Neural Networks for Audio: a Survey and Case Study

P. A. I. Forsyth
Department of Applied Mathematics
University of Waterloo
Waterloo, ON, N2L 3G1
pa2forsy@uwaterloo.ca

December 9, 2018

Abstract

After discussing the theoretical approximation capabilities of simple shallow neural networks, I survey recent work on the advantages of deep neural networks. I then overview some practical considerations for the training of deep convolutional neural networks. Finally, I describe my submission to the 2018 DCASE Freesound General-Purpose Audio Tagging Challenge, in which I trained deep convolutional neural networks to label the log spectrograms of audio clips. My submission ranked in the top 3%.

Contents

Chapter 1

Introduction

Artificial intelligence's early successes were in highly-structured problems, like chess[HGN90]. Less structured problems, like the recognition of images and sounds or the understanding of language, proved much more challenging. Although a human can easily decide whether an image is of a cat, determine whether a sound clip contains a cat's meow, or understand a sentence describing a cat, it is a tall order to explicitly specify the rules by which these judgments are made in a form usable by computers. Consequently, for a long time these problems appeared beyond the capabilities of AI.

Faced with such difficulties, some researchers abandoned the old axiom that "... one can explicitly, declaratively represent a body of knowledge in a way that renders it usable by ... programs."[LGP+90]. Instead of explicitly representing their knowledge about the problem, they programmed computers to implicitly acquire knowledge about the problem from a large number of examples of the problem being solved. Though this implicit knowledge was sometimes opaque to humans, it could often be used by the program solve previously unseen examples of the problem. This procedure became the basis of the vast field known as machine learning[1].

Machine learning frames the acquisition of implicit knowledge about a problem from examples as the approximation of an unknown function from observations of its values[2]. From this point of view, the essential question is what function approximators should be used. Machine learning offers myriad answers to this question, including— to name a few— linear and logistic regression; trees and ensembles thereof[Bre01, Fri01]; and density mixture models[FHT01, Chapter 8][3]. Recently, there has been a boom in interest in a powerful class of function approximators called neural networks.

Neural networks were studied in a nascent form in the middle of the twentieth century[MP43, WH60], were extensively developed in the 1980s[RHW85, LTHS88, LBD+89], and have recently achieved impressive performance on a wide range of tasks[KSH12, MKS+15, BCB14], largely as a consequence of the availability of GPUs, which enable fast multiplication of large dense matrices via tiling algorithms[KWM16, Chapters 4,16][4] Neural networks lie at the heart of an ongoing AI revolution, whose advances are being rapidly adopted by industry in such areas as machine translation[WSC+16], facial recognition[TYRW14], and the guidance of self-driving cars[HWT+15].

This essay is a survey of the theory and practice of machine learning with neural networks, including

[1]As mentioned above, early chess programming was primarily non-machine-learning AI. Even in this highly structured domain, however, the creators of the best chess programs eventually realized that there was much to be gained by the incorporation of machine learning techniques. *Deep Thought*, for example, automatically tuned the parameters of it's board-position evaluation function to minimize the difference between it's recommended moves and those played by grandmasters[HACN90, pg. 48]. In the modern AI era, the importance of machine learning to game-playing algorithms has only increased[SHM+16].

[2]Machine learning resembles statistics in that both fields approximate unknown functions using data. However, they differ in focus. Roughly speaking statisticians are more interested in understanding the function being approximated, while machine learning practitioners focus on the ability of the approximation to make predictions.

[3]Interestingly, in 2013 the standard approach to audio classification used Gaussian mixture models[SGB+15].

[4]The release of very large datasets[DDS+09, GEF+17, TSF+16], have also contributed to the success of neural networks.

a practical case study. It begins with Chapter 2, an overview of some classic and modern neural network theory. Chapter 2 is quite dense, and can be safely skimmed. A central theme of the chapter is that the key feature of neural networks is their compositional structure, and that this feature explains much of their power. In Sections 2.1 and 2.2, I introduce the computation graph framework, which is useful for describing neural networks and automatically differentiating them. In Section 2.3.1, I explore the expressive power of neural networks by establishing Proposition 4, a classic result that shows that even very simple neural networks can approximate a very large class of functions. In Section 2.3.2, I study the quality of this approximation as a function of the size of the network. I state and prove Proposition 6, an important result from the 1990s that shows that neural networks are at least as efficient approximators as the polynomials. In Section 2.4, I turn to more recent research on the power of increasing the degree of compositionality or "depth" of a neural network, and the advantages of deep networks over shallow ones. There are two strands of this research. The first strand shows that by various measures, the functions representable by deep neural networks are more complex and interesting that the ones representable by shallow networks. The second strand argues that functions of specific compositional structure can be approximated efficiently by deep neural networks matching that structure. In Section 2.5, I introduce the convolutional neural network, the most powerful and popular variant of neural network in use today. Convolutional neural networks have roots in computer vision research of the 1980s[LBD+89], but are being applied to increasingly diverse fields[GAG+17, CFS16]. The chapter concludes with Section 2.6, which describes how to integrate convolutional neural networks into a theoretical machine learning framework.

The remaining chapters discuss practical neural networks and their application to a Kaggle competition. The results of Sections 2.3.1, 2.3.2, and 2.4 are not directly applicable to practical neural networks, since those sections describe the optimal approximation of known functions, whereas in practical circumstances we do not know the function we are approximating and cannot tell if the neural network we have trained is optimal. Nevertheless, Chapter 2 provides general motivation for the use of neural networks by theoretically analyzing their expressive power.

Chapter 3 treats various topics pertaining to optimization for practical convolutional neural networks. These topics have a less firm theoretical foundation than those of the previous chapter, but they are of great practical importance. Section 3.2 introduces stochastic gradient descent and minibatch gradient descent, and offers a few arguments as to why they are preferred to standard gradient descent for optimizing convolutional neural networks. In Section 3.3, I discuss batch normalization, a well-established but poorly-understood technique for improving the optimization of convolutional neural networks. I cite some recent research that may better describe the mechanism by which batch normalization aids optimization than more traditional explanations.

As an illustration of the application of deep neural networks, Chapters 4 and 5 describe my submission to an audio-classification competition. The contest organizers provided participants with collection of audio clips, each with one of 41 labels such as "Acoustic Guitar", "Fireworks", or "Bus". The participants were instructed to use the audio clips and a machine learning algorithm of their choice to predict the labels of previously unseen clips. My submission, an ensemble of deep convolutional neural networks trained on the audio clips, scored in the top 3%. Chapter 4 focuses on the short-time Fourier transform (STFT), a way of analyzing how the frequency components of a sound change over time. I first state a simple theoretical result that reveals some important properties of the STFT, and then discuss some practical aspects. This chapter is of great import, since my convolutional neural networks accept as input not the raw audio clips, but their spectograms, which are generated using the STFT.

I begin Chapter 5 by drawing upon psychological research into the way humans process sound to argue that spectrograms are reasonable inputs for convolutional neural networks. I then move on to describe the specific convolutional neural networks I used in my submission. Finally, I discuss the data balancing, masking, and augmentation schemes used in my submission and in that of the contest winners. I conclude by noting the general importance of these data processing techniques to the training of deep neural networks.

Chapter 2

Neural Network Theory

In this chapter, we describe a formal framework for neural networks and survey some theoretical results.

2.1 Computation Graphs

The distinguishing feature of neural networks as function approximators is their compositional structure. Because it is suitable for describing compositional structure, we begin by introducing the computation graph framework.

Definition 1. Let (V, E) be a directed graph. Given a vertex $v \in V$, define its *input* and *output vertexes* by

$$\text{in}(v) = \{u \in V |\ (u, v) \in E\} \tag{2.1}$$
$$\text{out}(v) := \{u \in V |\ (v, u) \in E\}. \tag{2.2}$$

The graph's *input* and *output vertexes* are given by

$$V_{\text{in}} := \{v \in V |\ \#\,\text{in}(v) = 0\} \tag{2.3}$$
$$V_{\text{out}} := \{v \in V |\ \#\,\text{out}(v) = 0\}. \tag{2.4}$$

To simplify notation, for any symbol s and any vertex v we write $s_{\text{in}(v)}$ to mean $\bigoplus_{u \in \text{in}(v)} s_u$ where $\bigoplus$ denotes Cartesian product when the s_u are sets and tuple formation when the s_u are elements.

Definition 2. A *computation graph* consists of

1. A directed acyclic graph (V, E),
2. For each vertex $v \in V$, sets $\mathcal{D}_v$ and $\mathcal{R}_v$ (called the *vertex domain* and *vertex range*) such that at non-input vertexes ($v \notin V_{\text{in}}$), these sets satisfy

$$\mathcal{D}_v = \mathcal{R}_{\text{in}(v)}, \tag{2.5}$$

3. For each vertex $v \in V$, a *vertex function* $c_v : \mathcal{D}_v \to \mathcal{R}_v$.

The *range* $\mathcal{R}$ of a computation graph is the Cartesian product of the vertex ranges of its output vertexes. The *domain* $\mathcal{D}$ of a computation graph is the Cartesian product of the vertex domains of its input vertexes. In this essay, the domain, range, vertex domains, and vertex ranges of every computation graph will be subsets of Euclidean spaces. The *depth* of a computation graph is the length of its longest path. The non-input non-output vertexes in a computation graph are called *hidden units.*

Definition 3. A computation graph G *induces* a function $F : \mathcal{D} \to \mathcal{R}$ defined via intermediate functions $F_v : \mathcal{D} \to \mathcal{R}_v$ for $v \in V$. The F_v are recursively given by

$$F_v(x) := \begin{cases} c_v(x_v) & \text{if } v \in V_{\text{in}} \\ c_v\left(F_{\text{in}(v)}(x)\right) & \text{otherwise} \end{cases}, \tag{2.6}$$

where x_v denotes the component of x corresponding[1] to v. F is given by $F(x) := \bigoplus_{v \in V_{\text{out}}} F_v(x)$, where again $\bigoplus$ denotes tuple formation.

Thus F operates on an input by propagating it through the computation graph G using its vertex functions. This procedure is called *forward propagation*.

