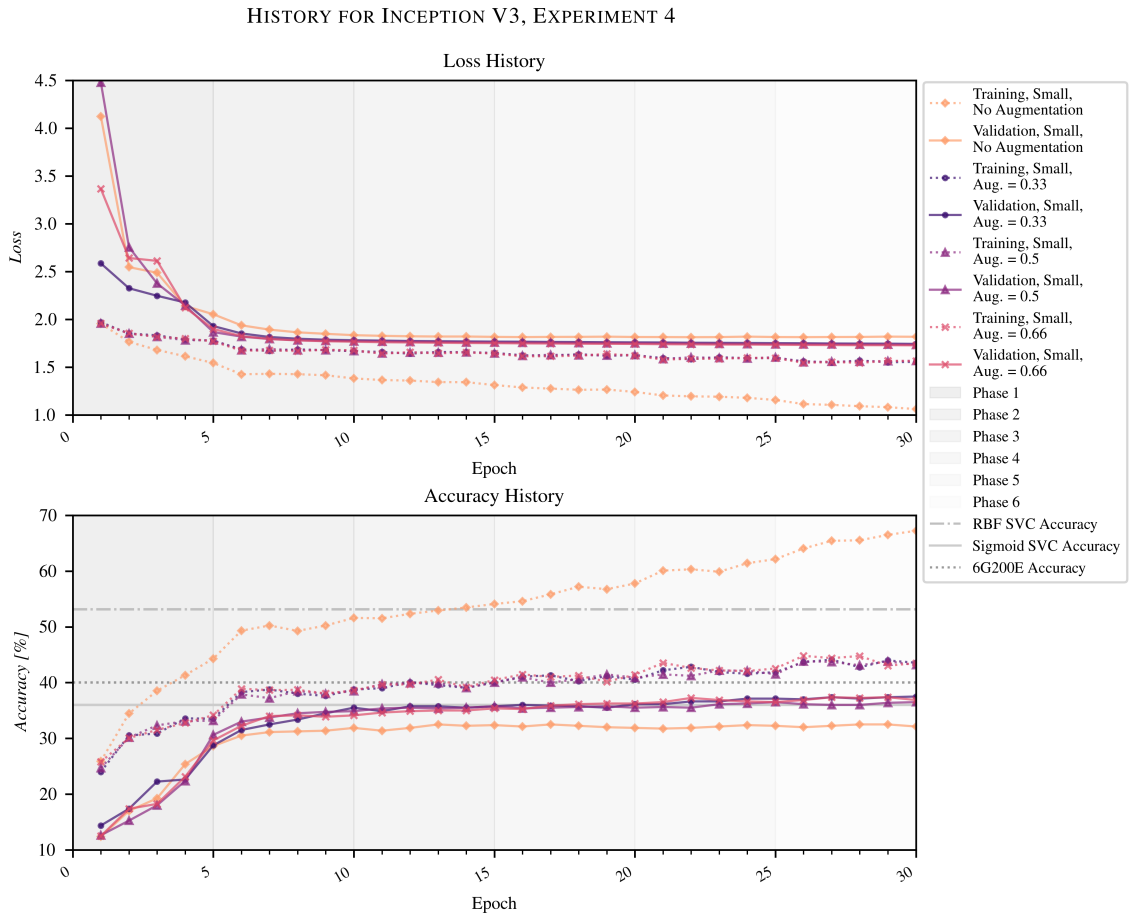


Figure 18: Loss and accuracy history for Inception V3, using the FMA small dataset, contrasting performance with no data augmentation and with data augmentation of 0.33, 0.5, and 0.66 (up to 33%, 50%, and 66% horizontal shift, respectively). As in Experiment 3, the first phase trained the classifier alone. Then layers 249 and above, 232 and above, 229 and above, 200 and above, and 187 and above were trained for Phases 2-6 respectively. As for the two-layer network (Figure 17), Inception V3 demonstrates less overfitting with data augmentation than without data augmentation. However, even with data augmentation, the small dataset-characteristic gap in performance between training and validation data can still be observed, suggesting early overfitting.

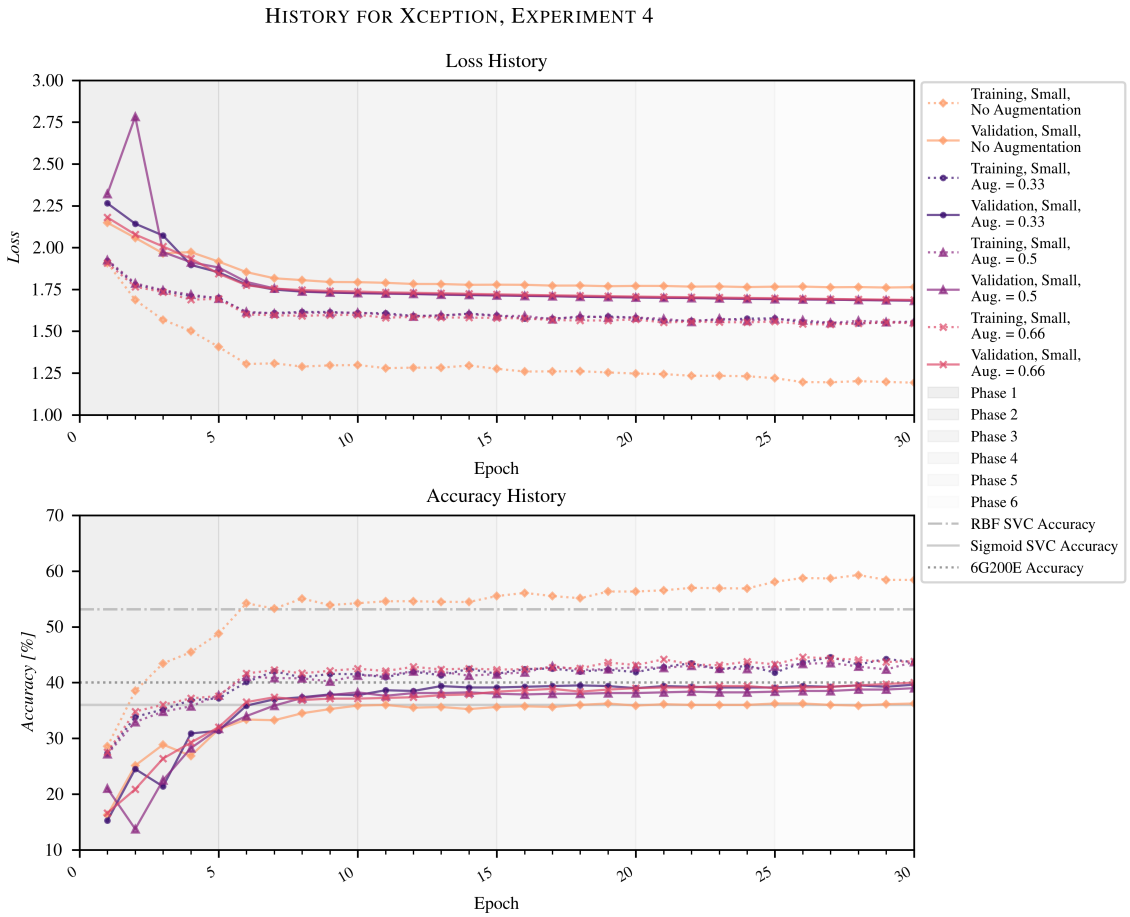


Figure 19: Loss and accuracy history for Xception, using the FMA small dataset, contrasting performance with no data augmentation and with data augmentation of 0.33, 0.5, and 0.66 (up to 33%, 50%, and 66% horizontal shift, respectively). As in Experiment 3, the first phase trained the classifier alone. Then layers 122 and above, 105 and above, 95 and above, 85 and above, and 75 and above were trained for Phases 2-6 respectively. As for earlier models (Figure 17 and and 18), Xception demonstrates less overfitting with data augmentation than without data augmentation. However, even with data augmentation, the small dataset-characteristic gap in performance between training and validation data can still be observed, suggesting early overfitting.

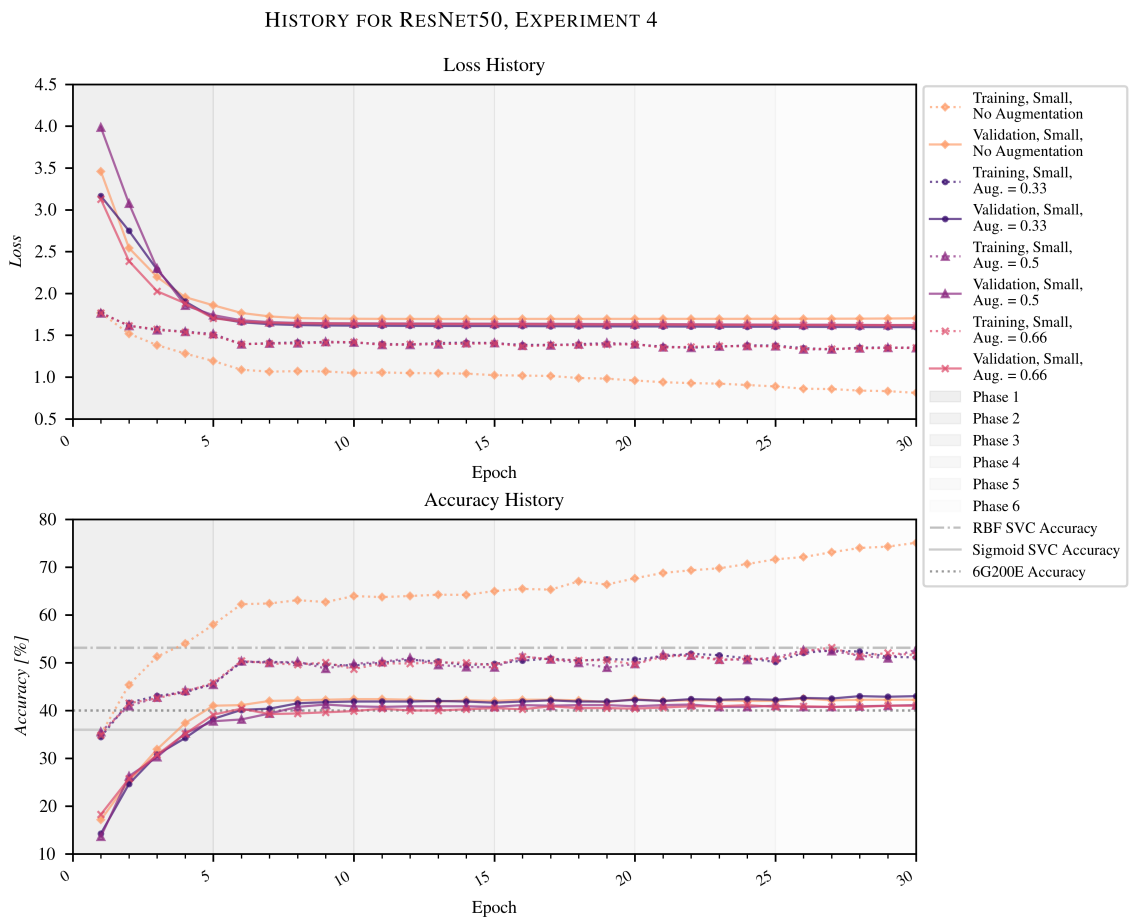


Figure 20: Loss and accuracy history for ResNet50, using the FMA small dataset, contrasting performance with no data augmentation and with data augmentation of 0.33, 0.5, and 0.66 (up to 33%, 50%, and 66% horizontal shift, respectively). As in Experiment 3, the first phase trained the classifier alone. Then layers 161 and above, 151 and above, 139 and above, 129 and above, and 119 and above were trained for Phases 2-6 respectively. As for earlier models (Figures 17, 18, and 19), ResNet50 demonstrates less overfitting with data augmentation than without data augmentation. However, even with data augmentation, the small dataset-characteristic gap in performance between training and validation data can still be observed, suggesting early overfitting.

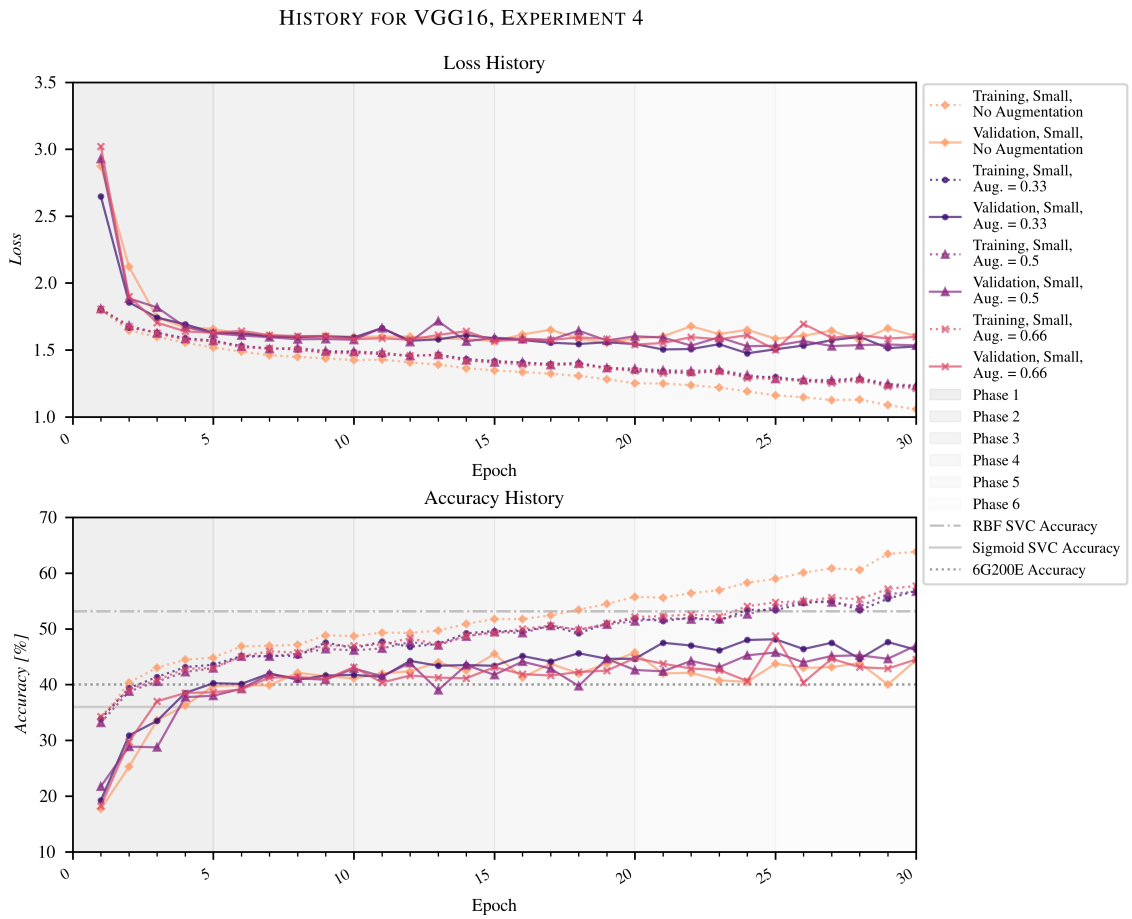


Figure 21: Loss and accuracy history for VGG16, using the FMA small dataset, contrasting performance with no data augmentation and with data augmentation of 0.33, 0.5, and 0.66 (up to 33%, 50%, and 66% horizontal shift, respectively). As in Experiment 3, the first phase trained the classifier alone. Then layers 17 and above, 15 and above, 13 and above, 11 and above, and 8 and above were trained for Phases 2-6 respectively. As for earlier models (Figures 17, 18, 19, and 20), VGG16 demonstrates less overfitting with data augmentation than without data augmentation. However, even with data augmentation, the small dataset-characteristic gap in performance between training and validation data can still be observed, suggesting early overfitting.