The function induced by a computation graph can be differentiated via a recursive procedure called backward propagation, which is key to efficient implementation. See Section A.

Example 1. Suppose we wish to implement the function $f_{\alpha,\beta} : \mathbb{R}^2 \to \mathbb{R}$ with $f_{\alpha,\beta}(x, y) = (\alpha x + \beta y)^2$. Set $V = \{M_1, M_2, S, Q\}$, $E = \{(M_1, S), (M_2, S), (S, Q)\}$, $\mathcal{D}_{M_1} = \mathcal{D}_{M_2} = \mathbb{R}$ and $\mathcal{R}_v = \mathbb{R}$ for all $v \in V$. Also set

$$c_{M_1}(z_) = \alpha z \tag{2.7}$$
$$c_{M_2}(z) = \beta z \tag{2.8}$$
$$c_S(z_1, z_2) = z_1 + z_2 \tag{2.9}$$
$$c_Q(z) = z^2. \tag{2.10}$$

This computation graph implements the desired function.

2.2 Neural Networks

Having established the computation graph framework, we are now in a position to introduce neural networks, a form of computation graph in which all nonlinearies are scalar.

Definition 4. With the notation of Section 2.1, a vertex in a computation graph is said to be a *simple neuron* with activation function σ for some $\sigma : \mathbb{R} \to \mathbb{R}$ if

1. $\mathcal{R}_v = \mathbb{R}$ and $\mathcal{D}_v = \mathbb{R}^{n_v}$ for some integer n_v.
2. For all $x \in \mathcal{D}_v$

$$c_v(x) = \sigma(\langle x, a \rangle + b) \tag{2.11}$$

for some $a \in \mathcal{D}_v$ and $b \in \mathbb{R}$ (where $\langle \cdot, \cdot \rangle$ denotes the standard real inner product: $\langle x, a \rangle = \sum_i x_i a_i$).

Definition 5. With the notation of Section 2.1, a vertex v in a computation graph is said to be a *transfer vertex* if

1. $\mathcal{R}_v = \mathcal{D}_v$
2. c_v is the identity function.

Definition 6. A computation graph is a *simple neural network* with activation function σ for some $\sigma : \mathbb{R} \to \mathbb{R}$ if every input vertex is a transfer vertex, every output vertex is a simple neuron with the identity as its activation function, and every non-input non-output vertex is a simple neuron with σ as its activation function.

[1]i.e. $x = \bigoplus_{v \in V} x_v$

Remark 1. Neural networks derive their name from their original interpretation, which was biological. The neurons of a neural network were seen as modeling biological neurons. $v \in \text{in}(w)$ indicated that neuron v should be viewed as being connected to neuron w by an axon. σ was generally the step function

$$\sigma(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2.12}$$

or a close approximation to it. $\sigma(x) = 1$ indicated that the voltages transmitted along the input axons to a given neuron were sufficient to excite an action potential, while $\sigma(x) = 0$ indicated that they were insufficient. Neural networks used for machine learning have since diverged from those used for biological modeling in response to the differing demands of the two fields. For the rest of this essay we focus on the former (see [ESC+12] for an example of the latter).

We introduce a particularly simple form of neural network, which has the advantage of having fairly well-understood theoretical properties.

Definition 7. Let $\sigma : \mathbb{R} \to \mathbb{R}$. A simple neural network with activation function σ is called a *simple shallow neural network* with activation function σ if

1. It has depth 2.
2. It has one input vertex.
3. It has one output vertex.

Simple shallow neural networks represent functions of the form $x \mapsto \sum_{i=1}^{w} v_i \sigma(\langle x, a^i \rangle + \theta_i) + \mu$. Without loss of generality we will assume $\mu = 0$. Figure 2.1 illustrates the structure of simple shallow neural networks.

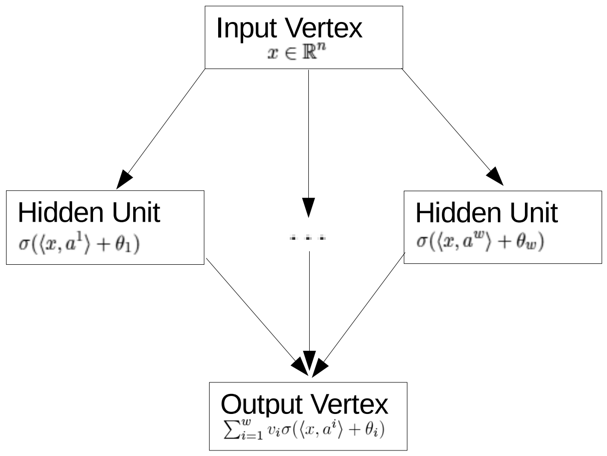
Input Vertex
$x \in \mathbb{R}^n$
Hidden Unit
$\sigma(\langle x, a^1 \rangle + \theta_1)$
...
Hidden Unit
$\sigma(\langle x, a^w \rangle + \theta_w)$
Output Vertex
$\sum_{i=1}^{w} v_i \sigma(\langle x, a^i \rangle + \theta_i)$

Figure 2.1: A simple shallow neural network.

2.3 Expressive Power of Neural Networks

2.3.1 Density of Shallow Networks

As neural networks are function approximators, it is natural to ask what functions they approximate. We defer until Section 2.3.2 the issue of efficiency, and here study only whether or not a given function can be approximated. In topological terms, we are interested in the sets of functions in which neural networks are dense. For this purpose, it will be sufficient to study only the simple shallow neural networks described in Definition 7, as we will find that any continuous function can be approximated by such a network with minimal conditions. The main propositions and proofs in this section are taken from [Pin99].

The study of simple shallow neural networks is closely linked to the study of ridge functions[Pin15].

Definition 8. A *ridge function* is a function $f : \mathbb{R}^n \to \mathbb{R}$ of the form

$$f(x) := g(\langle a, x\rangle) \tag{2.13}$$

for some $a \in \mathbb{R}^n$, and $g \in C(\mathbb{R})$ ($C(\mathbb{R})$ denotes the set of all continuous functions on $\mathbb{R}$). Define the span of the ridge functions in n dimensions as:

$$\text{ridge}_n := \text{span}\{g(\langle \cdot, a\rangle) | a \in \mathbb{R}^n, g \in C(\mathbb{R})\}, \tag{2.14}$$

where as usual $\text{span}(S)$ is the set of linear combinations of the functions in S.

Remark 2. Since ridge_n includes all functions of the form $x \mapsto \sum_{j=0}^{w_1} \sum_{i=0}^{w_2} \rho_{i,j} \langle a^j, x\rangle^i$ for $w_1, w_2 \in \mathbb{Z}_+, \rho_{i,j} \in \mathbb{R}, a^j \in \mathbb{R}^n \ \forall i, j$, and since every polynomial can be written in this form (we prove this in Proposition B.3), it follows by the Stone-Weierstrass Theorem[R$^+$76, Theorem 7.32 pg 162] that ridge_n is dense in the continuous functions on $\mathbb{R}^n$. Our strategy for proving that simple shallow neural networks with an appropriate activation function are dense in $C(\mathbb{R}^n)$ will employ this result as a dimension-reduction technique: we will approximate an arbitrary continuous function on $\mathbb{R}^n$ by a sum of ridge functions, and then approximate the one-dimensional g associated with each ridge function by a simple shallow neural network. We begin by defining the set of functions representable as simple shallow neural networks.

Definition 9. Assume $\sigma : \mathbb{R} \to \mathbb{R}$. Define

$$\text{SSNN}_n(\sigma) := \text{span}\{\sigma(\langle \cdot, a\rangle + \theta) | a \in \mathbb{R}^n, \theta \in \mathbb{R}\}. \tag{2.15}$$

We also define the set of functions representable by simple shallow neural networks with a fixed number of hidden units r.

Definition 10. Assume $\sigma : \mathbb{R} \to \mathbb{R}$ and $r > 0$. Define

$$\text{SSNN}_n^r(\sigma) := \{\sum_{i=1}^{r} \rho_i \sigma(\langle \cdot, a^i\rangle + \theta_i) | a^i \in \mathbb{R}^n, \theta_i \in \mathbb{R}, \rho_i \in \mathbb{R} \ \forall i\}. \tag{2.16}$$

Remark 3. We noted in Section 2.1 that a distinguishing characteristic of neural networks is their compositional structure. This observation applies even to simple shallow neural networks: we generate a member of the family $\text{SSNN}_n^r(\sigma)$ by pre-composing σ with r different affine functions, then taking a linear combination of the results.

Proposition 1 enables the dimension-reduction technique outlined in Remark 2.

Proposition 1 (Proposition 3.3 in [Pin99])**.** *Apply the topology of uniform convergence on compact sets to $C(\mathbb{R})$ and $C(\mathbb{R}^n)$. Let $\sigma : \mathbb{R} \to \mathbb{R}$. Suppose $\text{SSNN}_1(\sigma)$ is dense in $C(\mathbb{R})$. Then $\text{SSNN}_n(\sigma)$ is dense in $C(\mathbb{R}^n)$.*

Proof. Let $f \in C(\mathbb{R}^n)$. Given $\epsilon > 0$ and a compact set $K \subset \mathbb{R}^n$, since ridge_n is dense in $C(\mathbb{R}^n)$ (see Remark 2), we can find $r > 0$, $g_1, \ldots, g_r \in C(\mathbb{R})$, and $a^1, \ldots, a^r \in \mathbb{R}^n$ such that

$$\sup_{x \in K} |f(x) - \sum_{i=1}^{r} g_i(\langle a^i, x\rangle)| < \frac{\epsilon}{2}. \tag{2.17}$$

Note that since K is compact and $\langle \cdot, a^i \rangle$ is continuous, $K_i := \langle K, a^i \rangle$ is compact for all i.