Model	Data Augmentation Factor	Final Validation Accuracy	Final Test Set Accuracy
Two-layer network	0.00	32.0%	30.1%
	0.33	43.4%	39.4%
	0.50	44.0%	38.9%
	0.66	41.5%	39.1%
Inception V3	0.00	32.1%	29.9%
	0.33	37.5%	32.1%
	0.50	36.5%	32.4%
	0.66	36.9%	31.9%
Xception	0.00	36.3%	33.8%
	0.33	39.7%	36.0%
	0.50	39.0%	34.8%
	0.66	40.0%	35.4%
ResNet50	0.00	42.3%	38.0%
	0.33	43.0%	39.3%
	0.50	41.1%	38.6%
	0.66	41.0%	40.5%
VGG16	0.00	44.4%	39.1%
	0.33	46.3%	39.0%
	0.50	47.0%	38.8%
	0.66	44.5%	34.8%

Table 4: Statistics for all models trained for 30 epochs on the FMA small dataset, with data augmentation factors of 0, 0.33, 0.5, or 0.66 (no data augmentation, or data augmentation of up to 33%, up to 50%, or up to 66% horizontal shift, respectively). Both the final validation accuracy (at Epoch 30) and the final accuracy of the trained model on the test set are provided. While all models except for ResNet50 showed consistent improvement with the introduction of data augmentation on the validation set, VGG16 failed to show reflect this improvement on the test set. For the other models, the improvement due to data augmentation was most significant for the two-layer network, which showed a 9.3 percentage point jump in test set accuracy between no data augmentation and a data augmentation factor of 0.33.

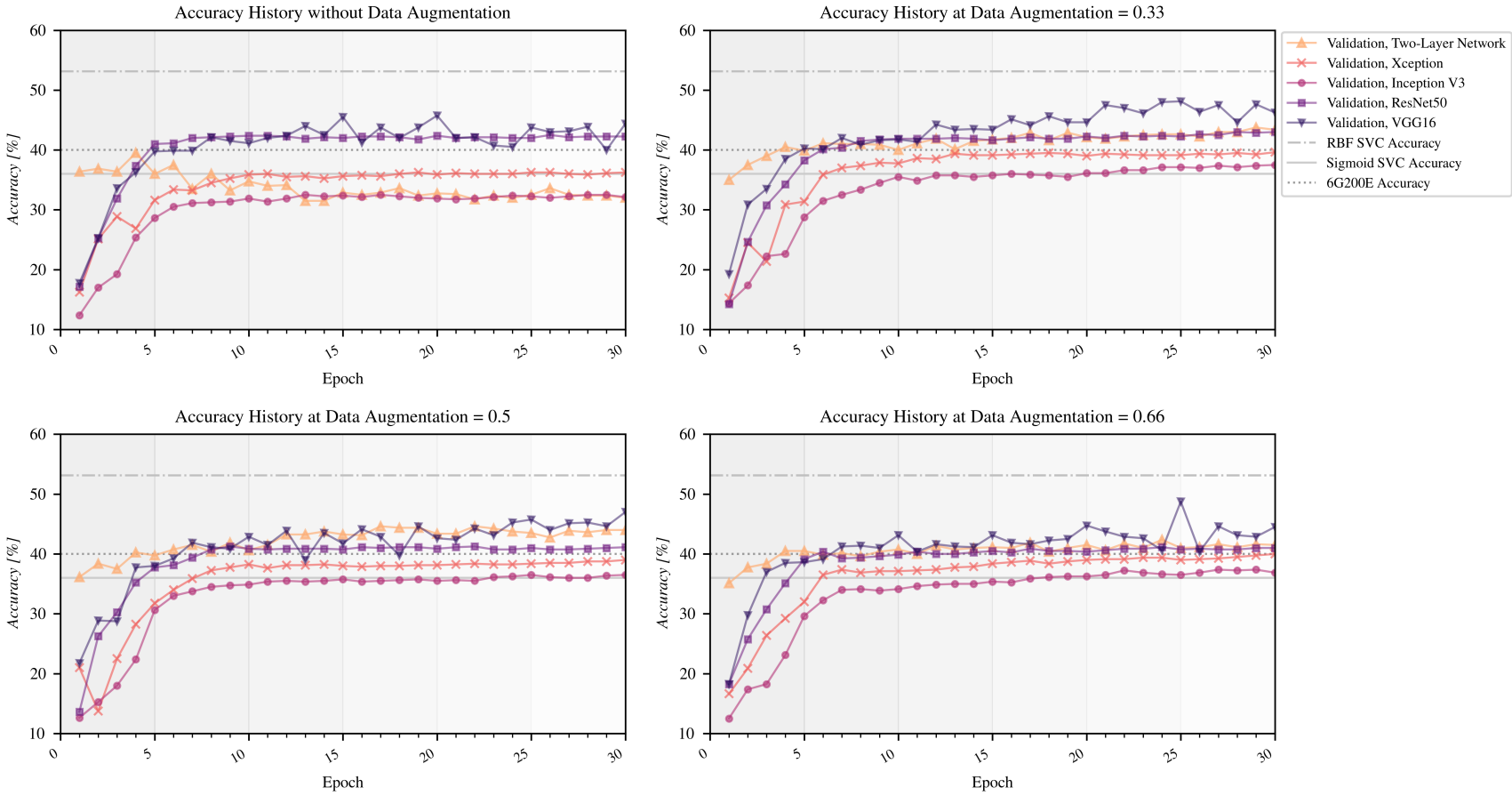


Figure 22: A comparison of the validation loss and accuracy for all five models, run on the small FMA dataset with no data augmentation (top left) and with data augmentation of 0.33 (top right), 0.5 (bottom left), and 0.66 (bottom right), or up to 33%, 50%, and 66% horizontal shift, respectively. Note that the results without data augmentation are the results from Experiment 3, and can also be viewed in Figure 16. They are repeated here for ease of comparison.

The poor performance of Xception and Inception V3 may be due to the fact that, as previously mentioned (“A Caveat Regarding Depthwise Separability,” p. 34), Inception architectures such as Inception V3 and Inception-like architectures such as Xception rely on partially or completely depthwise-separable convolutions for their optimization [79, 80, 81]. Again, while this assumption makes implicit sense when handling native image data, it may be less valid for the artificial, colorized wavelet data used in this project.

As in Experiment 3 (Table 3, Figure 16), the final *validation* accuracy of the VGG16 was the highest of the five models. However, with the addition of data augmentation, improvements in the *validation* accuracy for VGG16 were not reflected in the *test* accuracy. The ResNet50 and two-layer network both outperformed VGG16 on the test set, with test accuracies near 40%. This accuracy value again falls in between the test accuracies for the polynomial SVC model (39%) and the linear SVC model (42%), as in Experiment 3. No neural network model achieved a value comparable to the 46% test accuracy achieved by the RBF SVC model.

4.6. Experiment 5: Extended Dataset.

In Experiment 5, each of the five neural network models were trained for 30 epochs on the training split of the FMA extended dataset, after subtracting the mean and adjusting to unit variance, as in Experiments 3 and 4. This training was conducted first without data augmentation, then subsequently with a data augmentation factor of 0.33, triggering the Keras `ImageDataGenerator` module to augment the data by shifting the wavelet images horizontally by a random amount, up to 33% of the width of the image [63]. As in Experiment 4, the `'nearest' fill_mode` option was used when data augmentation was enabled.

As in Experiments 3 and 4, when training the two-layer neural network by itself, no changes were made during the 30 training epochs, and the best RMSprop op-

timizer found in Experiment 2 was used throughout. Also as in Experiments 3 and 4, for all the pre-trained models, training was divided into six five-epoch phases, using the best RMSprop optimizer found in Experiment 2 to train the classifier on the features extracted by the pre-trained model in phase 1, then an SGD optimizer with a low training rate for the remaining 5 phases, in which parts of the pre-trained model were successively unfrozen for updates.

Because Experiment 1 indicated that, as expected [55, 65], a higher validation accuracy overall could be achieved with the larger, FMA extended dataset than was achievable for the smaller, FMA small dataset, the SVC validation accuracy benchmarks used for Experiment 5 are these higher values. The 6G200E benchmark [18] remains unchanged.

The detailed training and validation history for Experiment 5 appears in Figure 23 (the two-layer network), Figure 24 (Inception V3), Figure 25 (Xception), Figure 26 (ResNet50), and Figure 27 (VGG16). A comparison of the validation accuracy throughout training for all five models appears in Figure 28. Because the validation accuracy fluctuates somewhat even late in training, both the final validation accuracy (at the end of Epoch 30) and the final-phase mean validation accuracy (for Epochs 26–30) appear in Table 5.

As previously mentioned ("Memory Management," 38), there are four times as many data points for results in Experiment 5, due to the necessity of checkpointing training mid-epoch.

Considering the final test set accuracy (Table 5), the two-layer network showed the greatest improvement with the introduction of data augmentation on the FMA small dataset, but the greatest improvement due to data augmentation on the FMA extended dataset was seen in the VGG16 model. However, as previously mentioned, the VGG16 model appears to be more susceptible to slight variations in training duration than the other models, so it is not known if this effect would persist after even one more epoch

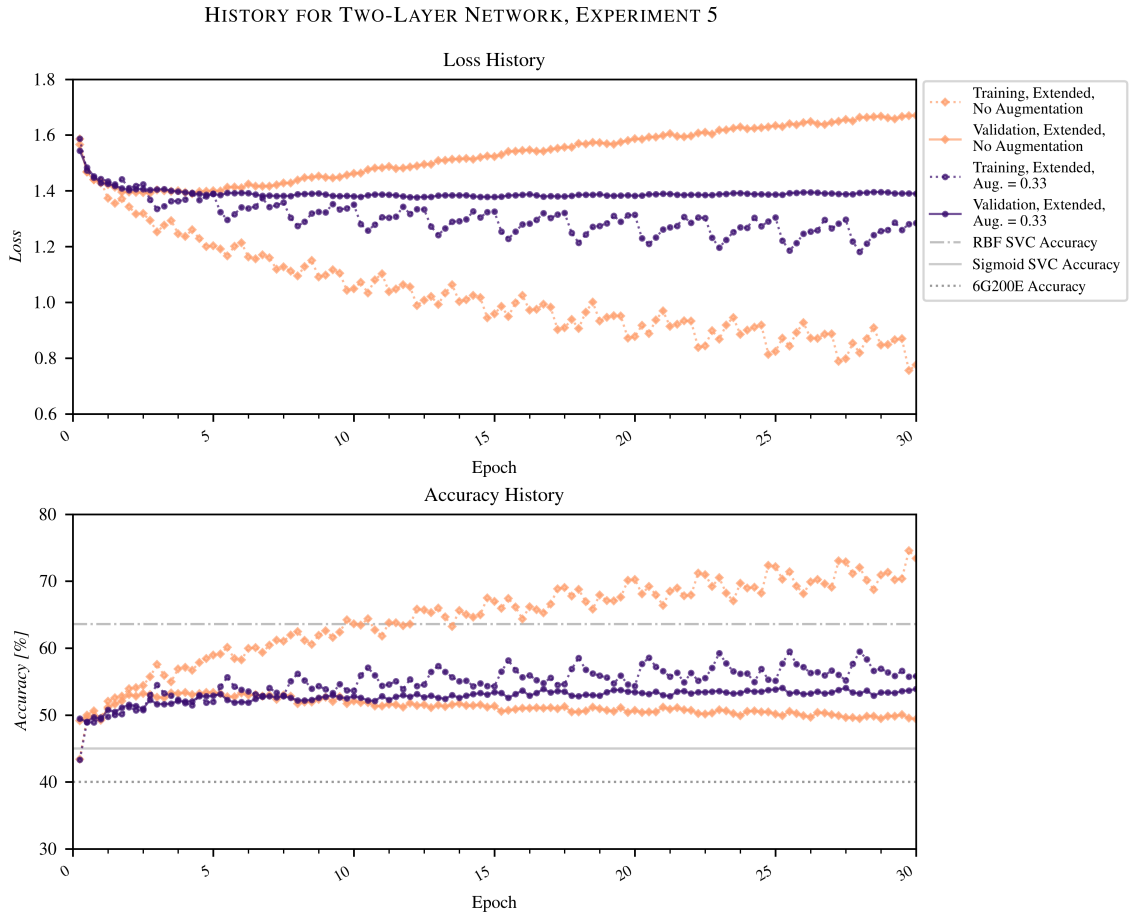


Figure 23: Loss and accuracy history for two-layer network (the classifier), using the FMA extended dataset, contrasting performance with no data augmentation and with data augmentation of 0.33 (up to 33% horizontal shift). As in Experiments 3 and 4, for the classifier training alone, no changes are made between the six training phases. As in Experiment 4 (Figure 17), adding data augmentation reduced the overfitting visible in the performance gap between the training and validation dataset splits. However, even without data augmentation, the two-layer network experienced much less overfitting than was seen on the small dataset in Experiment 3 (Figures 11 and 17).

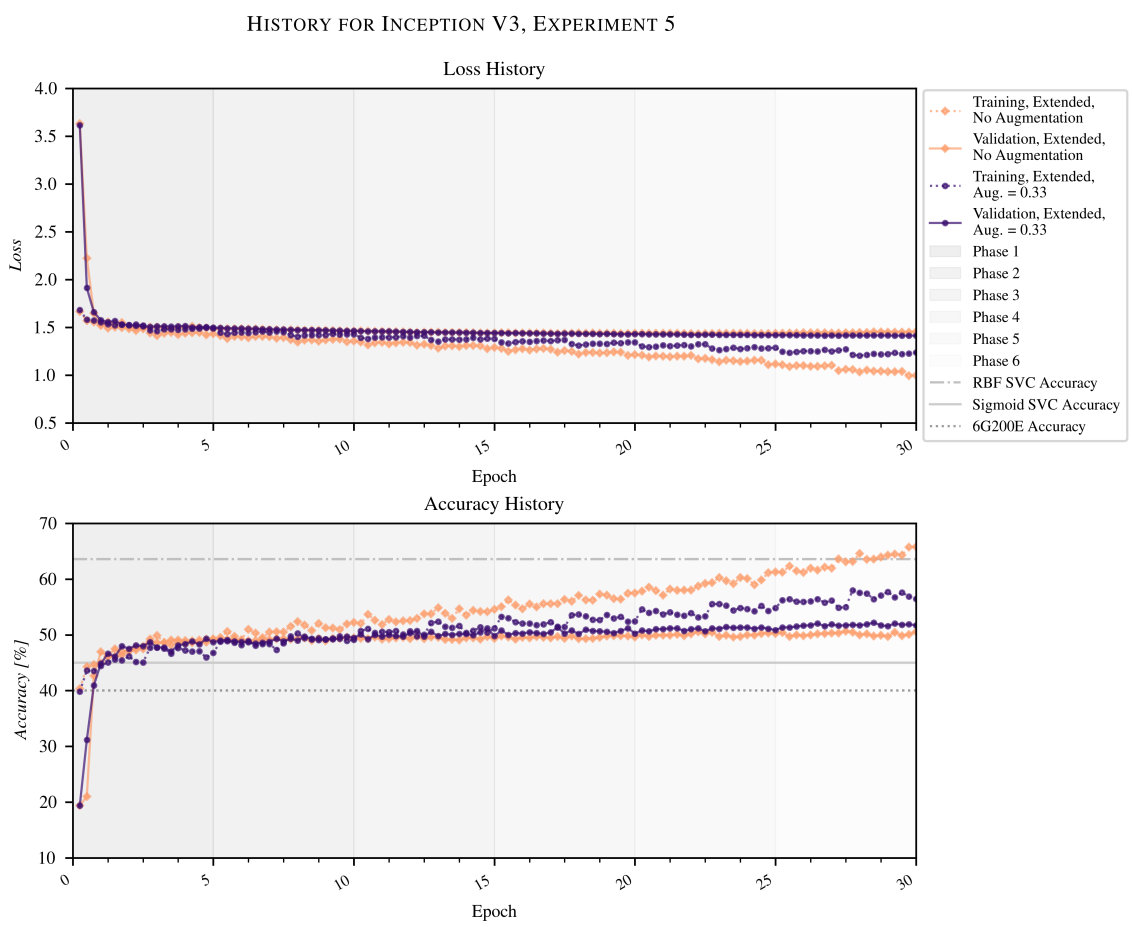


Figure 24: Loss and accuracy history for Inception V3, using the FMA extended dataset, contrasting performance with no data augmentation and with data augmentation of 0.33 (up to 33% horizontal shift). As in Experiments 3 and 4, the first phase trained the classifier alone. Then layers 249 and above, 232 and above, 229 and above, 200 and above, and 187 and above were trained for Phases 2-6 respectively. While adding augmentation did slightly improve the overall performance, this effect was less significant for the extended dataset than it was for the small dataset (Figure 18).

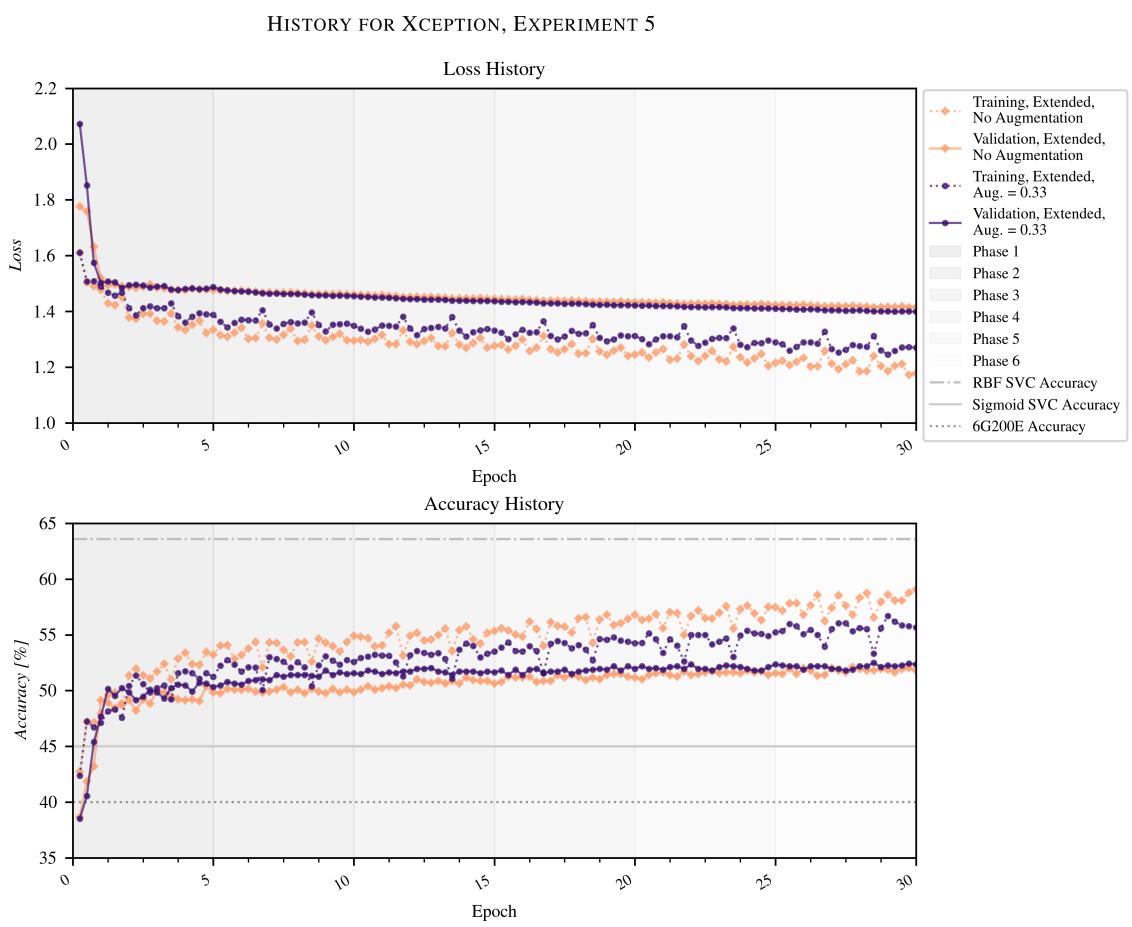


Figure 25: Loss and accuracy history for Xception, using the FMA extended dataset, contrasting performance with no data augmentation and with data augmentation of 0.33 (up to 33% horizontal shift). As in Experiments 3 and 4, the first phase trained the classifier alone. Then layers 122 and above, 105 and above, 95 and above, 85 and above, and 75 and above were trained for Phases 2-6 respectively. While adding augmentation did improve the overall performance, this effect was less significant for the extended dataset than it was for the small dataset (Figure 19).

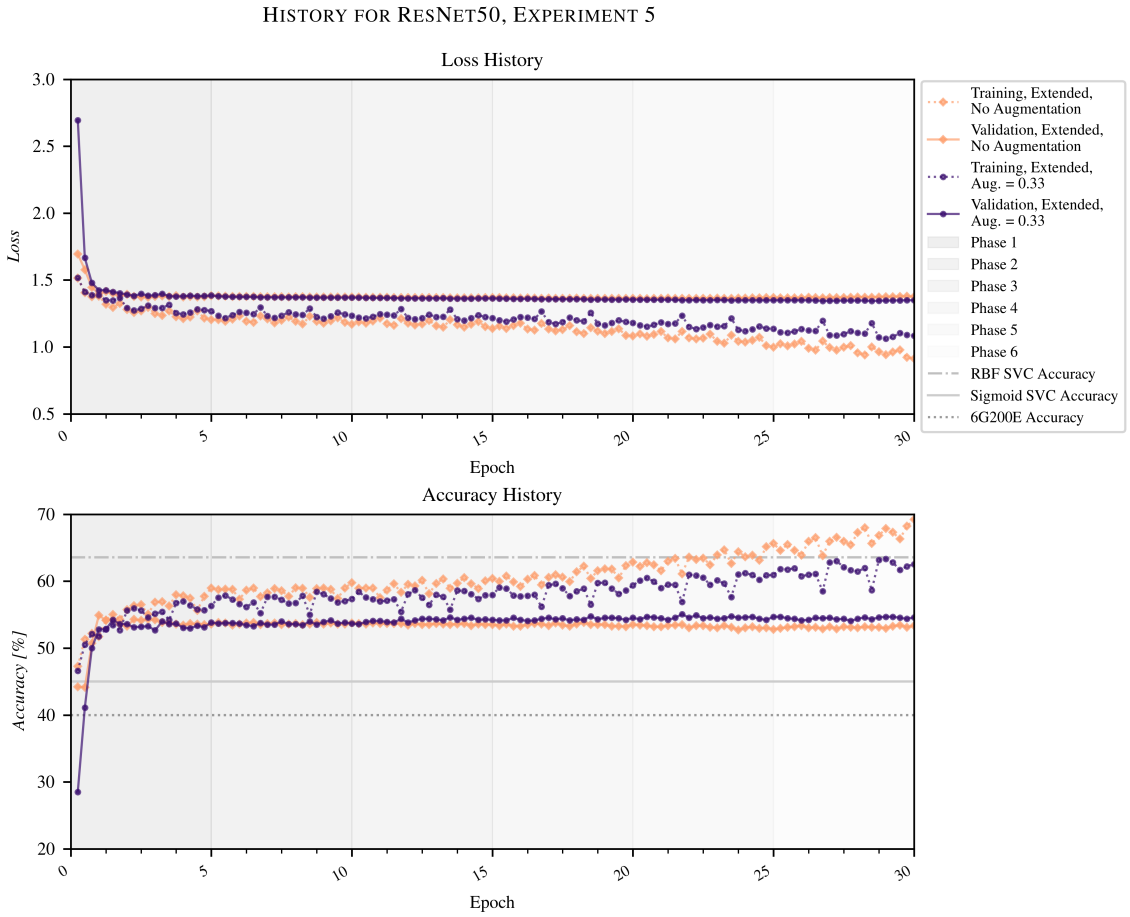


Figure 26: Loss and accuracy history for ResNet50, using the FMA extended dataset, contrasting performance with no data augmentation and with data augmentation of 0.33 (up to 33% horizontal shift). As in Experiments 3 and 4, the first phase trained the classifier alone. Then layers 161 and above, 151 and above, 139 and above, 129 and above, and 119 and above were trained for Phases 2-6 respectively. As was seen in Experiment 4 (Figure 20), the improvement from adding data augmentation was small for ResNet50.

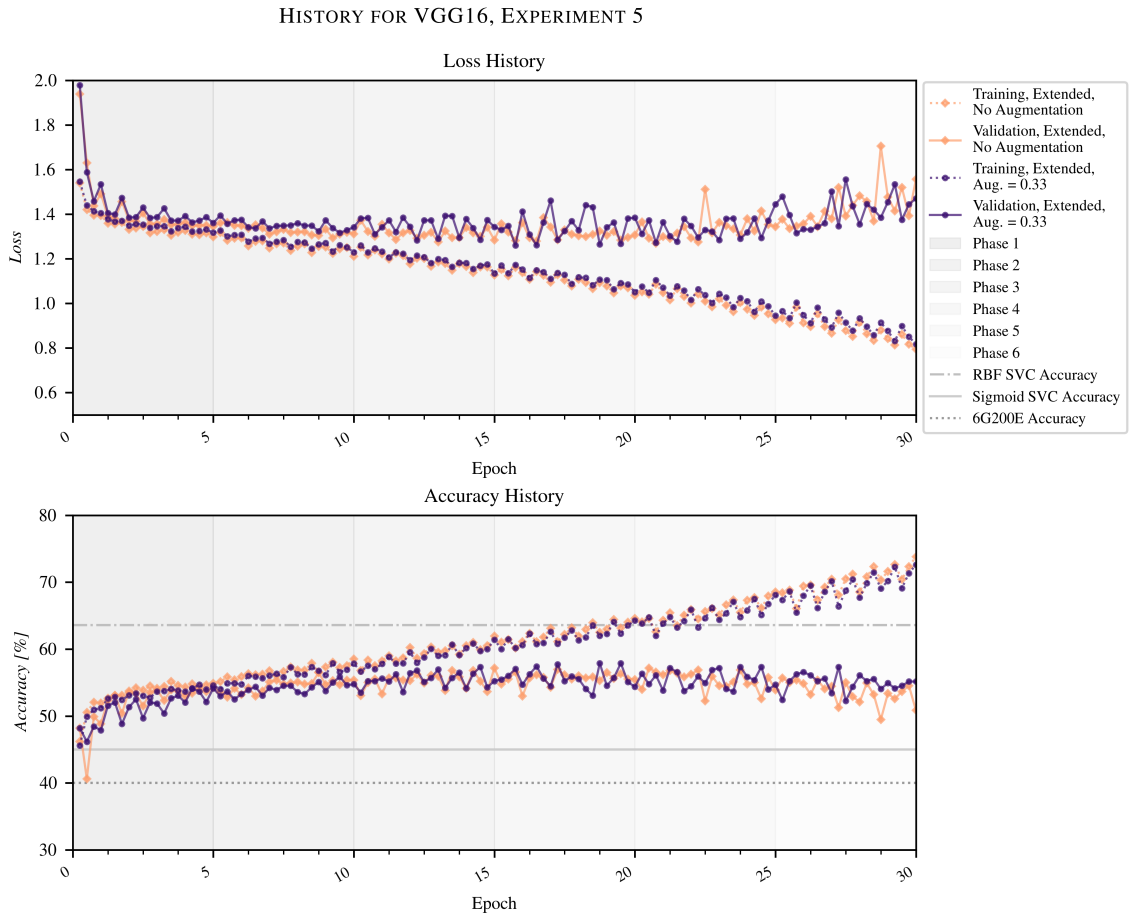


Figure 27: Loss and accuracy history for VGG16, using the FMA extended dataset, contrasting performance with no data augmentation and with data augmentation of 0.33 (up to 33% horizontal shift). As in Experiments 3 and 4, the first phase trained the classifier alone. Then layers 17 and above, 15 and above, 13 and above, 11 and above, and 8 and above were trained for Phases 2-6 respectively. While adding augmentation did slightly improve the overall performance, this effect was less significant for the extended dataset than it was for the small dataset (Figure 21). Note that VGG16's validation performance starts to deteriorate slightly after Epoch 15, both with and without data augmentation. This effect suggests early overfitting, even with data augmentation on the extended dataset.