Since $\mathrm{SSNN}_1(\sigma)$ is dense in $C(\mathbb{R})$ by hypothesis, for each g_i we can find $s_i \in \mathbb{Z}_{++}$, $\rho_{i,1}, \ldots, \rho_{i,s_i} \in \mathbb{R}$, $\lambda_{i,1}, \ldots, \lambda_{i,s_i} \in \mathbb{R}$, $\theta_{i,1}, \ldots, \theta_{i,s_i} \in \mathbb{R}$ such that

$$\sup_{y \in K_i} |g_i(y) - \sum_{j=1}^{s_i} \rho_{i,j} \sigma(\lambda_{i,j} y + \theta_{i,j})| < \frac{\epsilon}{2r}. \tag{2.18}$$

Thus we have

$$\begin{aligned}
&\sup_{x \in K} |f(x) - \sum_{i=1}^{r} \sum_{j=1}^{s_i} \rho_{i,j} \sigma(\lambda_{i,j} \langle a^i, x \rangle + \theta_{i,j})| && (2.19)\\
&\leq \sup_{x \in K} |f(x) - \sum_{i=1}^{r} g_i(\langle a^i, x \rangle)| + \sum_{i=1}^{r} |g_i(\langle a^i, x \rangle) - \sum_{j=1}^{s_i} \rho_{i,j} \sigma(\lambda_{i,j} \langle a^i, x \rangle + \theta_{i,j})| && (2.20)\\
&< \epsilon && (2.21)
\end{aligned}$$

as desired. □

We have reduced the problem of approximating by simple shallow neural networks on $\mathbb{R}^n$ to the corresponding problem on $\mathbb{R}$, which we now tackle. In what follows, $\mathrm{cl}(S)$ denotes the closure of the set S.

Proposition 2 (Proposition 3.4 in [Pin99][2]). *Apply the topology of uniform convergence on compact sets to $C(\mathbb{R})$. Assume $\sigma \in C^\infty(\mathbb{R})$ is not a polynomial. Then for all $s > 0$, the polynomials of degree less than or equal to s lie in $\mathrm{cl}\,\mathrm{SSNN}_1^{s+1}(\sigma)$. Hence all polynomials lie in $\mathrm{cl}\,\mathrm{SSNN}_1(\sigma)$.*

Proof. Fix $s > 0$ and let $0 \leq r \leq s$. Since $\sigma \in C^\infty(\mathbb{R})$ but is not a polynomial, there must exist a $\theta_r \in \mathbb{R}$ such that[3] $\sigma^{(r)}(\theta_r) \neq 0$. For $\lambda \in \mathbb{R}$, $h \in (0, \infty)$, and $k \in \{0, \ldots, r\}$ consider $f(\cdot; \lambda, h, k) : \mathbb{R} \to \mathbb{R}$ given by $f(t; \lambda, h, k) := \sigma((\lambda + kh)t + \theta_r)$. For each choice of h and λ, the $r + 1$ functions $\{f(\cdot; h, \lambda, k)\}_{k=0}^{r}$ can be used to construct an order-h forward-difference approximation[4] to the function $\frac{d^r}{d\lambda^r} f(\cdot; 0, \lambda, 0)$. Since it is a linear combination of $r + 1$ affine pre-compositions of σ, this approximation lies in $\mathrm{SSNN}_1^{r+1}(\sigma) \subset \mathrm{SSNN}_1^{s+1}(\sigma)$. Furthermore for a particular λ, when t is restricted to a compact set, this finite difference approximation converges uniformly[5] in t as $h \downarrow 0$ to the function $t \mapsto \frac{d^r}{d\lambda^r} \sigma(\lambda t + \theta_r)$, which therefore lies in $\mathrm{cl}\,\mathrm{SSNN}_1^{r+1}(\sigma)$. By chain rule this function is $t \mapsto t^r \sigma^{(r)}(\lambda t + \theta_r)$. Choosing $\lambda = 0$, we have that $t \mapsto t^r \sigma^{(r)}(\theta_r)$ lies in $\mathrm{cl}\,\mathrm{SSNN}_1^{r+1}(\sigma)$. But by our choice of θ_r, we have $\sigma^{(r)}(\theta_r) \neq 0$, so the monomial $t \mapsto t^r$ lies in $\mathrm{cl}\,\mathrm{SSNN}_1^{r+1}(\sigma)$. Hence all monomials of degree less than or equal to s lie in $\mathrm{cl}\,\mathrm{SSNN}_1^{s+1}(\sigma)$. So all polynomials of degree less than or equal to s lie in $\mathrm{cl}\,\mathrm{SSNN}_1^{s+1}(\sigma)$ as desired. □

Remark 4. In Remark 3 we noted that a key attribute of families of neural networks is their compositional structure. This structure was used to the proof of Proposition 2, where it enabled the application of the chain rule.

Proposition 2's requirement that $\sigma \in C^\infty(\mathbb{R})$ can be weakened so that σ need only be continuous.

[2] Modified to remove dependence on another theorem.

[3] $\sigma^{(r)}$ denotes the rth derivative of σ.

[4] i.e. $\frac{d^r}{d\lambda^r} f(\cdot; 0, \lambda, 0) = \frac{d^r}{d\lambda^r} \sigma(\lambda \cdot + \theta_r) \approx \frac{1}{h^r} \sum_{k=0}^{r} (-1)^{r-k} \binom{r}{k} \sigma((\lambda + kh) \cdot + \theta_r) = \frac{1}{h^r} \sum_{k=0}^{r} (-1)^{r-k} \binom{r}{k} f(\cdot; h, \lambda, k)$. See (2.17), (2.23), and (2.29) in[Boo81].

[5] To see this, note that by Theorem 2.2 in [Boo81], our finite difference approximation equals $\frac{d^r}{d\lambda^r} f(\cdot; 0, \lambda', 0)$ for some $\lambda' \in (\lambda, \lambda + rh)$. Since we know $\sigma \in C^\infty(\mathbb{R})$, we can bound $|\frac{d^r}{d\lambda^r} f(\cdot; 0, \lambda', 0) - \frac{d^r}{d\lambda^r} f(\cdot; 0, \lambda, 0)| = |(\lambda' - \lambda) \frac{d^{r+1}}{d\lambda^{r+1}} f(\cdot; 0, \lambda'', 0)| \leq rh |\frac{d^{r+1}}{d\lambda^{r+1}} f(\cdot; 0, \lambda'', 0)|$ for some $\lambda'' \in (\lambda, \lambda + rh)$. Since $t \in K$ for a compact interval K, by choosing $h < \tilde{h}$ for some $\tilde{h}$, we have that $\lambda'' t + \theta_r \in [\lambda, \lambda + r\tilde{h}]K + \theta_r$, which is also a compact interval. By continuity this provides us with a uniform bound in t on $|\frac{d^{r+1}}{d\lambda^{r+1}} f(t; 0, \lambda'', 0)| = |t^{r+1} \sigma^{(r+1)}(\lambda'' t + \theta_r)|$ and thus a uniform bound in t on the approximation error.

Proposition 3 (Proposition 3.7 in [Pin99])**.** *Apply the topology of uniform convergence on compact sets to $C(\mathbb{R})$. Assume $\sigma \in C(\mathbb{R})$ is not a polynomial. Then all polynomials lie in $\operatorname{cl}\mathrm{SSNN}_1(\sigma)$.*

See Section B for the proof. We now have the tools to establish a very general density result.

Proposition 4. *Apply the topology of uniform convergence on compact sets to $C(\mathbb{R}^n)$. Assume $\sigma \in C(\mathbb{R})$ is not a polynomial. Then $\mathrm{SSNN}_n(\sigma)$ is dense in $C(\mathbb{R}^n)$.*

Proof. By Proposition 3, $\operatorname{cl}\mathrm{SSNN}_1(\sigma)$ contains the polynomials. By the Stone-Weirstrauss theorem, the polynomials are dense in $C(\mathbb{R})$[R$^+$76, pg. 162 Theorem 7.32]. By Proposition 1, we have the desired result. $\square$

Remark 5.

1. Proposition 4 shows that simple shallow neural networks are very expressive, as they possess the prodigious approximation capabilities of the polynomials. This observation can form the start of an argument for the use of neural networks as function approximators, but much ground remains to be covered. In Section 2.3.2 we will consider the efficiency of simple shallow neural networks, while in Section 2.4 we will survey a few arguments about the benefits of depth.

2. The requirement that the activation function be nonpolynomial is necessary. A neural network with fixed depth and a fixed polynomial activation function is itself a polynomial whose degree is bounded by a function of the depth and activation function. Since the set of polynomials of degree bounded by a constant is closed, such networks cannot be universal approximators.

3. There are numerous alternative approaches for proving analogous results. For example, the authors of [Hor93] and [Cyb89] take a functional analysis approach. To get a contradiction they assume that $\operatorname{cl}\mathrm{SSNN}_n(\sigma)$ is not all of $C(\mathbb{R}^n)$. Under this assumption, they apply a corollary of the Hahn-Banach Theorem[Fol13, Theorem 5.8 pg. 159] to find a nonzero linear functional that is zero on $\operatorname{cl}\mathrm{SSNN}_n(\sigma)$. They then use the Riesz Representation Theorem[Fol13, Theorem 7.17 pg. 223] to represent this functional as a nonzero Radon measure. Finally, they generate a contradiction by showing that Radon measures that integrate to zero for all functions in $\mathrm{SSNN}_n(\sigma)$ must be zero.

4. The proofs presented here and the functional analysis proofs referred to above are nonconstructive, and hence provide no information on how to build the approximating networks. Several researchers have given constructive proofs. For example, the authors of [CCL95] assume a sigmoidal activation function, approximate their target functions by piecewise constant functions, and then approximate these piecewise constant functions with their neural networks. These constructive proofs tend to be more complex and apply to a more limited class of activation functions than nonconstructive proofs.

5. Unlike Proposition 2, Proposition 3 does not specify how many neurons are required to approximate polynomials of given degree. This is because of the Riemmann integration step in the proof of Proposition 3 (see Section B).

2.3.2 Order of Approximation of Shallow Neural Networks

Section 2.3.1 showed that neural networks have the universal approximation property by relating them to polynomials, which are known to have this property. This section takes a similar approach to establishing the order of approximation of neural networks. Throughout this section, we fix a compact convex set $K \subset \mathbb{R}^n$ that contains an open ball. All function norms[6] are with respect to K. We rely on the following result about polynomials[7].

[6]When we say $f \in C^m(K)$ for a compact set K, we mean $f \in C^m(O)$ for an open set O such that $K \subset O$. Nevertheless all norms are with respect to K. For example: $\|f - g\|_\infty = \sup_{x\in K} |f(x) - g(x)|$.

[7]We use muli-indexes as defined in Section B.2.1.

Proposition 5 (Theorem 2 in [BBL02]). *Let $K \subset \mathbb{R}^n$ be a compact convex set. Let $m \in [0, \infty)$. Let $f \in C^m(K)$. Then for each positive integer k, there is a polynomial p of degree at most k on $\mathbb{R}^n$ such that*

$$\|f - p\|_\infty \leq \frac{C}{k^m} \sum_{|\alpha| \leq m} \|D^\alpha f\|_\infty \tag{2.22}$$

where C depends on n and m.

Our ability to approximate a function efficiently will depend on the rapidity with which it varies, as expressed by the magnitude of its derivatives. Accordingly, define for $m \geq 1$ and $q \in [1, \infty]$ the function-space balls

$$\mathcal{B}^{m,q,n} := \{f \in C^m(K) : \|D^\alpha f\|_q \leq 1 \ \forall |\alpha| \leq m\}. \tag{2.23}$$

The main result on the order of approximation of simple shallow networks is Proposition 6, which tells us that approximating a function from $\mathcal{B}^{m,\infty,n}$ using u neurons is order $u^{-\frac{m}{n}}$.

Proposition 6 (Theorem 6.8 in [Pin99]). *Assume $\sigma \in C^\infty(\mathbb{R})$ is not a polynomial. Then for all $m \geq 1$, $n \geq 2$, $u \geq 1$, we have*

$$\sup_{f \in \mathcal{B}^{m,\infty,n}} \inf_{g \in \mathrm{SSNN}_n^u(\sigma)} \|f - g\|_\infty \leq C u^{-\frac{m}{n}}. \tag{2.24}$$

where C depends on n and m.

Proof. Let C_1 be a a positive integer constant such that[8] for all $k \geq 1$ we have $(k+1)\binom{n-1+k}{n-1} \leq C_1 k^n$. We first consider the case in which $(\frac{u}{C_1})^{\frac{1}{n}}$ is an integer. Set $k := (\frac{u}{C_1})^{\frac{1}{n}}$. Let $w := \binom{n-1+k}{n-1}$. For any polynomial p in n variables of degree less than or equal to k, we know that by Proposition B.3 that we can write p as $p(x) = \sum_{i=1}^w \pi_i(\langle x, a^i \rangle)$ where $a^1, \ldots, a^w \subset \mathbb{R}^n$ and where the π_i are univariate polynomials of degree less than or equal to k. By Proposition 2 each such univariate polynomial lies in $\mathrm{cl}\,\mathrm{SSNN}_1^{k+1}(\sigma)$. It follows that we can approximate p to arbitrary accuracy using $\mathrm{SSNN}_n^{w(k+1)}$. Thus letting P_k denote the space of polynomials in n variables of degree less than or equal to k, we have

$$P_k \subset \mathrm{cl}\,\mathrm{SSNN}_n^{w(k+1)}(\sigma) \subset \mathrm{cl}\,\mathrm{SSNN}_n^{C_1 k^n}(\sigma) = \mathrm{cl}\,\mathrm{SSNN}_n^u(\sigma). \tag{2.25}$$

Thus

$$\begin{aligned}
&\sup_{f \in \mathcal{B}^{m,\infty,n}} \inf_{g \in \mathrm{SSNN}_n^u(\sigma)} \|f - g\|_\infty \\
&\leq \sup_{f \in \mathcal{B}^{m,\infty,n}} \inf_{g \in P_k} \|f - g\|_\infty \\
&\leq C_2 k^{-m} && \text{by Proposition 5 (C_2 depends on m and n)} \\
&\leq C_1^{\frac{m}{n}} C_2 u^{-\frac{m}{n}} && \text{by choice of } k.
\end{aligned} \tag{2.26}$$

Note that the constants depend on m and n.

Now consider the case in which $u > C_1$, but $(\frac{u}{C_1})^{\frac{1}{n}}$ is not an integer. Then let $u' := C_1 \left(\lfloor \left(\frac{u}{C_1} \right)^{\frac{1}{n}} \rfloor \right)^n$,

[8]For example, we can use the very loose bound $(k+1)\binom{n-1+k}{n-1} \leq \frac{(k+1)(n-1+k)^{n-1}}{(n-1)!} \leq \frac{(k+1)\left(\frac{n-1+k}{k}\right)^{n-1}}{(n-1)!} k^{n-1} \leq \lceil 2\frac{n^{n-1}}{(n-1)!} \rceil k^n$.

which is an integer. We can bound $\frac{u}{u'}$ by

$$\begin{aligned}\frac{u}{u'} &= \frac{1}{C_1}\left(\frac{u^{\frac{1}{n}}}{\lfloor\left(\frac{u}{C_1}\right)^{\frac{1}{n}}\rfloor}\right)^n \\ &= \left(\frac{\left(\frac{u}{C_1}\right)^{\frac{1}{n}}}{\lfloor\left(\frac{u}{C_1}\right)^{\frac{1}{n}}\rfloor}\right)^n \\ &\leq 2^n. \qquad (2.27)\end{aligned}$$

We have

$$\begin{aligned}\sup_{f\in\mathcal{B}^{m,\infty,n}}\inf_{g\in\mathrm{SSNN}^u_n(\sigma)}\|f-g\|_\infty &\leq \sup_{f\in\mathcal{B}^{m,\infty,n}}\inf_{g\in\mathrm{SSNN}^{u'}_n(\sigma)}\|f-g\|_\infty \\ &\leq C_1^{\frac{m}{n}}C_2\frac{1}{(u')^{\frac{m}{n}}} && \text{by previous case} \\ &= C_1^{\frac{m}{n}}C_2\frac{1}{u^{\frac{m}{n}}}(\frac{u}{u'})^{\frac{m}{n}} \\ &\leq C_1^{\frac{m}{n}}C_2 2^m\frac{1}{u^{\frac{m}{n}}} && \text{using bound from (2.27).} \qquad (2.28)\end{aligned}$$

Now consider the $C_1 - 1$ cases in which $u < C_1$. In these cases, note that since $f \in B^{m,\infty,n}$, we have $\|f\|_\infty \leq 1$ and we can simply take $g = 0$ and choose a sufficiently large constant.

Now we have proved the result for each of $1 + C_1$ cases, each with a different constant depending on m and n. Taking the final C to be the maximum of these constants, we have the desired result.

□

Unlike in the analogous situation in Section 2.3.1, extending Proposition 6 to non-$C^\infty(\mathbb{R})$ activation functions is not straightforward. [Pin99, Corollary 6.10] quotes a result from [Pet98]:

Proposition 7. *Let $s \in \mathbb{Z}_+$ and let*

$$\sigma(t) := \begin{cases} t^s & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases}. \qquad (2.29)$$

Then if $m \in \{1, \ldots, s+1+\frac{n-1}{2}\}$, we have

$$\sup_{f\in B^{m,2,n}}\inf_{g\in\mathrm{SSNN}^u_n(\sigma)}\|f-g\|_2 \leq Cu^{-\frac{m}{n}} \qquad (2.30)$$

where C depends on m and n.

This class of functions considered in Proposition 7 includes the Heaviside step function as well as the popular $\mathrm{ReLU}(x) := \max(x, 0)$. Unlike Proposition 6, the present proposition is limited to the L_2 norm. As we shall see in Section 2.4, Proposition 7 becomes false when L_2 is replaced by L_∞.

Remark 6.

1. Having upper-bounded the order of approximation by $Cu^{-\frac{m}{n}}$ with Propositions 6 and 7, we might ask for a corresponding lower bound. [Mai10, Theorem 1] shows that for $p \in [1, \infty]$, there is an f in the Sobolev space $W^{m,p}(K)$ (which can be thought of an the completion of $C^m(K)$ under a certain norm) such that the error of approximating f using a linear combination of u ridge functions is bounded below by $Cu^{-\frac{m}{n-1}}$. Since the functions representable as simple shallow neural networks are a subset of the functions representable as sums of ridge functions, this by provides a lower bound for approximation by simple shallow neural networks.

2. The results of this section and Section 2.3.1 are proven by transferring the approximation properties of polynomials to simple shallow neural networks. On the one hand, the polynomials have many desirable approximation properties, and so this is a reasonable approach. On the other hand, in practice neural networks exhibit performance far exceeding any polynomial method. To justify this performance it is necessary either to consider specific classes of functions on which neural networks perform well (we shall do some of this in Section 2.4), or to look beyond approximation theory, perhaps to the theory of generalization in machine learning.

2.4 The Approximation Benefits of Depth

2.4.1 The Number of Linear Regions of Deep and Shallow Networks

Most neural networks used successfully in practice are deep, but the most well-established neural network theory pertains to shallow networks. Recently, attempts have been made to theoretically explain the benefits of depth. While these attempts are diverse, many can be seen as instances of the same underlying principle. The principle pertains to neural networks with piecewise linear activation functions, which we call piecewise linear networks. It consists of the observation that a function induced by a deep piecewise linear network can have many more linear pieces than one induced by a shallow piecewise linear network with the same number of neurons.

One Dimensional Example

In the one dimensional case, the principle is demonstrated easily, as [Tel15] showed.

Proposition 8. *Let $f, g : [0,1] \to \mathbb{R}$ be piecewise linear, with n_f and n_g pieces respectively. Then $f + g$ has at most $n_f + n_g$ pieces.*

Proof. This follows from the observation that each maximal interval over which $f + g$ is linear has as its left endpoint the left endpoint of a maximal interval over which f is linear or the left endpoint of a maximal interval over which g is linear. Since there are at most $n_f + n_g$ such left endpoints, we have the desired result. □

That is, simple shallow piecewise linear networks have a number of pieces linear in their number of neurons. Now consider enlarging a piecewise linear network by making it deeper, which corresponds to composing piecewise linear functions. Using the notation of the previous proposition, $f \circ g$ has at most $n_f n_g$ pieces, which is achieved when the image of g over each of its maximal intervals of linearity intersects all the intervals of linearity of f. That is, deep piecewise linear networks can have a number of linear pieces exponential in their number of neurons.