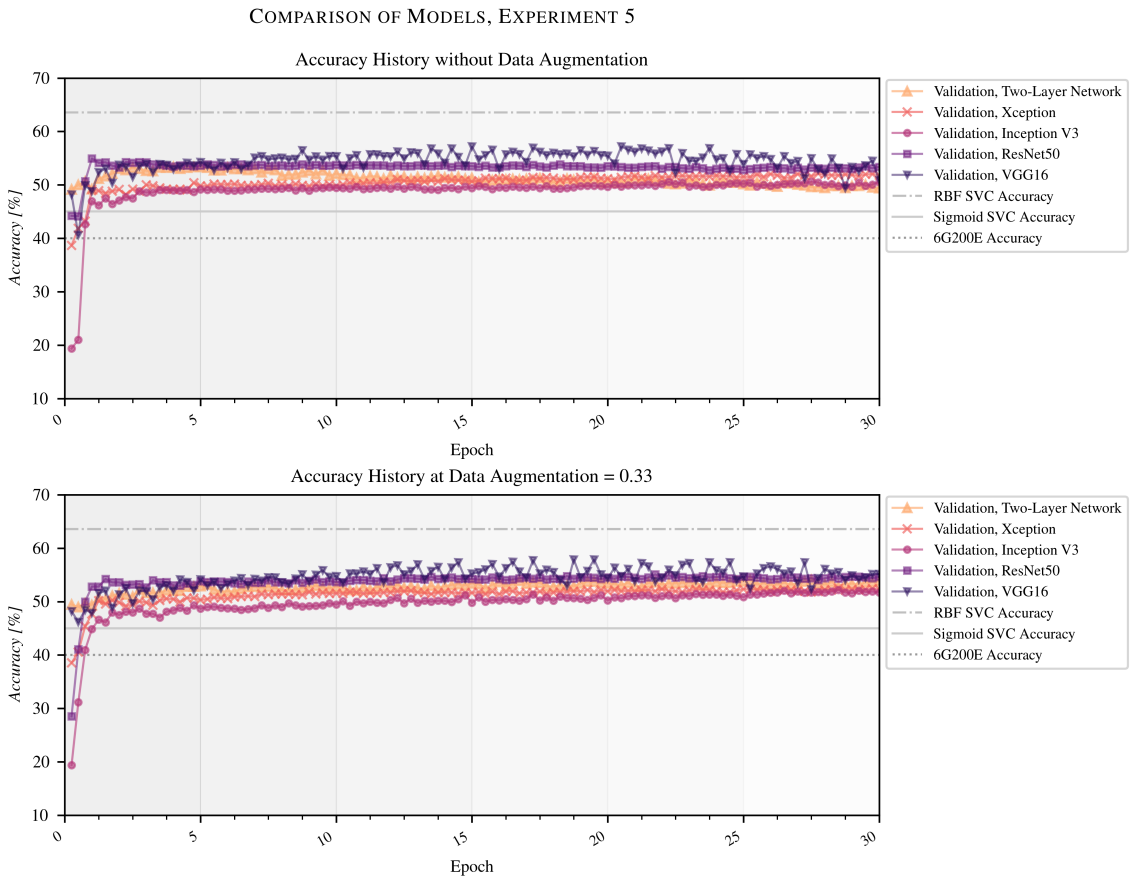


Figure 28: A comparison of the validation loss and accuracy for all five models, run on the extended FMA dataset with 0 and 33% data augmentation.

Model	Data Augmentation Factor	Final Validation Accuracy, Extended Dataset	Final Test Set Accuracy, Extended Dataset	Final Test Set Accuracy, Small Dataset
Two-layer network	0.00	49.4%	48.1	30.1%
	0.33	53.9%	51.4%	39.4%
Inception V3	0.00	50.5%	39.5%	29.9%
	0.33	51.7%	41.2%	32.1%
Xception	0.00	51.8%	43.0%	33.8%
	0.33	52.3%	40.6%	36.0%
ResNet50	0.00	53.4%	36.4%	38.0%
	0.33	54.6%	36.4%	39.3%
VGG16	0.00	50.9%	35.2%	39.1%
	0.33	55.2%	47.6%	39.0%

Table 5: Statistics for all models trained for 30 true epochs on the FMA extended dataset, with data augmentation factors of 0 or 0.33 (no data augmentation, or data augmentation of up to 33% horizontal shift, respectively). Both the final validation accuracy (at Epoch 30) and the final accuracy of the trained model on the test set are provided, and the final test accuracy for the small dataset is repeated from Table 4 for comparison purposes.

of training. Except for VGG16, for the extended dataset overall, the effect of data augmentation tended to be less significant than it was for the small dataset (see also Table 4 and Figure 22). For the Xception model, the addition of data augmentation in fact decreased final test set performance on the extended dataset, despite its validation accuracy showing a slight improvement.

As was seen in Experiment 4, both Xception and Inception V3 consistently underperformed the other models on the validation set. However, on the extended dataset, ResNet50's test accuracy dropped precipitously. This may be an effect of the extended dataset being unbalanced. Overall, for the extended dataset, the spread between models in terms of validation accuracies was smaller than it had been on the small dataset, but the gap between validation and test accuracies was larger.

The best overall result among neural networks was achieved with the two-layer network and a data augmentation factor of 0.33, which achieved a final test set accuracy of 51.4%. However, this result fails to match the test set performance of the three

best SVC models: the polynomial model (56%), the linear model (58%), and the RBF model (61%).

5. DISCUSSION

5.1. Model Comparisons.

While the SVC RBF model produced the best accuracy overall (Figure 28), with careful training, the neural network models performed competitively with mid-range SVC models, and outperformed the sigmoid SVC model. This suggests that, given additional refinements, the general approach of using convolutional neural networks designed for native image data on image representations of music may be feasible. However, given that the RBF SVC model was both faster to train and more accurate than all neural network models examined, of the strategies tested in this project, the RBF SVC remains the best extant solution for doing musical genre recognition. All models tested performed significantly better than chance.

Many techniques might offer potential improvements to the models used in this project. First, it is possible to use support vector classification as the classifier of a neural network. Just as, in Experiments 3–5, the two-layer network was used as the classifier for features extracted by way of the pre-trained models, an SVC model could be used on those features instead. If the weakness of the neural network models in fact lies in the classifier, using an SVC model might lead to improved results. This technique was not implemented as an additional experiment in this project due to time and cloud computing cost constraints, but would be a fruitful area for further investigation.

Second, much of the literature in machine learning and classification tasks uses ensemble models to achieve substantially improved results on those seen by individual networks operating alone [18, 48, 50, 75]. By combining the outputs of multiple models independently trained on the same data through a technique such as majority voting, it might be possible to achieve improved accuracy. This might improve the value of networks that underperformed after training independently, but more research would be required to confirm this hypothesis.

It should be noted that, among the pre-trained neural network models, VGG16 and ResNet50 performed particularly well on the small (balanced) dataset (Tables 3, 4, 5); and VGG16 performed particularly well on the extended dataset (Figures 16, 22, 28). These models therefore offer promising directions for further research, especially in the context of ensemble learning. Between these two models, ResNet50 performed only slightly worse as long as the dataset was balanced (typically on the order of 1 percentage point), but because it has about a fifth the number of parameters, it achieves those results at a significant reduction in resource cost (Table 2), and, attendantly, training time. The two-layer network was also able to produce good results with careful training, and in fact produced the best results on the extended (unbalanced) dataset. However, it was significantly more vulnerable to overfitting and small dataset size than were the more complex models (Figure 16).

5.2. Revisiting Depthwise Separable Convolutions.

In contrast to the moderately good performance of VGG16, ResNet50, and the two-layer network, the Inception V3 and Xception architectures, which both encode image-based assumptions about the input data into the network architecture, frequently failed to meet the performance achieved by the other models, including the 6G200E base benchmark established in [18] and the sigmoid SVC, the worst overall model (Tables 3, 4, 5; Figures 16, 22, 28).

This result underscores the potential pitfalls of adapting models designed for image processing to other domains. As was mentioned previously (“A Caveat Regarding Depthwise Separability”, p. 34), Inception V3 and Xception assume that correlation across color channels and correlation across location (width and height) are not related, and can be handled separately. This allows for optimizations in model architecture while maintaining high accuracy when processing native image data (Table 2). However, for the colorized wavelet data, where a certain amount of cross-channel

correlation is introduced by the colormapping process, the assumption central to these Inception-based models was no longer valid. It is therefore unsurprising that these models underperformed networks that do not encode a structural reliance on depthwise separability.

5.3. Resource Considerations.

As previously mentioned, the SVC models were both more accurate and faster to train than the neural network models. The neural network models also have high memory and computational demands, which necessitated the use of cloud computing GPU resources to evaluate efficiently.

However, the use of pre-trained neural network models offered a significant savings in terms of training time. By utilizing those pre-trained weights for the convolutional model along with a randomly initialized classifier, by the end of 30 epochs, even without data augmentation, most models were able to exceed the 200 epoch training accuracy benchmark which [18] had set on a simpler classification problem (Figure 16). When data augmentation was added or the dataset expanded, *all* models were able to exceed that benchmark, frequently within 5 or 10 epochs (Figures 22 and 28). While training these large networks was still slow, generally taking hours or days even with GPU resources, the pre-training significantly reduced what would have otherwise been an untenable training load.

5.4. Small versus Extended Datasets.

The large performance gaps between the FMA small dataset and the FMA extended dataset—typically on the order of 10–15 percentage points in terms of validation accuracy, given identical models and data augmentation settings (Figure 9 and Table 5)—suggest that the small size of available audio datasets remains a factor in the lower

accuracies achieved in the audio domain relative to those achieved in the image domain. In this regard, however, the FMA dataset [20] offers a significant improvement on many previously-available datasets, particularly if access to the original audio is required. This project also shows that data augmentation can be used on audio data to counter some small dataset effects, like the extreme overfitting seen in Figure 11.

However, the extended dataset also introduced significant variation between the validation and test dataset results. It is hypothesized that this is an effect of the extended dataset being unbalanced by class, though further investigation would be needed to confirm this hypothesis. It should be noted here again that methods for combating these effects, such as data weighting [65], exist but were beyond the scope of this project. The gap between validation and test accuracies for the extended dataset does, however, underscore the difficulty of working with unbalanced data and remains a promising area for further research.

5.5. MFCC versus Wavelet Features.

The best SVC results (obtained using MFCC features) consistently exceeded the best neural network results (obtained using wavelet features) across all experiments in this project. However, while multiple SVC models were tested on a single feature set, and multiple neural networks were tested on a single feature set, it was not possible to apply multiple feature sets to both two model types, due to the differing input data dimensionality requirements of SVC models and neural networks ("Features," p. 24). This means that the question of the relative merits of MFCC features and wavelet features remains unanswered. However, it was possible to exceed two sensical benchmarks (the 6G200E benchmark from [18] and the sigmoid SVC benchmark derived in Experiment 1) with the wavelet features, which implies that wavelet features do merit future study.

A possible approach would be to generate MFCC images similar to the wavelet images [20]. These MFCC images could be sized to avoid dimensionality problems, and used to train neural network models much as the wavelet images were in this project. Similarly, the wavelet data could be scaled down until loading it into the SVC models became feasible, though this would necessarily result in a loss of precision within the wavelet data. Additionally, as previously discussed ("Feature Extraction for Music," p. 5), many other approaches for using wavelets in audio-based machine learning have been previously investigated. Techniques such as the use of the DWT instead of the DFT in the MFCC calculation [27] offer potential methods for constructing a wavelet feature set that could be also used with the SVC models examined in this project.

5.6. Approaches to Data Augmentation.

An important consideration in analyzing the results of Experiments 4 and 5 is that the data augmentation used in this project was designed for native image data. While this project intended to investigate the adaptability of image techniques for image representations of audio data, it should be reiterated that the embedded assumptions of image-based data augmentation are not necessarily appropriate for audio.

While the horizontal shift augmentation used in this project has a meaning that is sensical in the audio domain—taking windowed, time-limited clips of the underlying musical track—the issue of how the windowed image was then returned to the required 256×256 size, via the `'nearest'` `fill_mode` setting in the data generator, remains a potential source of error. An exhaustive investigation into the possible settings for this option using multiple training runs and cross-validation might yield improved results. A still better approach might be to perform instead a form of data augmentation tailored to the underlying audio nature of the dataset. This augmentation could be done at the point of wavelet image generation, rather than adapting image-based data augmentation

techniques after the fact. This approach would result in fewer image-based distortions being introduced into the results.