The following example from [Tel15, Lemma 2.4] exhibits a function with a large number of linear pieces that can consequently be approximated much more efficiently by deep piecewise linear networks than by shallow piecewise linear networks.

Example 2. Define the *mirror function* $m : [0,1] \to [0,1]$ by

$$m(x) = \begin{cases} 2x & \text{if } 0 \le x \le \frac{1}{2} \\ 2(1-x) & \text{if } \frac{1}{2} < x \le 1 \end{cases}. \tag{2.31}$$

Note that since $m(x) = 2\,\mathrm{ReLU}(x) - 4\,\mathrm{ReLU}(x - \frac{1}{2})$ for $x \in [0,1]$, m is exactly representable by a simple shallow neural network with ReLU activation (recall $\mathrm{ReLU}(x) = \max(x, 0)$).

For $k \in \mathbb{Z}_+$, define the sawtooth function $s_k : [0,1] \to [0,1]$ as follows. For $x \in [0,1]$, let $r_k(x) := 2^k x - \lfloor 2^k x \rfloor$. Then define

$$s_k(x) := m(r_k(x)). \tag{2.32}$$

See Figure 2.2. We shall show that

$$s_k = \underbrace{m \circ m \circ \ldots \circ m}_{k+1 \text{ times}} =: m^{k+1}. \tag{2.33}$$

The proof is by induction on k. The base case ($s_0 = m$) is clear. Now assume that the result is true for k. We shall prove it for $k+1$. Note that m^{k+1} is clearly mirror symmetric about $\frac{1}{2}$. We also claim that that s_k is also mirror symmetric about $\frac{1}{2}$. To prove this, first consider the case in which $2^{k+1}x$ is not an integer[9]. We have

$$\begin{aligned}
&s_k(1-x)\\
&= m(r_k(1-x))\\
&= m(2^k(1-x) - \lfloor 2^k(1-x) \rfloor)\\
&= m(2^k - 2^k x - \lfloor 2^k(1-x) \rfloor)\\
&= m(1 + \lfloor 2^k(1-x) \rfloor + \lfloor 2^k x \rfloor - 2^k x - \lfloor 2^k(1-x) \rfloor)\\
&= m(1 - r_k(x))\\
&= m(r_k(x)) && \text{by mirror symmetry of } m\\
&= s_k(x).
\end{aligned} \tag{2.34}$$

Now consider the case in which $2^k x$ is an integer. Then so is $2^k(1-x)$, and thus

$$s_k(1-x) = m(r_k(1-x)) = m(0) = m(r_k(x)) = s_k(x), \tag{2.35}$$

proving our claim that s_k is mirror symmetric abour $\frac{1}{2}$. Since both s_k and m^{k+1} are mirror symmetric about $\frac{1}{2}$, it suffices to prove the result for $0 \leq x \leq \frac{1}{2}$. We have $m^{k+1} = s_{k-1} \circ m$ by the inductive assumption. Thus,

$$\begin{aligned}
&m^{k+1}(x) = s_{k-1} \circ m(x) = s_{k-1}(2x) = m(r_{k-1}(2x))\\
&= (2^{k-1}(2x) - \lfloor 2^{k-1}(2x) \rfloor) = m(2^k x - \lfloor 2^k x \rfloor) = m(r_k(x)) = s_k(x)
\end{aligned} \tag{2.36}$$

as desired. This completes the induction and establishes (2.33).

We have exhibited s_k as the composition of $k+1$ copies of m, which we already know is exactly representable as a simple shallow neural network with ReLU activation. Hence s_k is exactly representable by a (non-shallow) neural network with ReLU activation, depth $2(k+1)$, and a number of neurons linear in k. In contrast, since s_k has 2^{k+1} linear pieces, a shallow piecewise linear neural network requires $O(2^{k+1})$ neurons to exactly represent it, by Proposition 8. Furthermore, it is easy to show that the ∞-norm error of approximating s_k with too few linear pieces is lower bounded by a constant.

Extension Beyond One Dimension

The principle that deeper piecewise linear networks are more expressive than shallow piecewise linear networks because they can have more linear pieces with the same neuron budget is explored in different ways by numerous authors. [MPCB14, Mon17] extend the above analysis to n dimensions, drawing upon the theory of hyperplane arrangements[S$^+$04, Proposition 2.4] to show that a simple shallow neural network with ReLU activation, input dimension n, and u hidden units can have at most $\sum_{i=0}^{n} \binom{u}{i}$ linear pieces, which is polynomial in u. Since a deep neural network can have a number of linear pieces exponential in the number of neurons, the analysis of [MPCB14] confirms that in n dimensions, deep piecewise linear networks can have more linear pieces than simple shallow ones.

[9]We will use that if $b \in \mathbb{Z}_{++}$, $0 < a < b$, and a is not an integer, then $\lfloor b-a \rfloor + \lfloor a \rfloor + 1 = b$.

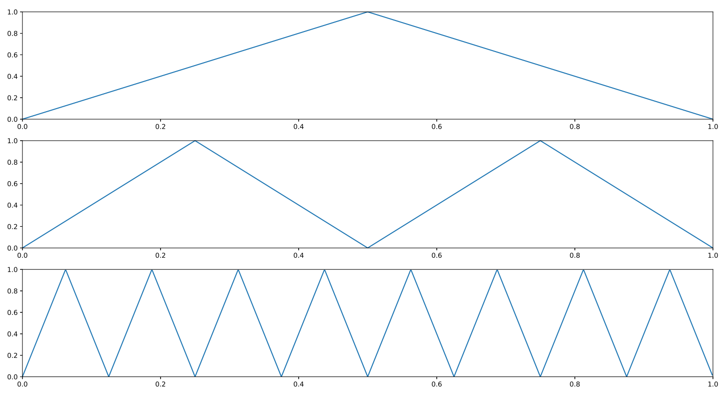

Figure 2.2: s_k for $k = 0, 1, 3$

The above results do not address the question of whether deep neural networks' many linear pieces are actually useful for approximating typical functions. So far, we have only seen one rather pathological function for which these many linear pieces are useful. [Yar17] attempts to address the issue with the following results:

Proposition 9 ([Yar17, Theorem 6]). *Let $n \geq 1$. Let $f \in C^2([0,1]^n)$ be a non-affine function. Fix $L > 2$. Let $u > 2$. Let g be a function induced by a simple neural network with* ReLU *activation, u neurons and depth less than or equal to L. We must have*

$$\|f - g\|_\infty \geq C u^{-2(L-2)} \tag{2.37}$$

where C depends on f and L.

Proposition 10 (([Yar17, Theorem 1] and [Yar18, pg. 2])). *Let $m \geq 1$ and $n \geq 1$. Let $f \in C^m([0,1]^n)$. Let $u > 1$ Then there exists a simple neural network with* ReLU *activation, u edges, and at most u neurons inducing a function g that satisfies*

$$\|f - g\|_\infty \leq C u^{-\frac{m}{n}} (\log u)^{\frac{m}{n}} \tag{2.38}$$

where C depends on m and n.

Comparing Propositions 10 and 9, we see that deep ReLU networks have better order of approximation than shallow ReLU networks when the function to be approximated is non-affine but very smooth (m is large). The proofs of both propositions rely on the same principle: deep ReLU nets have more linear pieces than shallow ones. To prove Proposition 9, the non-affine function f is locally approximated by a quadratic, and then the minimum error of approximating a quadratic with a given number of linear pieces is calculated. To prove Proposition 10, f is approximated by a piecewise polynomial, which is in turn approximated by the neural network, where the neural network's depth enables the high-degree terms of the piecewise polynomial to be efficiently approximated by piecewise linear functions with many pieces.

The results of [Yar17] are suggestive, but also unsatisfactory, since the order of approximation of deep ReLU nets in Proposition 10 is worse than that of shallow nets with smooth activations given in Proposition 6. Nevertheless, ReLU activations are the most commonly used activation functions in practice today, likely because of their compatibility with optimization. It may be that Proposition 9 and 10 explain at least part of power of depth for such networks.

Extension to Other Properties

In [Mon17], the author notes that the kind of analysis used to show that deep piecewise linear networks can have exponentially many linear pieces can be applied to properties other than the number of linear pieces. For example, one can show that deep ReLU neural networks can map exponentially many maximal connected disjoint input regions to the same output value, whereas shallow ReLU networks cannot. Furthermore, this analysis of the number of input regions mapping to the same output value can be extended beyond ReLU networks to networks with certain smooth nonlinear activation functions. Thus by several measures, deep neural networks are more complex than shallow ones.

2.4.2 Depth for Approximating Compositional Functions

An alternative to the approach of [Yar17], which compares the approximation power of shallow and deep ReLU nets over very general classes of functions is that of [MP16], which studies the power of deep networks for approximating functions with specific compositional structure.

Definition 11. Let (V, E) denote a directed acyclic graph with one output vertex. Let $Q \subset \mathbb{R}$. Let $\{S_d\}_{d \in \mathbb{Z}_{++}}$ be a family of sets of functions such that all $f \in S_d$ have domain Q^d (i.e. the d-fold Cartesian product of Q with itself) and range Q. Then define $\mathcal{G}(\{S_d\}, (V, E), Q)$ to be the set of every function representable by a computation graph with vertexes V and edges E such that for every vertex $v \in V$, the corresponding vertex domain $\mathcal{D}_v$, vertex range $\mathcal{R}_v$, and vertex function c_v satisfy

1. $\mathcal{R}_v = Q$.
2. If v is a non-input vertex, then $\mathcal{D}_v = Q^{\#\text{in}(v)}$ and $c_v \in S_{\#\text{in}(v)}$.
3. If v is an input vertex, $\mathcal{D}_v = Q$ and c_v is the identity.