For example, in this project, the best results for most models were seen for clips that were 20–30 seconds long (i.e., a data augmentation factor of 0.33). Since a constant input image size is required for input to the pre-trained models, and because altering the horizontal scale of output images would be equivalent to altering the underlying audio, it would be necessary to select a fixed clip length, to avoid scaling distortions. However, once such a length was selected (e.g. 20 seconds), the wavelet image generation code could select some number of random windows onto the wavelet decomposition data, each with that fixed (e.g. 20 second) length, and generate wavelet PNG files for each of those windows. Because the wavelet image generation heavily compresses the data in the time dimension while writing the wavelet data out to PNG format (i.e. the wavelet decomposition data is much wider than 256 elements on the images' time axis), this would result in more accurate (less time compressed) wavelet image representations, while simultaneously introducing an element of randomness similar to that introduced by the image-based data augmentation used in Experiments 4 and 5. This approach would, however, increase the storage space required for the dataset's wavelet features, by increasing the number of images used to represent it.

Because of the high audio sample rate, the number of unique potential 20-second clips that can be taken from a 30-second clip is large: the audio tracks in the FMA dataset are sampled at either 44.1 kHz or 48 kHz, meaning that any individual 30-second track in the dataset could in theory generate a minimum of 441,000 20-second audio clips that vary by at least one sample from one another. There would likely be a smaller number of unique $256 \times 256 \times 3$ image files generated from those audio files, because of the time compression that occurs during the wavelet PNG image generation process. However, while interpolation effects make it difficult to know precisely how many unique images could be generated for a given track, it must be at least 129. This

minimum can be illustrated by a simpler—though probably less diverse—data augmentation technique in which, instead of generating a 256×256×3 image from many 20-second windows, we generate a single 384×256×3 image from the 30-second clip, and then extract all possible 256×256×3 slices of that image.

This proposal assumes a relatively long (20 second) fixed window time. However, it should also be noted that the results of this project do not make it possible to distinguish between a fall in accuracy due to diminishing returns on an increased quantity of data augmentation, and a fall in accuracy due to increased negative effects from the image-based nature of the data augmentation. In other words, it is possible that the data augmentation factor of 0.33 yielded the best results simply because, at higher data augmentation rates, as shorter windows of the underlying audio were taken, the noise introduced by the `'nearest' fill_mode` option came to dominate any improvements that might've been seen from the increase in the diversity of the dataset. A data augmentation technique that avoided the use of that option, such as either of the methods proposed above, might be able to achieve good results on a shorter window length, which would lead to a larger number of augmented wavelet images. Again referring to the simpler technique (in which a wider image is generated, and then all possible 256×256×3 slices of that image are taken), if a 5-second window length was used, the full 30-second clip would be turned into a 1536×256×3 image, from which 1281 unique 256×256×3 images could be taken, each representing a different 5-second window. Further investigation would be required to determine whether or not 5-second clips were adequate for genre determination.

While more audio-native data augmentation approaches might not fully compensate for the small size of the dataset relative to the very large datasets used in the image domain, they would offer a more semantically meaningful approach for doing audio-based data augmentation, while still being able to leverage large networks that had been pre-trained for image recognition. This approach was not tested in this

project, however, due to time and cloud computing cost constraints, but it remains a promising area for future research.

6. CONCLUSION

This project sought to investigate a selection of common machine learning strategies (support vector classification, traditional neural networks, and convolutional neural networks) on two types of audio features (Mel-frequency cepstral coefficients and the discrete wavelet transform), using a recent, large-scale music dataset to perform genre classification.

All model and feature set combinations tested performed significantly better than chance (12.5%, for the 8-genre classification task used in all experiments). The overall best result was obtained with an SVC model using an RBF kernel, which attained a test accuracy of 61% on the FMA extended dataset and 46% on the FMA small dataset. No neural network model outperformed that result. However, of the neural network models, the two-layer network, VGG16, and ResNet50 performed particularly well. ResNet50 (as long as the dataset was balanced) and VGG16 consistently outperformed the benchmark set in [18], achieving accuracies near the middle of the range of those achieved via SVC models. VGG16 achieved 47.6% final test accuracy (using a data augmentation factor of 0.33) on the extended dataset and 39.1% on the small dataset (using no data augmentation). ResNet50, on the other hand, achieved a 40.5% final test accuracy (using a data augmentation factor of 0.66) on the small dataset, with a substantial decrease in training time and resource cost. However, ResNet50's performance fell significantly on the extended dataset, which is not balanced by genre.

With careful training, the two-layer network operating alone achieved better results than any other neural network model (51.4% on the extended dataset), but it also proved the most vulnerable of all the models to overfitting, which was particularly a factor in this project due to the FMA dataset's overall small size.

This project indicated that the general technique of adapting networks trained for image recognition may have some promise in the music domain. Significant training time savings were achieved by using publicly-available weights that had been pre-

trained on the ImageNet dataset. However, this project also demonstrated that techniques that rely on structural properties of image data (such as depthwise separability, as used in Inception-based models like Inception V3 and Xception) may not be adaptable to music-based tasks.

The relatively small size of the dataset relative to datasets in other domains and sub-optimal data augmentation techniques for the wavelet features are suggested as factors in the inability of the machine-learning models evaluated in this project to achieve the quality of results observed in the image domain. However, the FMA dataset used in this project appears to offer a significant improvement on previously available full-audio datasets, based on comparison to previously published benchmarks such as [18].

To improve upon these results, this project suggests several possible areas of future research. The first would be the use of support vector classification, which performed well on MFCC features, on visual features extracted by pre-trained image recognition networks. The second would be the use of ensemble models, in which the results of multiple models were aggregated in a systematic way, such as majority voting, to provide improved accuracy. The third would be the use of a data augmentation strategy that was less image-native, and better represented the nature of the underlying audio data.

While all three of these strategies offer potential improvements, the second and third are perhaps of the most interest. Ensemble models have been repeatedly shown to yield improved results over solo models, and it is known that the data augmentation strategy used in this project encoded certain image-based assumptions that are less applicable to music.

Additionally, further research into all possible feature-model combinations suggested in this experiment (i.e. SVC models with wavelet images, neural network with MFCC data) would allow for value judgements to be made about the relative merits of MFCC and wavelet features. However, because of the resource and dimensionality con-

cerns that prevented all combinations from being tested in this project, to conclusively state that one feature set was better than another would be unmerited at this time.

In addition to the above suggestions for potential avenues for further investigation, it is hoped that the results found in this project offer meaningful benchmarks to be used by other researchers.

REFERENCES

[1] Billboard. "IFPI global report 2017: Industry sees highest revenue growth in decades, but the YouTube issue remains" [Online]. Available: http://www.billboard.com/articles/business/7775019/ifpi-global-report-2017-music-industry-highest-revenue-growth-decades. Last checked: 2017 October 2

[2] Spotify. "About" [Online]. Available: https://press.spotify.com/us/about/. Last checked: 2017 October 2

[3] Apple. "iTunes" [Online]. Available: https://www.apple.com/itunes/music/. Last checked: 2017 October 2

[4] M. Welsh *et al.*, "Querying large collections of music for similarity," University of California, Berkeley, Tech. Rep., 1999.

[5] M. A. Casey *et al.*, "Content-based music information retrieval: Current directions and future challenges," *Proceedings of the IEEE*, vol. 96, no. 4, pp. 668–696, 2008.

[6] D. Pye, "Content-based methods for the management of digital music," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Istanbul, Turkey, 2000, pp. 2437–2440.

[7] J. Shen *et al.*, "Modeling concept dynamics for large scale music search," *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 455–464.

[8] T. Li and M. Ogihara, "Toward intelligent music information retrieval," *IEEE Transactions on Multimedia*, vol. 8, no. 3, pp. 564–574, 2006.

[9] E. Law *et al.*, "Evaluation of algorithms using games: The case of music tagging." *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, Kobe, Japan, 2009, pp. 387–392.

[10] J. T. Foote, "Content-based retrieval of music and audio," *Multimedia Storage and Archiving Systems II*, 1997, pp. 138–148.

[11] T. Li *et al.*, "A comparative study on content-based music genre classification," *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 282–289.

[12] T. Fritz *et al.*, "Universal recognition of three basic emotions in music," *Current Biology*, vol. 19, no. 7, pp. 573–576, 2009.

[13] Z. W. Ras and A. A. Wieczorkowska, *Advances in Music Information Retrieval*, Springer, 2010.

[14] S. McAdams *et al.*, "Perception and modeling of affective qualities of musical instrument sounds across pitch registers," *Frontiers in Psychology*, vol. 8, 2017.

[15] P. Lamere, "Social tagging and music information retrieval," *Journal of New Music Research*, vol. 37, no. 2, pp. 101–114, 2008.

[16] T. Bertin-Mahieux *et al.*, "The million song dataset," *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, Miami, FL, USA, 2011, pp. 591–596.

[17] G. Tzanetakis *et al.*, "Audio analysis using the discrete wavelet transform," *Proceedings of the WSES International Conference on Acoustics and Music: Theory and Applications (AMTA 2001)*, Skiathos, Greece, 2001.

[18] T. L. Li *et al.*, "Automatic musical pattern feature extraction using convolutional neural network," *Lecture Notes in Engineering and Computer Science*, vol. 2180, no. 1, pp. 546–550, Mar. 2010 [Online]. Available: http://www.iaeng.org/publication/IMECS2010/IMECS2010_pp546-550.pdf. Last checked: 2017 October 18

[19] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[20] K. Benzi *et al.*, "FMA: A dataset for music analysis," *CoRR*, vol. abs/1612.01840, Dec. 2016 [Online]. Available: http://arxiv.org/abs/1612.01840. Last checked: 2017 November 2

[21] B. Logan *et al.*, "Mel frequency cepstral coefficients for music modeling," Plymouth, MA, USA, 2000.

[22] T. D. Rossing *et al.*, *The Science of Sound*, 3rd ed., San Francisco: Addison Wesley, 2002.

[23] S. Imai, "Cepstral analysis synthesis on the Mel frequency scale," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Boston, MA, USA, 1983, pp. 93–96.

[24] R. Vergin *et al.*, "Generalized Mel frequency cepstral coefficients for large-vocabulary speaker-independent continuous-speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 5, pp. 525–532, 1999.

[25] T. Ganchev *et al.*, "Comparative evaluation of various MFCC implementations on the speaker verification task," *Proceedings of the 10th International Conference on Speech and Computer (SPECOM)*, Patras, Greece, 2005, pp. 191–194.

[26] L. Muda *et al.*, "Voice recognition algorithms using Mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques," Mar. 2010 [Online]. Available: http://arxiv.org/abs/1003.4083.

[27] C. Turner and A. Joseph, "A wavelet packet and Mel-frequency cepstral coefficients-based feature extraction method for speaker identification," *Procedia Computer Science*, vol. 61, pp. 416–421, 2015.

[28] D.-N. Jiang *et al.*, "Music type classification by spectral contrast feature," *IEEE International Conference on Multimedia and Expo*, Lausanne, Switzerland, 2002, pp. 113–116.

[29] S. Sigurdsson *et al.*, “Mel frequency cepstral coefficients: An evaluation of robustness of MP3 encoded music,” *Proceedings of the International Symposium on Music Information Retrieval (ISMIR)*, Victoria, BC, Canada, 2006.

[30] J. Portelo *et al.*, “Non-speech audio event detection,” *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 1973–1976.

[31] H. G. C. Sidney Burrus, Ramesh A. Gopinath, *Introduction to Wavelets and Wavelet Transforms: A Primer*, Prentice-Hall, 1998.

[32] H. L. Resnikoff and R. O. W. Jr., *Wavelet Analysis: The Scalable Structure of Information*., Springer, 1998.

[33] B. B. Hubbard, *The World According to Wavelets: The Story of a Mathematical Technique in the Making*, 2nd ed., Wellesley, MA: A K Peters, 1998.

[34] F. Wasilewski *et al.*, “PyWavelets: Wavelet transforms in Python,” 2010 [Online]. Available: http://pywavelets.readthedocs.io/. Last checked: 2017 October 27

[35] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[36] Pedregosa *et al.* (2017) "Support vector machines" [Online]. Available: http://scikit-learn.org/stable/modules/svm.html. Last checked: 2017 October 30

[37] C.-W. Hsu and C.-J. Lin, “A comparison of methods for multiclass support vector machines,” *IEEE transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[38] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, 2000.

[39] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015 [Online]. Available: http://neuralnetworksanddeeplearning.com/. Last checked: 2017 October 31

[40] D. Britz. "Deep learning glossary" [Online]. Available: http://www.wildml.com/deep-learning-glossary/. Last checked: 2017 November 1

[41] A. Karpathy *et al.* (2017) "Neural networks part 1: Setting up the architecture" [Online]. Available: http://cs231n.github.io/neural-networks-1/. Last checked: 2017 November 1

[42] S. v. d. Walt *et al.*, “The NumPy array: a structure for efficient numerical computation,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[43] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.

[44] A. Karpathy *et al.* (2017, Apr.) "Linear classification: Support vector machine, Softmax" [Online]. Available: http://cs231n.github.io/linear-classify/. Last checked: 2017 October 30

[45] ——. (2017, May) "Optimization: Stochastic gradient descent" [Online]. Available: http://cs231n.github.io/optimization-1/. Last checked: 2017 October 31

[46] ——. (2016, May) "Image classification: Data-driven approach, k-nearest neighbor, train/val/test splits" [Online]. Available: http://cs231n.github.io/classification/. Last checked: 2017 October 30

[47] ——. (2017, May) "Neural networks part 3: Learning and evaluation" [Online]. Available: http://cs231n.github.io/neural-networks-3/. Last checked: 2017 October 31

[48] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[49] A. Karpathy *et al.* (2017, May) "Neural networks part 2: Setting up the data and the loss" [Online]. Available: http://cs231n.github.io/neural-networks-2/. Last checked: 2017 November 1

[50] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *International Conference on Machine Learning*, Lille, France, 2015, pp. 448–456.

[51] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016 [Online]. Available: http://arxiv.org/abs/1609.04747. Last checked: 2017 November 2

[52] G. Hinton. (2014) "Neural networks for machine learning: Lecture 6a: Overview of mini-batch gradient descent" [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Last checked: 2017 August 31

[53] "What is the class of this image?" [Online]. Available: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html. Last checked: 2017 September 4

[54] A. Karpathy *et al.* (2017, May) "Convolutional neural networks: Architectures, convolution / pooling layers" [Online]. Available: http://cs231n.github.io/convolutional-networks/. Last checked: 2017 November 2

[55] I. Goodfellow *et al.*, *Deep Learning*, MIT Press, 2016 [Online]. Available: http://www.deeplearningbook.org. Last checked: 2017 October 28

[56] A. Karpathy *et al.* (2016, Nov.) "Transfer learning and fine-tuning convolutional neural networks" [Online]. Available: http://cs231n.github.io/transfer-learning/. Last checked: 2017 November 3

[57] F. Chollet *et al.* (2015) "Keras" [Online]. Available: https://github.com/fchollet/keras. Last checked: 2017 September 4

[58] Google. "Compute engine" [Online]. Available: https://cloud.google.com/compute/. Last checked: 2017 October 31

[59] A. S. Razavian *et al.*, "CNN features off-the-shelf: an astounding baseline for recognition," *CoRR*, vol. abs/1403.6382, 2014 [Online]. Available: http://arxiv.org/abs/1403.6382. Last checked: 2017 November 4

[60] Z. Zając. (2013, Oct.) "How much data is enough?" [Online]. Available: http://fastml.com/how-much-data-is-enough/. Last checked: 2017 October 30

[61] J. Leek. (2017, May) "Don't use deep learning your data isn't that big" [Online]. Available: https://simplystatistics.org/2017/05/31/deeplearning-vs-leekasso/. Last checked: 2017 October 18

[62] J. VanderPlas. (2015, Jul.) "The model complexity myth" [Online]. Available: https://jakevdp.github.io/blog/2015/07/06/model-complexity-myth/. Last checked: 2017 October 30

[63] F. Chollet. (2016, Jun.) "Building powerful image classification models using very little data" [Online]. Available: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html. Last checked: 2017 September 26

[64] A. L. Beam. (2017, Jun.) "You can probably use deep learning even if your data isn't that big" [Online]. Available: https://beamandrew.github.io/deeplearning/2017/06/04/deep_learning_works.html. Last checked: 2017 October 18

[65] N. V. Chawla *et al.*, "Special issue on learning from imbalanced data sets," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[66] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[67] S. van der Walt and N. Smith. "mpl colormaps" [Online]. Available: http://bids.github.io/colormap/. Last checked: 2017 October 27

[68] F. Chollet. (2016, Jan.) "How convolutional neural networks see the world: An exploration of convnet filters with Keras" [Online]. Available: https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html. Last checked: 2017 October 28

[69] Pedregosa *et al.* (2017) "Preprocessing data" [Online]. Available: http://scikit-learn.org/stable/modules/preprocessing.html. Last checked: 2017 October 30

[70] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[71] Pedregosa *et al.* (2017) "Kernel approximation" [Online]. Available: http://scikit-learn.org/stable/modules/kernel_approximation.html. Last checked: 2017 October 30

[72] A. Karpathy *et al.* (2017, May) "Putting it together: Minimal neural network case study" [Online]. Available: http://cs231n.github.io/neural-networks-case-study/. Last checked: 2017 October 28

[73] ——. (2017, Apr.) "Assignment 2: Fully-connected nets, batch normalization, dropout, convolutional nets" [Online]. Available: http://cs231n.github.io/assignments2017/assignment2/. Last checked: 2017 October 28

[74] F. Chollet *et al.* "Applications" [Online]. Available: https://keras.io/applications/. Last checked: 2017 October 4

[75] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014 [Online]. Available: http://arxiv.org/abs/1409.1556. Last checked: 2017 October 29

[76] J. Long *et al.*, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 3431–3440, Apr. 2017.

[77] L. Gatys *et al.*, "Texture synthesis using convolutional neural networks," *Proceedings of the 29th Annual Conference on Neural Information Processing Systems (NIPS)*, Montréal, QC, Canada, 2015, pp. 262–270.

[78] S. Ren *et al.*, "Object detection networks on convolutional feature maps," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 7, pp. 1476–1481, July 2017.

[79] C. Szegedy *et al.*, "Going deeper with convolutions," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

[80] ——, "Rethinking the Inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015 [Online]. Available: http://arxiv.org/abs/1512.00567. Last checked: 2017 September 18

[81] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *CoRR*, vol. abs/1610.02357, 2016 [Online]. Available: http://arxiv.org/abs/1610.02357. Last checked: 2017 September 21

[82] K. He *et al.*, "Deep residual learning for image recognition," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

[83] C. Szegedy *et al.*, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016 [Online]. Available: http://arxiv.org/abs/1602.07261. Last checked: 2017 October 29

[84] G. Tzanetakis and P. Cook, “GTZAN genre collection,” *Music Analysis, Retrieval and Synthesis for Audio Signals*, 2002.

[85] civilman628 *et al.* (2016, Oct.) "Resource exhausted error in the middle of training". Forum post [Online]. Available: https://github.com/tensorflow/tensorflow/issues/4735. Last checked: 2017 October 30

[86] tzachar *et al.* (2016, Mar.) "Memory leak when using tensorflow". Forum post [Online]. Available: https://github.com/fchollet/keras/issues/2102. Last checked: 2017 October 31

[87] F. Chollet *et al.* (2017) "Image preprocessing" [Online]. Available: https://keras.io/preprocessing/image/. Last checked: 2017 October 2017

[88] E. Jones *et al.*, “SciPy: Open source scientific tools for Python,” 2001– [Online]. Available: http://www.scipy.org/. Last checked: 2017 October 27

[89] F. Pérez and B. E. Granger, “IPython: a system for interactive scientific computing,” *Computing in Science & Engineering*, vol. 9, no. 3, 2007.

[90] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org [Online]. Available: https://www.tensorflow.org/. Last checked: 2017 October 31

APPENDIX A: MODEL STRUCTURES

In the diagrams in this appendix, layers are labeled as they appear in the instantiated Keras models. Layer numbers within a model indicate the order in which they are yielded by iteration in Keras and correspond to the layer numbers used when freezing and unfreezing layers for training. The trailing, type instantiation identification numbers have been dropped where they vary from instantiation to instantiation (e.g. "`add_10`" is listed simply as "`add`" if, on reinstantiation, the value "10" may be change).

General layer types are color-coded, with affine layers in yellow, activation layers in green, batch normalization layers in purple, dropout layers in blue-green, pooling layers in blue, convolutional layers in pink, and separable convolution layers in red (Xception only). Other layer types (e.g. `input`, `add`) appear in gray. Note that Keras refers to affine layers as "`dense`."

In all diagrams in this appendix, connection lines (in black) between layers flow from top to bottom, except for the arrows (in blue) which represent connections from column to column in models that are too large to be represented as a single column (Figures 30, 31, 32, and 33). Note that some layers have multiple inputs or outputs. In the interests of space, arrowheads are omitted wherever they are not necessary for visual clarity.

Due to their size and complexity, two of the model diagrams, Figure 30 for Inception V3 and Figure 32 for ResNet50, stretch across multiple pages. Red arrows and additional labeled blocks indicate where the model connections occur across page boundaries.

MODEL STRUCTURE: TWO-LAYER NETWORK

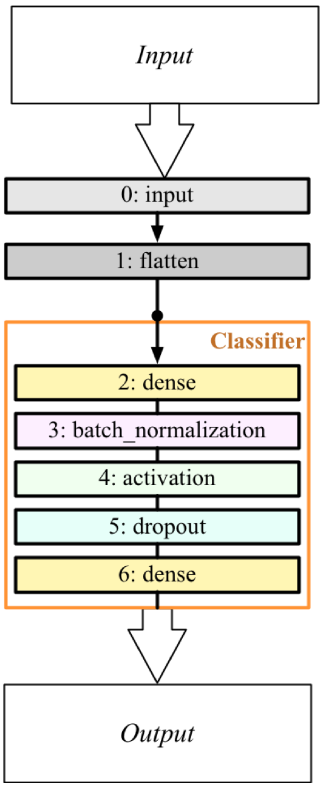


Figure 29: Layer structure of the two-layer network model. Layer 0 is an input layer customized for the 256×256×3 wavelet images used in this project. Layer 1 flattens that input into the shape expected by the first dense (affine) layer, i.e., a one-dimensional vector for each input example. Layers 2-6 form the classifier, also used at the output of the pre-trained models used in this project (Figures 30, 31, 32, and 33).

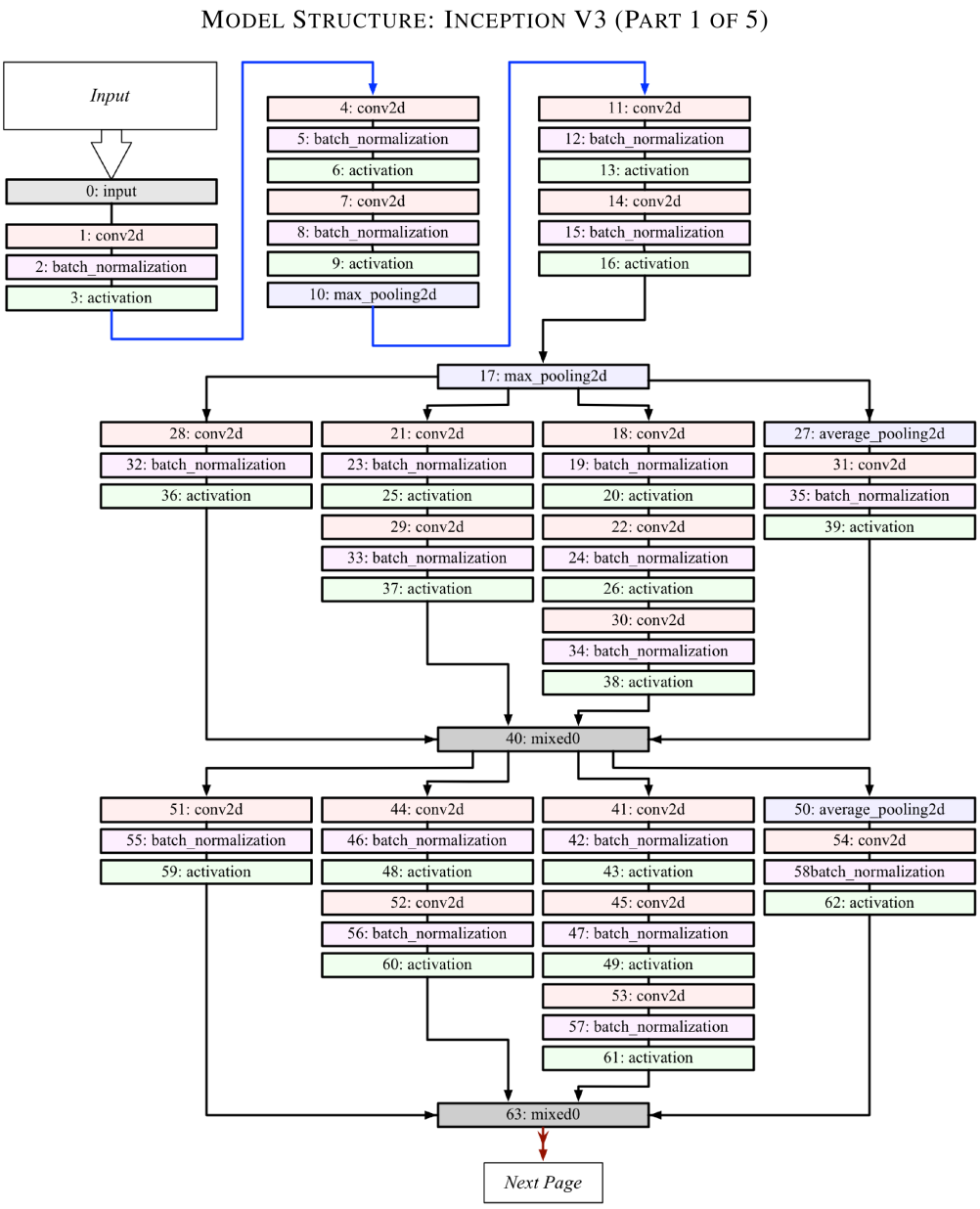
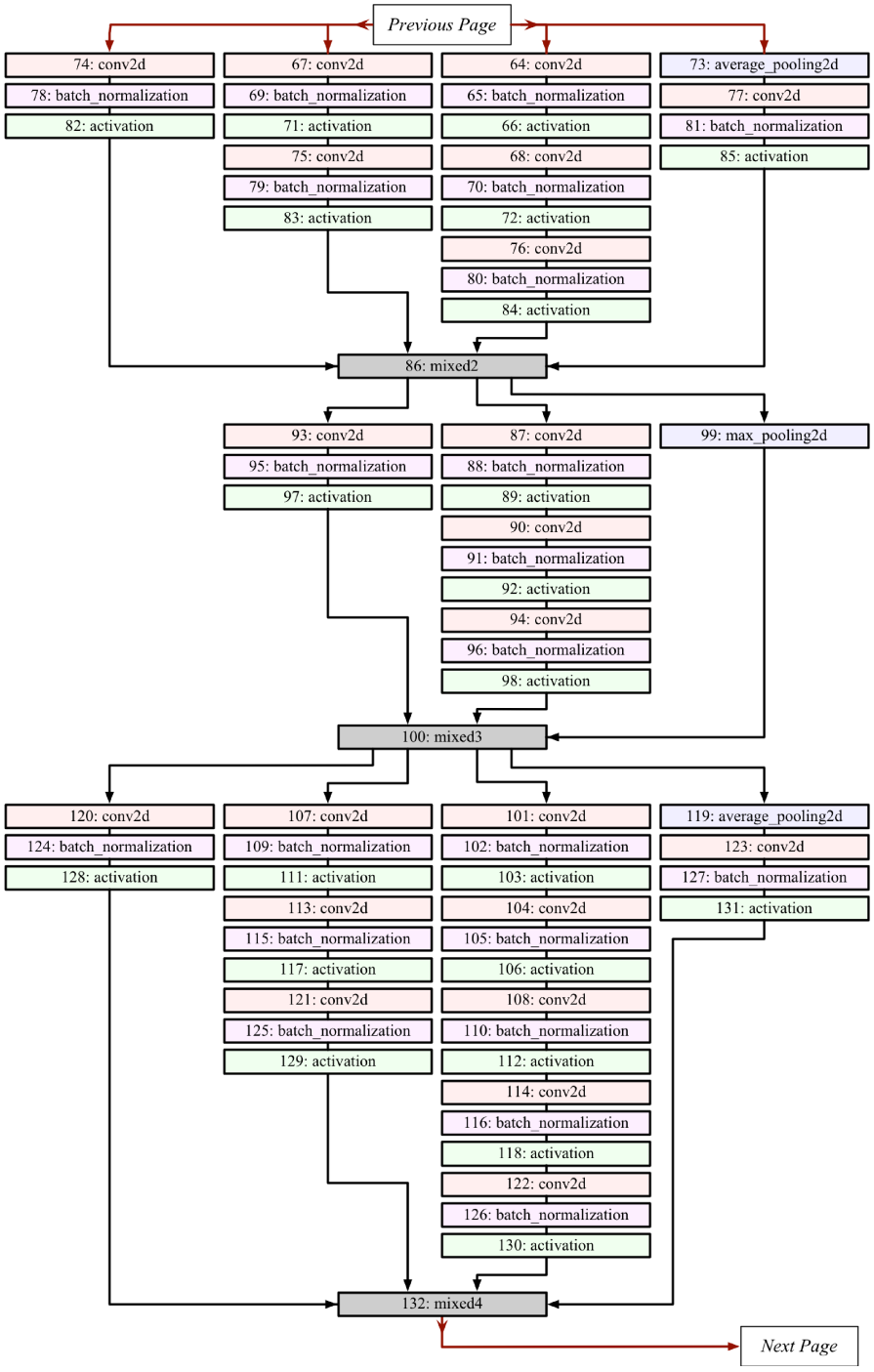
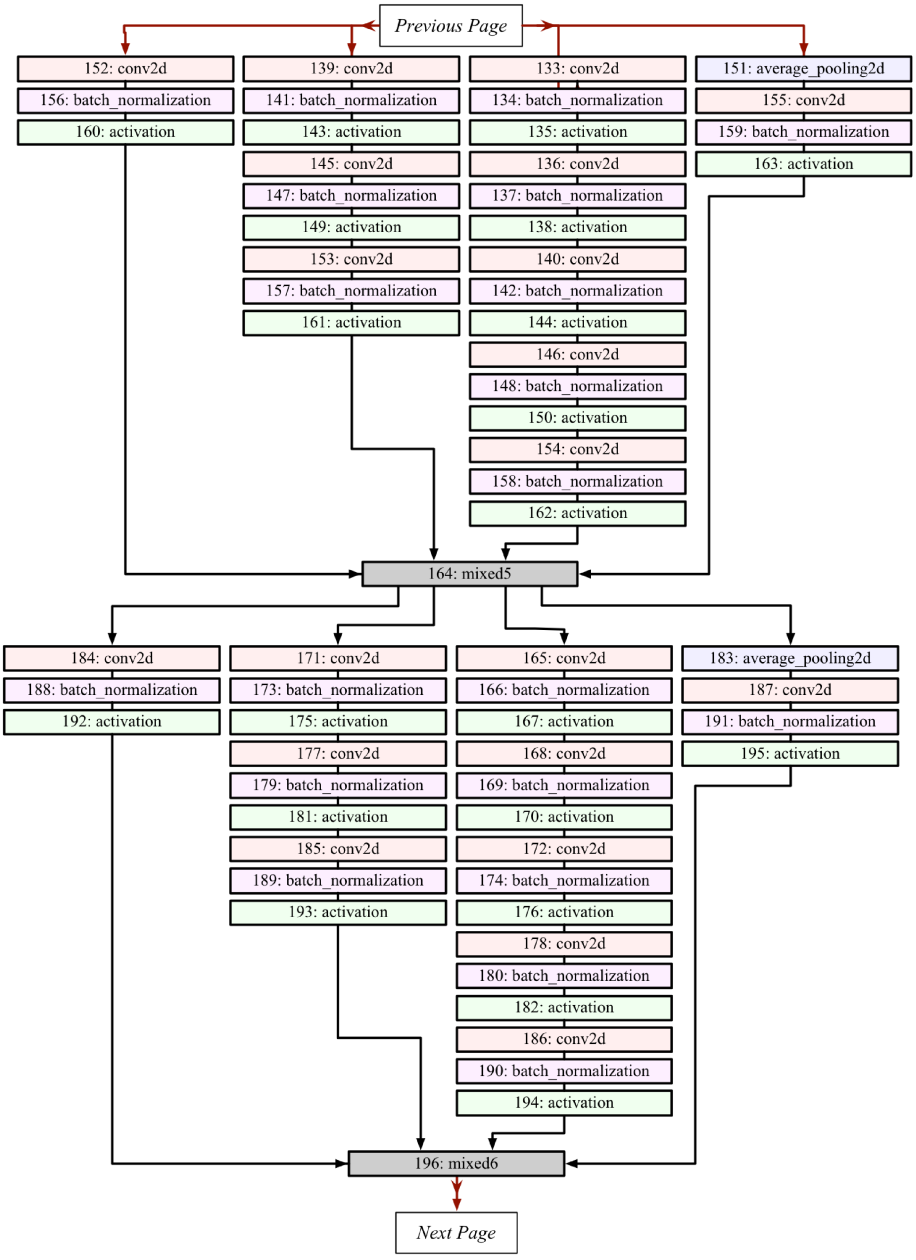