$\mathcal{G}(\{S_d\}, (V, E), Q)$ describes a set of functions induced by computation graphs with a given graph structure whose vertex functions come from a given family. For convenience define $\text{indim}(v)$ to be $\#\text{in}(v)$ if v is not an input vertex and 1 if v is an input vertex.

Example 3. As a simple example, consider $V := \{V_1, V_2, V_3, V_4\}$, $E := \{(V_1, V_2), (V_1, V_3), (V_2, V_4), (V_3, V_4)\}$, $Q = [0, 1]$, and $S_d := \{x \mapsto \sigma_1(\langle x, a \rangle)\}_{a \in \mathbb{R}^d}$ with $\sigma_1(x) := \frac{1}{1+\exp(-x)}$ and $\sigma_2(x) = x$. Then $\mathcal{G}(\{S_d\}, (V, E), Q)$ includes (among other functions) squashed (into $[0, 1]$) versions of the one-dimensional simple shallow neural networks on with 2 hidden units.

To handle some technicalities, we need some additional definitions. First, given $\sigma : \mathbb{R} \to \mathbb{R}$ and $[a, b] \subset \mathbb{R}$ define $\text{SSNN}_n^\gamma(\sigma; [a, b])$ by replacing each function $f \in \text{SSNN}_n^\gamma(\sigma)$ with the function formed by restricting the domain of f to $[a, b]^n$ and post-composing f with $\min(\max(\cdot, a), b)$. Second, define $\mathcal{B}^{m,\infty,d}[0, 1]$ to be the restriction to $[0, 1]^d$ of those functions in $\mathcal{B}^{m,\infty,d}$ mapping $[0, 1]^d$ to a subset of $[0, 1]$.

Thus $\text{SSNN}_d^\gamma(\sigma; [0, 1])$ and $\mathcal{B}^{m,\infty,d}[0, 1]$ are sets of functions mapping $[0, 1]^d \to [0, 1]$. Note that the functions in $\mathcal{G}(\{\text{SSNN}_d^\gamma(\sigma; [0, 1])\}_{d \in \mathbb{Z}}, (V, E), [0, 1])$ could all be expressed with standard neural networks as described in Section 2.2. However, expressing them in the manner of the present section reveals their hierarchical structure: they are induced by computation graphs whose vertex functions are simple shallow neural networks. The key result is the following.

Proposition 11 ([MP16, Theorem 2.1b])**.** *Let (V, E) be a directed acylic graph. Let $\sigma \in C^\infty(\mathbb{R})$. Let $m, n \geq 1$. Let $\xi := \max_{v \in V} \text{indim}(v)$. Let $f \in \mathcal{G}(\{\mathcal{B}^{m\infty,d}[0, 1]\}_{d \in \mathbb{Z}_+}, (V, E), [0, 1])$. Then for each $\gamma > 0$ there exists a $g \in \mathcal{G}(\{\text{SSNN}_d^\gamma(\sigma; [0, 1])\}_{d \in \mathbb{Z}}, (V, E), [0, 1])$ such that*

$$\|f - g\|_\infty \leq C\gamma^{-\frac{m}{\xi}}. \tag{2.39}$$

where C depends on m, n, and the graph (V, E) but not γ.

Proof sketch. Note first that by assumption all the vertex functions of f lie in $\mathcal{B}^{m,\infty,d}$ for some $d \leq \xi$, so they are all Lipshitz with the same ∞-norm Lipshitz constant[10] $L := \xi$. We will inductively construct g. Let $v \in V$, and inductively assume that the vertex functions of g associated with vertexes preceding v in topological order have already been chosen. Let $d = \operatorname{indim}(v)$. Let f_v and g_v be the vertex functions of f and g respectively at v (we will specify g_v below). Suppose both f and g have been fed the same input, and let $x_1, \ldots, x_d \in [0,1]$ and $y_1, \ldots, y_d$ be the values fed to f_v and g_v respectively by forward propagation. For all $1 \leq i \leq d$, inductively assume that $|x_i - y_i| \leq C_i \gamma^{-\frac{m}{\xi}}$ with C_i depending on m, n and the graph structure (the base cases are clear, since if v is an input vertex, then $x_i = y_i$ for $1 \leq i \leq d$). Then

$$\begin{aligned}
&|f_v(x_1, \ldots, x_d) - g_v(y_1, \ldots, y_d)| \\
&\leq |f_v(x_1, \ldots, x_d) - f_v(y_1, \ldots, y_d)| + |f_v(y_1, \ldots, y_d) - g_v(y_1, \ldots, y_d)| \\
&\leq L(\max_i C_i)\gamma^{-\frac{m}{\xi}} + |f_v(y_1, \ldots, y_d) - g_v(y_1, \ldots, y_d)| && \text{By Lipshitz property and induction} \\
&\leq L(\max_i C_i)\gamma^{-\frac{m}{\xi}} + C'\gamma^{-\frac{m}{d}} && \text{Choose } g_v \text{ and } C' \text{ via Proposition 6} \\
&\leq (L(\max_i C_i) + C')\gamma^{-\frac{m}{\xi}}, && (2.40)
\end{aligned}$$

completing the inductive step.

Thus by induction, at the output of the final vertex, f and g will differ by $C\gamma^{-\frac{m}{\xi}}$ where C depends on m, n and the structure of the graph but not on γ, as desired. □

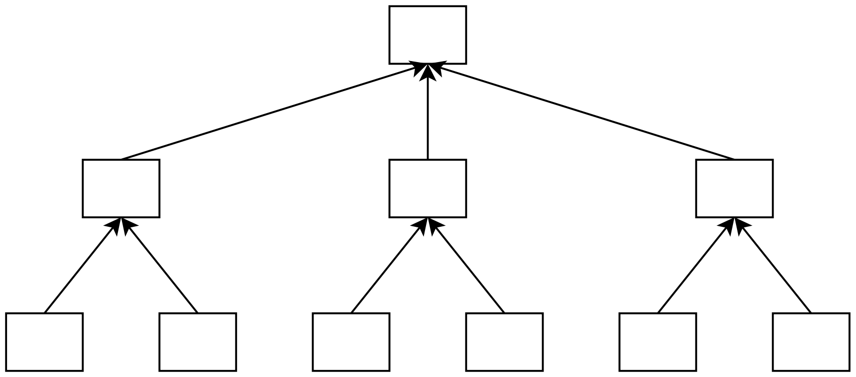

Figure 2.3: If the target function f conforms to the above graph in the sense of Definition 11, then it has input dimension 6, and so Proposition 6 guarantees only an $O(u^{-\frac{m}{6}})$ approximation using a shallow neural network. If we use a neural network conforming to the same graph structure, then Proposition 11 guarantees us an improved $O(u^{-\frac{m}{3}})$ approximation, since the graph has maximal in-degree of 3.

Note that if we take $u := (\#V)\gamma$ then clearly $\|f - g\|_\infty \leq C(\#V)^{\frac{m}{\xi}} u^{-\frac{m}{\xi}}$. Thus if we have a neural network whose structure conforms to the same graph as the true function we wish to estimate, then our order of approximation is significantly improved compared to what we could achieve with a shallow neural network, assuming $\xi \ll n$ (compare with Proposition 6. See also Figure 2.3). This is an appealing result, since it says that even if we have very high dimensional data, if the function we wish to approximate has compositional structure corresponding to a graph with limited maximal in-degree, we can still get a good order of approximation. It is natural to ask, however, what this tells us about neural networks in practice.

On the one hand, several aspects of Proposition 11 correspond to phenomena in real-world neural network approximation. First, in many contexts it is reasonable to expect that that the functions we wish

[10] This follows since if f_v is the vertex function for a vertex v with $d = \operatorname{indim}(v)$, then $|f_v(x) - f_v(y)| = |\langle \nabla f_v(\theta), y - x\rangle| \leq \|\nabla f_v(\theta)\|_1 \|y - x\|_\infty \leq \xi \|y - x\|_\infty$ for some θ on the line between y and x.

to approximate will be highly compositional. In computer vision, pixels combine to form edges, which form contours, which form objects, which are parts of larger objects, etc. In audio analysis, the compositional structure is less clear, but the author of [Bre94] has persuasively argued that local time/frequency patches combine to form higher level auditory scene descriptors. Second, convolutional neural networks, the most successful neural networks developed to date, consist of stacked layers of convolutions with very small kernels. These small kernels correspond to a graph with a small maximal in-degree. Hence it is possible that part of the performance of convolutional neural networks can be explained by the behavior described by Proposition 11.

On the other hand, Proposition 11 requires the graph structure of the function to be approximated to exactly match that of the approximating neural network. As Mhaskar and Poggio[MP16] note, Proposition 11 still holds if the graph of the approximating networks contains the graph of the target function as a subgraph. This however, is insufficient. What is really needed is a result describing the ability of neural networks to approximate functions with compositional structure similar to but not exactly matching their own. Such a result remains to be discovered by cunning theoreticians.

2.5 Convolutional Neural Networks

Convolutional neural networks (ConvNets) are the most popular and powerful form of neural network in use today. Like much of the key technology in the field of deep learning, they were developed in the 1980s[LCJB+89], but their popularity spiked when a convolutional neural network implemented on a pair of GPUs won the 2012 ImageNet image recognition challenge[KSH12]. Broadly, a ConvNet is a type of computation graph in which most vertexes receive as input and produce as output a tensor, and in which a form of locality and spatial invariance is enforced on the vertex functions. For the purposes of this essay, a tensor is an n-dimensional array of real numbers[KB09]. To simplify the explanation, we describe ConvNets here in the case of two-dimensional convolutions applied to three-dimensional tensors, but their extension to arbitrary dimension is simple. We begin by defining the convolutional layer, ConvNets' key vertex type. Table 2.1 provides a reference for the meaning of symbols pertaining to convolutional layers.

Definition 12. Let v be a vertex in a computation graph and let $\sigma : \mathbb{R} \to \mathbb{R}$. Call v a *convolutional layer* with activation function σ if

1. There exist $d_i, d_o, W, L \in \mathbb{Z}_{++}$ such that v has vertex domain $\mathbb{R}^{d_i \times L \times W}$ and vertex range $\mathbb{R}^{d_o \times L \times W}$.

2. There exist odd positive integers k_W, k_L and tensors $\{T^1, \dots, T^{d_o}\} \subset \mathbb{R}^{d_i \times k_w \times k_L}$ and a bias vector $b \in \mathbb{R}^{d_o}$ such that the vertex function $c_v : \mathbb{R}^{d_i \times L \times W} \to \mathbb{R}^{d_o \times L \times W}$ is specified by

$$\begin{aligned} \Xi_v(A)_{i,j_w,j_l} &:= \sum_{p,q_w,q_l \in \mathbb{Z}} T^i_{p,q_w,q_l} A_{p,j_w-q_w,j_l-q_l} + b_i \\ c_v(A)_{i,j_w,j_l} &:= \sigma\left(\Xi_v(A)_{i,j_w,j_l}\right) \end{aligned} \tag{2.41}$$

   where we use zero-padding boundary conditions for the tensors A and $\{T^i\}_{i=1}^{d_o}$.

Although (2.41) is complicated, the underlying idea is simple: we convolve an input tensor with a kernel, then apply the activation function entrywise. Figure 2.4 illustrates a simple example.

Because of their practical utility, many variant ConvNet architectures have been developed, some of which include special layers types. See Section C for a description of some of these layer types.

2.5.1 ConvNet Definition

Definition 13. Let $\sigma : \mathbb{R} \to \mathbb{R}$. A computation graph is called a *convolutional neural network* with activation function σ if all its vertexes are either (a) convolution layers with activation function σ or the identity, or (b) one of the layer types of Section C.

Symbol	Meaning
d_i	input channels
d_o	output channels
W	input width
L	input length
k_w	width kernel size
k_l	length kernel size
b	bias vector
$\{T^1, \dots, T^{d_o}\}$	weight tensors

Table 2.1: Symbols in definition of convolutional layer

Remark 7.

1. Given a tensor $A \in \mathbb{R}^{d \times W \times L}$, the matrices $A_{1,:,:}, \dots, A_{d,:,:}$ (using MATLAB notation) are called the *channels* of A. Each channel of the output tensor of a convolutional layer is the result of applying a separate convolution to the input tensor.

2. The operation described in (2.41) can be applied to N input tensors simultaneously via the multiplication of a $d_o \times d_i k_l k_w$ matrix by a $d_i k_l k_w \times NLW$ matrix[CWV$^+$14, Section 3][11]. Because GPUs specialize in tasks which like the multiplication of large dense matrices involve a high ratio of floating point operations to memory accesses, convolutional layers are efficiently implementable on GPUs.

3. Originally designed for image processing, ConvNets are now used to solve diverse problems from numerous fields. For example, in machine translation, convolutional neural networks have begun to out-perform recurrent neural networks, which were previously believed to be optimal for sequence data[GAG$^+$17].

4. Convolutional layers enforce spatial invariance by applying the same transformation to every patch of the input tensor. Besides decreasing the number of parameters that need to be learned, this invariance also serves as a reasonable constraint for many problems of interest. The clearest example comes from image processing, in which displacing an object in an image does not affect what the object is.

5. Although the theory of deep ConvNets is less well developed than the theory of general deep neural networks described in Section 2.4, we can still draw from the general theory for insight about the behaviour of ConvNets. For instance, we could argue using the effect described in Section 2.4.1 that deep piecewise linear ConvNets can have more linear pieces than shallow ones. Such an argument would need to take into account that convolutional layers can only represent convolutions, not arbitrary linear transformations, but I do not expect this would prove a fudamental impediment.

   We could also draw on theory of Section 2.4.2, arguing that ConvNets resemble the hierarchical neural networks of that section. If k_l and k_w are small, then each entry of an output tensor produced by a convolutional layer depends on only a small number of entries in the input tensor. Hence any function induced by a convolutional neural network can be rewritten in the form of Section 2.4.2 where all vertexes have small indegree. The correspondence between convolutional and hierarchical neural networks is not perfect, as general ConvNets do not allow an arbitrarily wide shallow network to operate on each patch of the input tensor as required by Proposition 11. However, a ConvNet in which many different convolutional layers operate on the same input tensor and the output tensors

[11]The second matrix is usually formed lazily to conserve memory.

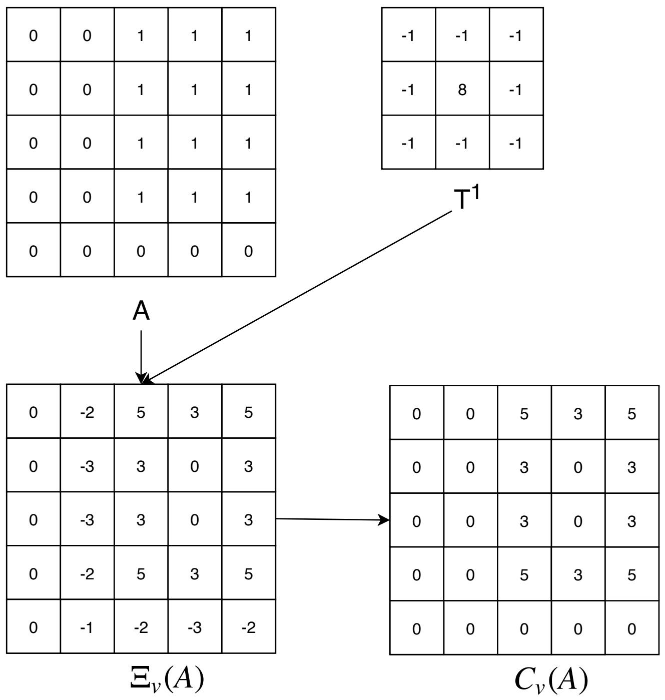

Figure 2.4: The action of a convolution layer on an input tensor A. The convolutional layer has $d_i = d_o = 1$, $W = L = 5$, $k_w = k_l = 3$, $b = 0$, and activation function $\text{ReLU}(x) := \max(x, 0)$.

produced by these layers are combined with a sum layer (see Section C) can more closely mimic the effect described in Section 2.4.2.

Definition 14. A *ConvNet architecture* is a directed acyclic graph (V, E) together with an assignment of a layer type (see Definition 12 and Section C) to each vertex $v \in V$, and a specification of all parameters of each layer except for the weight tensors of convolutional layers and the weight matricies of fully-connected layers.

Thus, a ConvNet architecture generates a family of ConvNets, each of which is specified by selecting weight tensors for each each convolutional layer and a weight matrix for each fully-connected layer (here we include the bias vector b as a weight tensor).

Example 4. As an example, consider the variant of the LENet architecture[LBBH98] shown in Figure 2.5. The architecture has domain $\mathbb{R}^{I_L \times I_w}$ and range $\mathbb{R}^D$, and consists of 3 convolutional layers followed by a flattening layer, a fully connected layer, and a softmax layer (see Section C). To get a convolutional neural network from the architecture, one must specify the weight tensors for the convolutional layers and the weight matrix for the fully connected layer.

2.6 Machine Learning

So far we have described neural networks as tools for approximating known functions. In most practical problems we do not know the function we wish to approximate, but have a limited number of observations

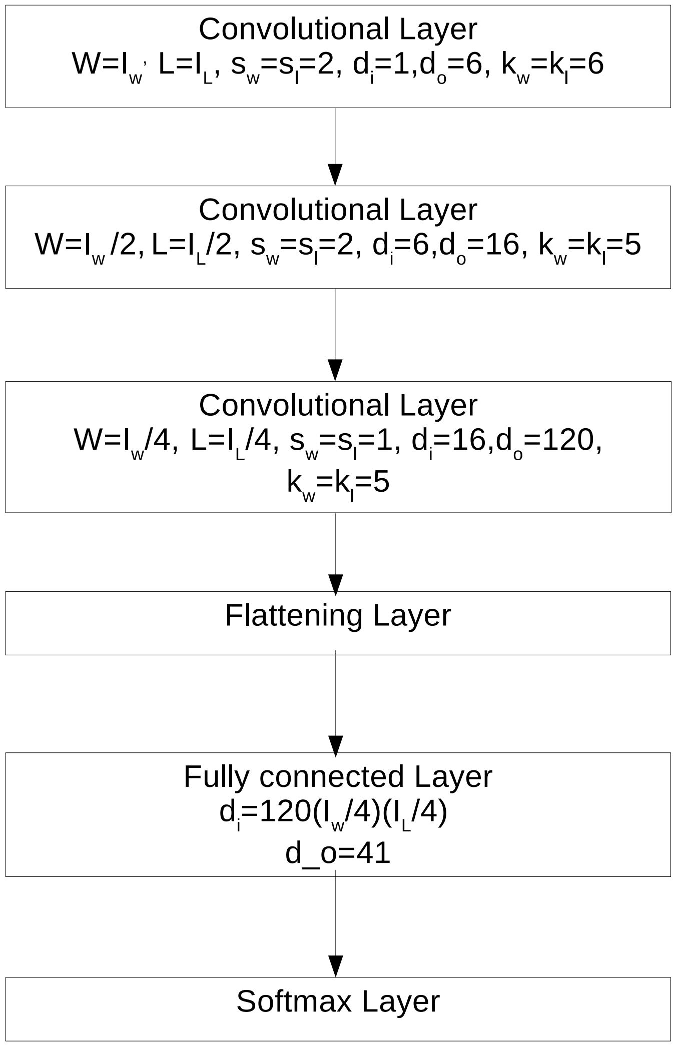
Convolutional Layer
W=I_w, L=I_L, s_w=s_l=2, d_i=1,d_o=6, k_w=k_l=6
Convolutional Layer
W=I_w/2, L=I_L/2, s_w=s_l=2, d_i=6,d_o=16, k_w=k_l=5
Convolutional Layer
W=I_w/4, L=I_L/4, s_w=s_l=1, d_i=16,d_o=120, k_w=k_l=5
Flattening Layer
Fully connected Layer
d_i=120(I_w/4)(I_L/4)
d_o=41
Softmax Layer

Figure 2.5: Modified LENet architecture. See Section C and Section 2.5 for layer types.

of its input and output. The machine learning framework was built to describe such problems. In Section 2.6.1 we outline the machine learning framework, and then in Section 2.6.2 we explain how it can be combined with the convolutional neural networks of Section 2.5.