Figure 30: Layer structure of the Inception V3 [80] model. The layers termed "`mixed`" and "`concatenate`" both represent concatenation operations, in which outputs are stacked in the concatenation dimension, creating larger three-dimensional (volume) outputs [57, 80]. Layer 0 is an input layer customized for the 256×256×3 wavelet images used in this project. Layers 1-310 are the original Inception V3 model, for which weights pre-trained on ImageNet were loaded at initialization [74, 80]. Layer 311 is a global average pooling layer added after the example in [63]. Layers 312-316 are the classifier, identical to layers 2-6 of the two-layer network shown in Figure 29. The classifier was initialized randomly.

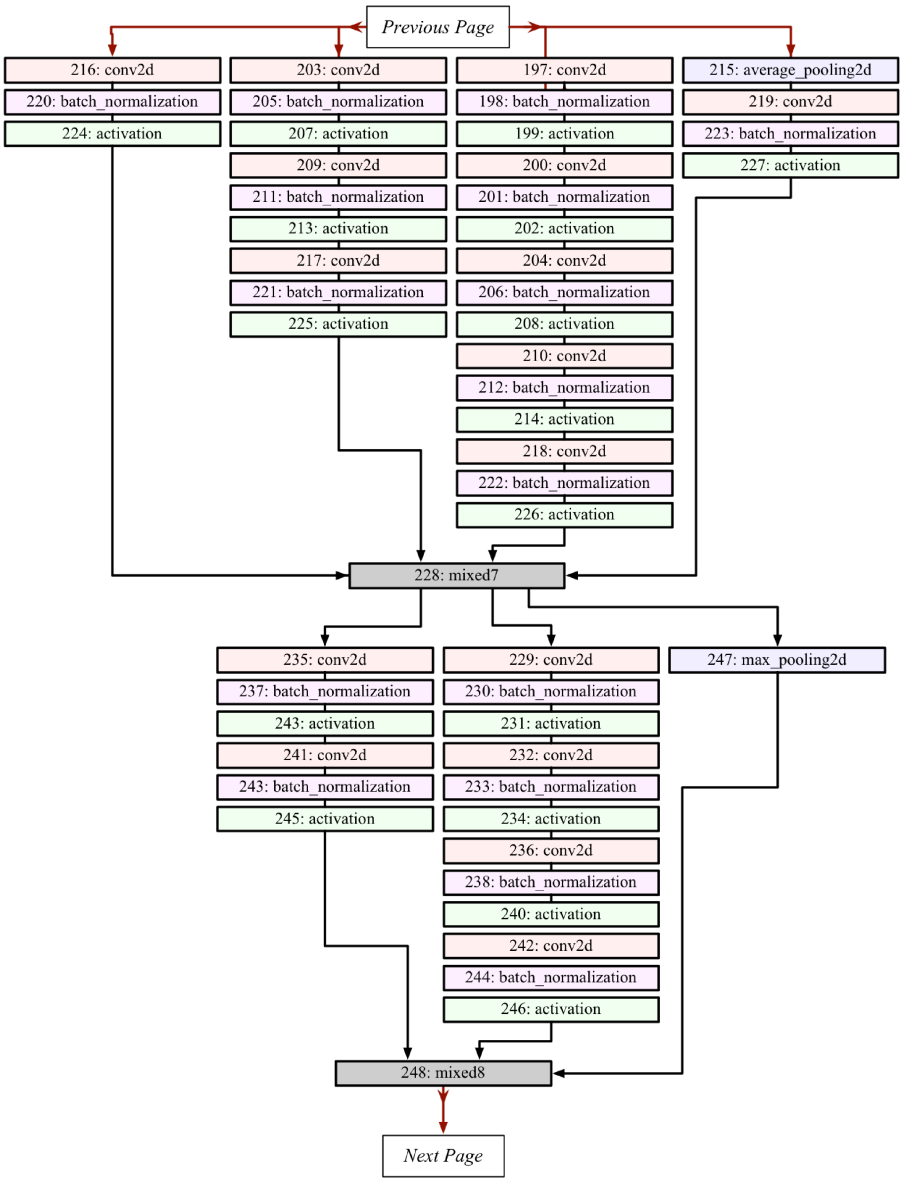
MODEL STRUCTURE: INCEPTION V3 (PART 2 OF 5)



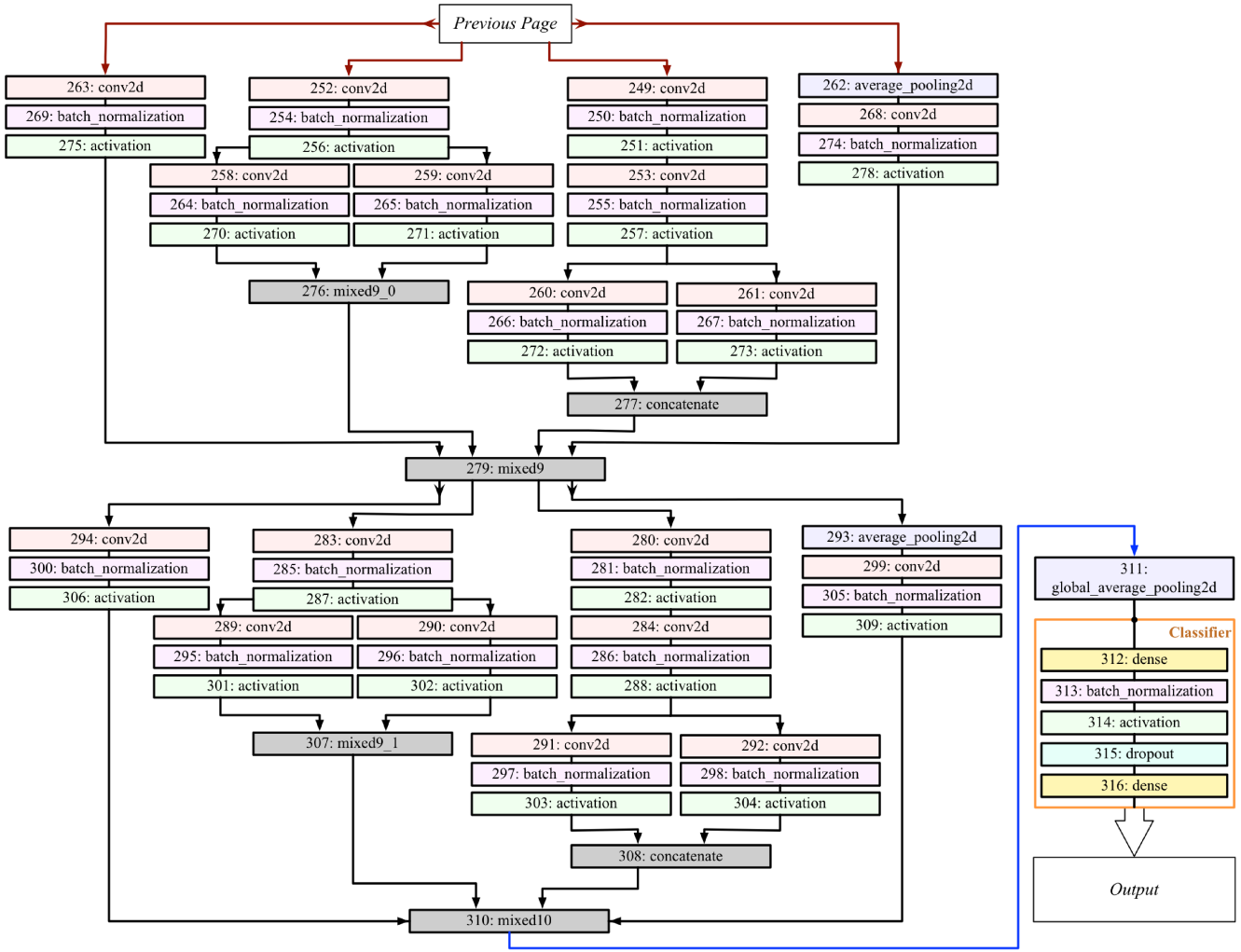
MODEL STRUCTURE: INCEPTION V3 (PART 3 OF 5)



MODEL STRUCTURE: INCEPTION V3 (PART 4 OF 5)



Model Structure: Inception V3 (Part 5 of 5)



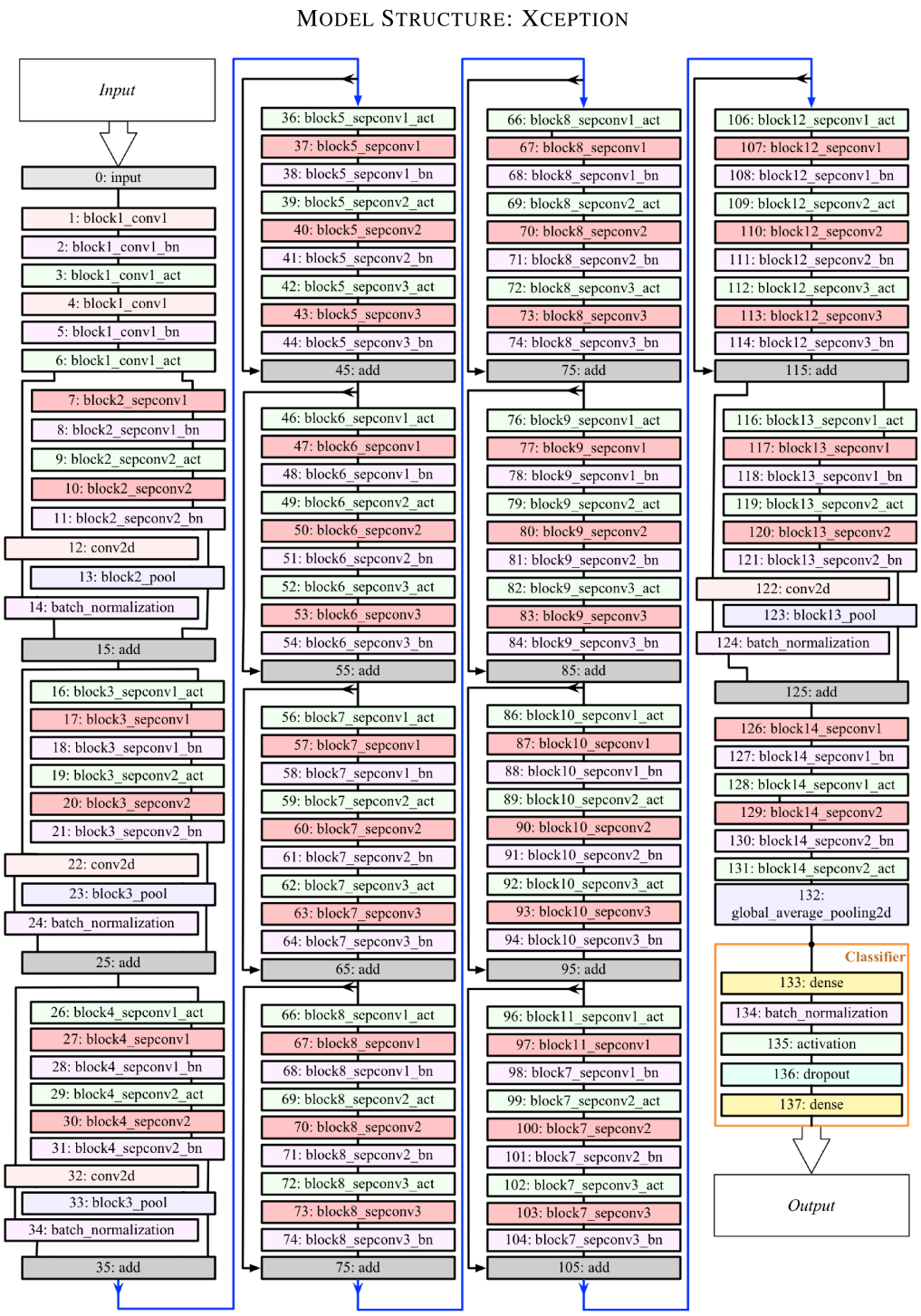


Figure 31: Layer structure of the Xception [81] model. Layer 0 is an input layer customized for the 256×256×3 wavelet images used in this project. Layers 1-131 are the original Xception model, for which weights pre-trained on ImageNet were loaded at initialization [74, 81]. Layer 132 is a global average pooling layer added after the example in [63], and a similar global pooling layer was used in the original Xception paper [81]. Layers 133-137 are the classifier, identical to layers 2-6 of the two-layer network shown in Figure 29. The classifier was initialized randomly.

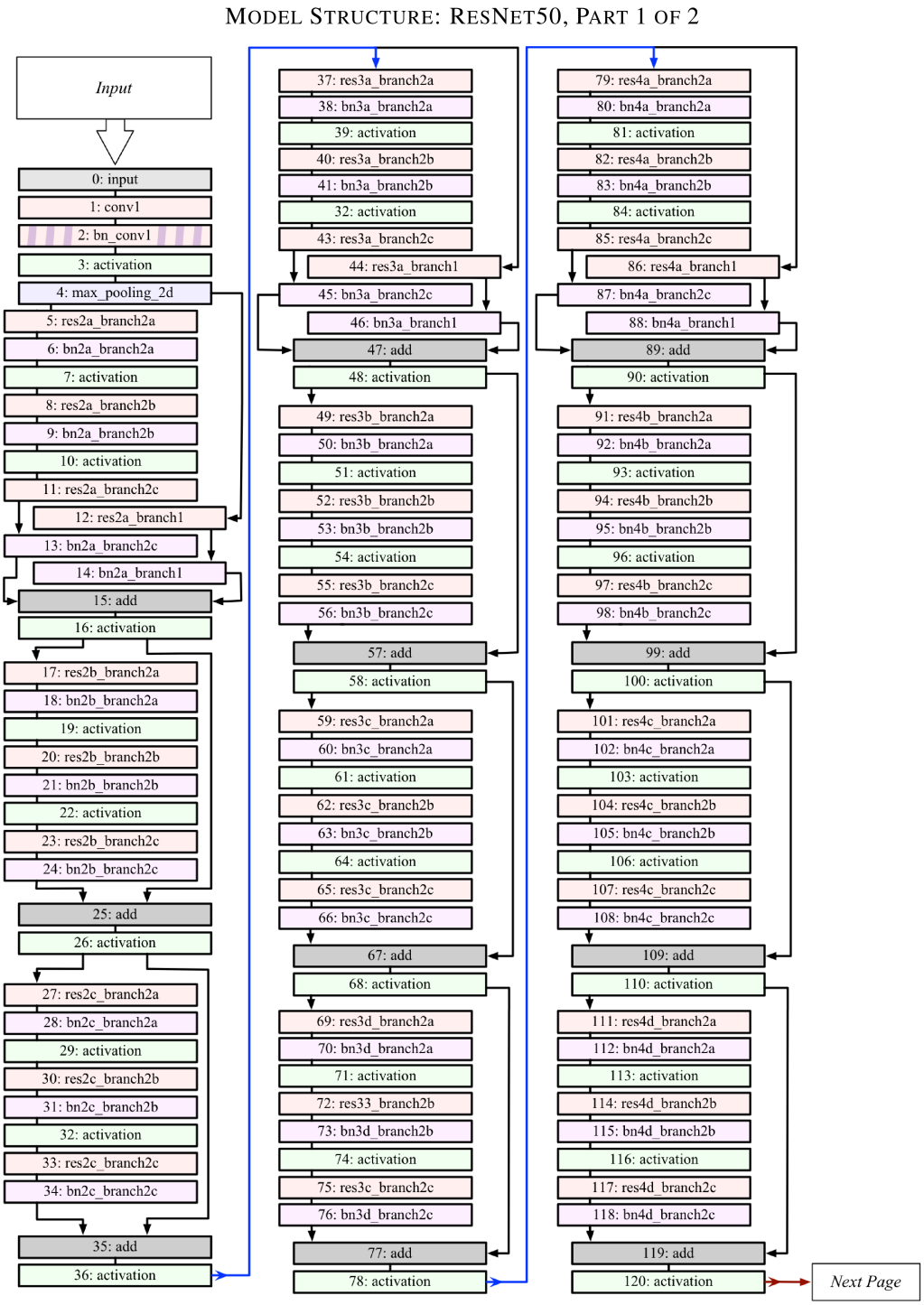
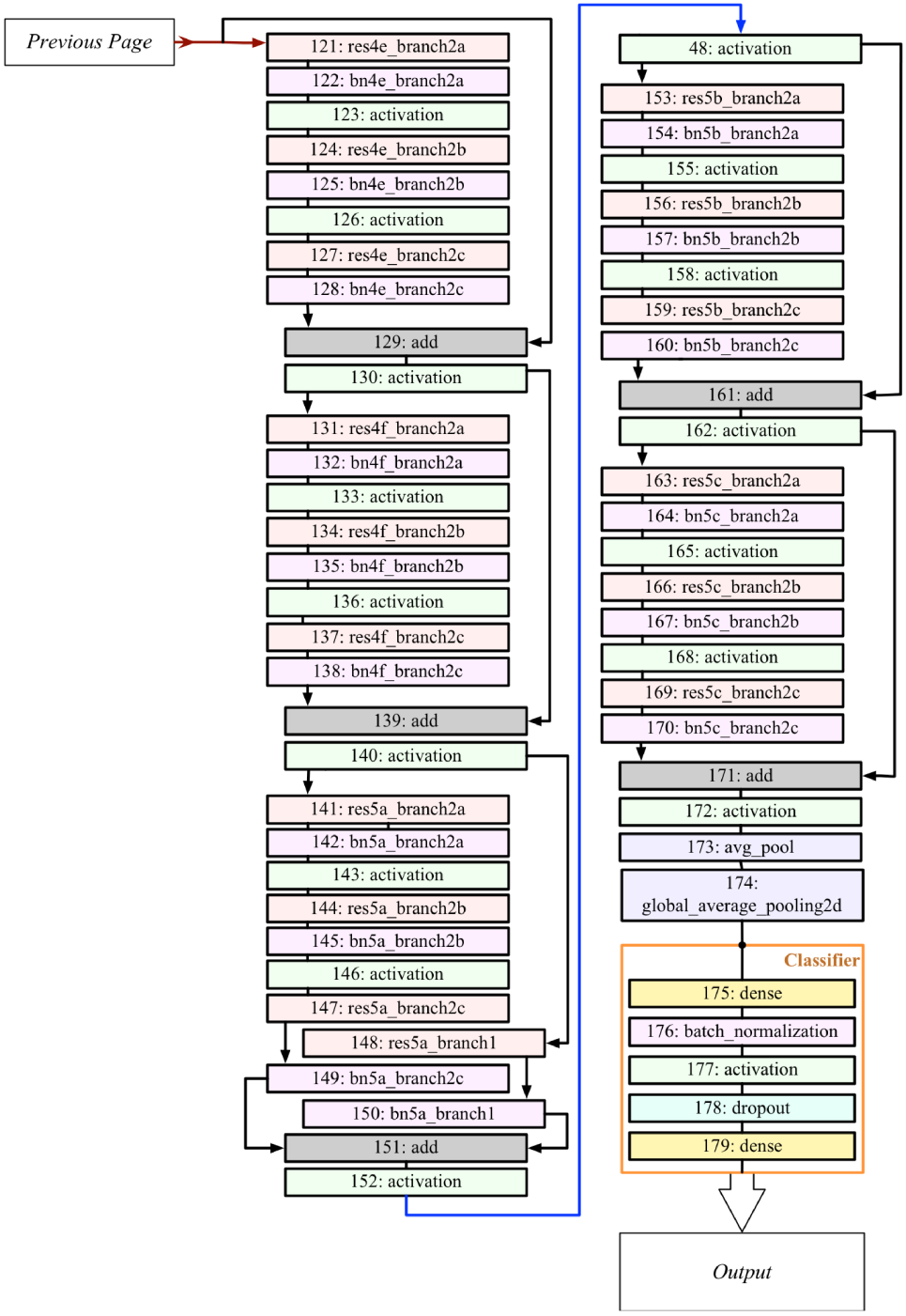


Figure 32: Layer structure of the ResNet50 [82] model. Note that in Layer 2, ResNet50 combines batch normalization ("bn") and convolution ("conv") into a single layer ("bn_conv"). Layer 0 is an input layer customized for the 256×256×3 wavelet images used in this project. Layers 1-173 are the original ResNet50 model, for which weights pre-trained on ImageNet were loaded at initialization [74, 82]. Layer 174 is a global average pooling layer added after the example in [63]. Layers 175-179 are the classifier, identical to layers 2-6 of the two-layer network shown in Figure 29. The classifier was initialized randomly.

MODEL STRUCTURE: RESNET50, PART 2 OF 2



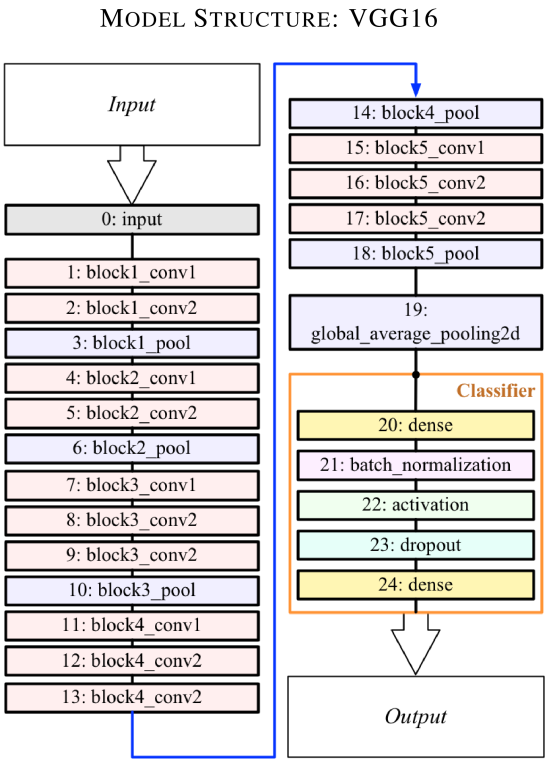


Figure 33: Layer structure of the VGG16 [75] model. Layer 0 is an input layer customized for the 256×256×3 wavelet images used in this project. Layers 1-18 are the original VGG16 model, for which weights pre-trained on ImageNet were loaded at initialization [74, 82]. Layer 19 is a global average pooling layer added after the example in [63]. Layers 20-24 are the classifier, identical to layers 2-6 of the two-layer network shown in Figure 29. The classifier was initialized randomly. Note that the VGG16 model predates (and therefore does not use) batch normalization within the pre-trained model [50, 75]. Also, VGG16 embeds ReLU activation into its convolutional layers [75], and therefore separate activation layers do not appear in this model.