2.6.1 The Machine Learning Framework

Since the machine learning framework is abstract, we begin with a concrete example.

Example 5. A medical scientist has measured the concentrations of various lymphocyte (white blood cell) types for each patient in a cohort of patients with family history of a certain autoimmune disease. Let $\mathcal{X} := \mathbb{R}^n_+$ denote the space of possible measurements, where for $x \in \mathcal{X}$, x_i is the concentration of the ith lymphocyte type. The medical scientist also recorded which patients ended up getting the disease. Let $\mathcal{Y} = \{1, 2\}$ be the set of possible patient states, where 1 indicates that a patient has the disease and 2 indicates that they do not. Furthermore let $\mathcal{S} \subset \mathcal{X} \times \mathcal{Y}$ be the finite set of measurement-state pairs found in the scientist's cohort data. $\mathcal{S}$ is called the training set. The scientist wishes to use this data to predict in advance which new asymptomatic patients will fall ill. Prior research has determined that the probability that a patient with a faimly history of the disease and a lymphocyte profile $x \in \mathcal{X}$ will get the disease is given by the probability rule $p_1(x; (w, b)) = \frac{\exp(\langle x,w\rangle+b)}{1+\exp(\langle x,w\rangle+b)}$ for some $w \in \mathbb{R}^n, b \in \mathbb{R}$, and correspondingly the probability that they will not get the disease is $p_2(x; (w, b)) := 1 - p_1(x; (w, b))$. Thus the possible probability rules are parameterized by $\mathcal{H} := \{(w, b) |\ w \in \mathbb{R}^n,\ b \in \mathbb{R}\}$. The scientist's problem is to determine the correct $(w^*, b^*) \in \mathcal{H}$. Let $\overline{\mathcal{S}} \supset \mathcal{S}$ denote the set of all patients in the world with family history of the disease. Conceptually, according to the statistical principle of maximum likelihood, it might be reasonable for the scientist to expect the correct $(w^*, b^*) \in \mathcal{H}$ to maximize the joint probability of $\overline{\mathcal{S}}$, which is $\prod_{(x,y)\in\overline{\mathcal{S}}} p_y(x; (w, b))$. The scientist does not have access to $\overline{\mathcal{S}}$, and so she must make do with the training set $\mathcal{S}$, finding (w^*, b^*) by maximizing $\prod_{(x,y)\in\mathcal{S}} p_y(x; (w, b))$, or equivalently minimizing $\sum_{(x,y)\in\mathcal{S}} -\log(p_y(x; (w, b)))$.

We now introduce the abstract framework. Following [SSBD14, chapter 2-3], a machine learning labeling problem consists of (i) a domain set $\mathcal{X}$, all possible objects the algorithm may need to label; (ii) a label set $\mathcal{Y}$, all possible labels; (iii) a hypothesis class $\mathcal{H}$, a parameterization of the valid labeling rules; (iv) a training set $\mathcal{S}$, a finite set of pairs[12] $\{(x_i, y_i)\}$ from $\mathcal{X} \times \mathcal{Y}$; (v) a loss function $l : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$, where $l(h, x, y)$ measures how undesirable it is to use labeling rule corresponding to h on an object x whose true label is y. We seek an $h^* \in \mathcal{H}$ that will perform well according to l not only on the training set $\mathcal{S}$, but on some yet-unseen data.

Symbol	Meaning
$\mathcal{X}$	domain set
$\mathcal{Y}$	label set
$\mathcal{H}$	hypothesis class
$\mathcal{S}$	training data or training set
l	loss function
R	regularizer

Table 2.2: Symbols in the machine learning framework

The most common rule for selecting $h^* \in \mathcal{H}$ is

$$h^* \in \underset{h\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{\#\mathcal{S}} \sum_{(x,y)\in\mathcal{S}} l(h, x, y). \tag{2.42}$$

That is, we choose a hypothesis that yields the lowest average loss on the training data.

When applying the machine learning framework, we must choose $\mathcal{H}$ and l such that solving (2.42) is both (a) tractable, and (b) yields an h^* such that $L(h^*, x, y)$ is low for pairs $(x, y) \notin \mathcal{S}$ we are likely to encounter.

Example 5 (continuing from p. 21)**.** In the scientist's example, $l((w, b), x, y) := -\log(p_y(x; (w, b)))$ up to a scaling factor.

[12]The individual $(x, y) \in \mathcal{S}$ are called training examples or datapoints.

2.6.2 Machine Learning with Convolutional Neural Networks

In this section we narrow our focus to machine learning problems for which the domain set $\mathcal{X}$ consists of tensors: $\mathcal{X} \subset \mathbb{R}^{d\times W\times L}$. We can generate a hypothesis class for such a problem using a ConvNet architecture with domain $\mathbb{R}^{d\times W\times L}$ (see Definition 14) by letting the hypotheses be assignments of weight tensors and weight matrices to the convolutional and fully connected layers of the architecture. Each such hypothesis $h \in \mathcal{H}$ converts the ConvNet architecture into a ConvNet and induces a function f_h. Since our intent is to label elements of $\mathcal{X}$ with labels from $\mathcal{Y}$, we consider only architectures whose last layer is a $\#\mathcal{Y}$-dimensional softmax (see Definition C.7). Such architectures have range $\Delta_+^{\mathcal{Y}}$, the positive probability distributions over $\mathcal{Y}$.

Example 6. A historian has recently obtained a cache of handwritten historical documents. The historian wishes to identify which of these documents were written by a particularly intriguing viscount. The historian decides to apply machine learning with convolutional neural networks. Having been apprised of Yann LeCun's famous work on handwritten digit recognition, the historian decides to use LENet as his convolutional architecture (see Example 4 and Figure 2.5). He will feed the architecture greyscale images of the documents, so that the domain set is $\mathcal{X} := \mathbb{R}^{W\times L}$ where W and L are the pixel dimensions of the images. As the historian is only interested in whether or not each document was written by the viscount, we have that the label set is $\mathcal{Y} := \{1, 2\}$ where 1 indicates the document was written by the viscount and 2 indicates that it was not. The historian spends his research grant paying graduate students to painstakingly analyze a subset of the documents by hand. The product of this labour is $\mathcal{S} \subset \mathcal{X} \times \mathcal{Y}$, a training set consisting of labeled documents. Each hypothesis in the hypothesis class $\mathcal{H}$ is a choice of weight tensors for the convolutional layers and a weight matrix for the fully connected layer of LENet. Since the LENet architecture ends with a softmax (see Definition C.7), the convolutional neural networks associated with the hypothesis class map each greyscale image of a document to a probability distribution over whether or not that document was written by the viscount.

As we saw in Example 5, one approach to devising a loss function function is via the statistical principle of maximum likelihood. Because every hypothesis h is associated with a function $f_h : \mathcal{X} \to \Delta_+^{\mathcal{Y}}$ mapping tensors to probability distributions over labels, we can compute the probability according to any $h \in \mathcal{H}$ of observing the training data, under the assumption that the labels of the training examples are drawn independently. This probability is

$$\prod_{(x,y)\in\mathcal{S}} (f_h(x))_y. \tag{2.43}$$

The maximum likelihood principle says that we should select the hypothesis according to which the training set $\mathcal{S}$ is most likely. That is, we should chose h^* to maximize (2.43) or, equivalently, choose h^* satisfying

$$h^* \in \operatorname*{argmin}_h \sum_{(x,y)\in\mathcal{S}} -\log(f_h(x))_y. \tag{2.44}$$

The form of (2.44) suggests the loss function

$$l(h, x, y) = -\log(f_h(x))_y. \tag{2.45}$$

Chapter 3

Optimization for Practical Convolutional Neural Networks

3.1 Introduction

This section discusses optimization for the application of convolutional neural networks to practical machine learning problems.

3.2 Optimization Algorithms

According to Section 2.6.2, we should select a hypothesis $h^* \in \mathcal{H}$ solving

$$h^* \in \operatorname*{argmin}_h \frac{1}{\#\mathcal{S}} \sum_{(x,y)\in\mathcal{S}} l(h,x,y), \tag{3.1}$$

where each $h \in \mathcal{H}$ is an assignment of weight tensors and matrices to a ConvNet architecture (see Definition 14). For convenience, in this section we flatten[1] all these tensors and matrices so that hypotheses are column vectors: $\mathcal{H} \subset \mathbb{R}^{d_H}$ for some d_H. A standard optimization approach to (3.1) is gradient descent[2], according to which we select some initial guess h^0, and then in each iteration i update

$$h^i = h^{i-1} - \frac{1}{\#\mathcal{S}} \alpha^i \nabla_h \sum_{(x,y)\in\mathcal{S}} l(h,x,y) \tag{3.2}$$

for learning rates $\{\alpha^i\}_{i=1}^{\infty} \subset \mathbb{R}_{++}$. This process of iteratively searching for the optimal $h \in \mathcal{H}$ is called *training.*

In many machine learning problems, $\#\mathcal{S}$ is very large, and consequently gradient descent is slow, since $\nabla_h l(h,x,y)$ needs to be computed for each $(x,y) \in \mathcal{S}$ in every iteration. To address this issue, researchers developed variants of gradient descent known as *stochastic gradient descent* (SGD) and *minibatch gradient descent* (MGD). In iteration i of SGD, we sample $(x^i, y^i) \in \mathcal{S}$ uniformly at random and update

$$h^i = h^{i-1} - \alpha^i \nabla_h l(h, x^i, y^i). \tag{3.3}$$

[1]To flatten a tensor means to rearrange its entries into a column vector. That is, we replace elements of $\mathbb{R}^{d_1 \times d_2 \times \dots \times d_n}$ with elements of $\mathbb{R}^{d_1 d_2 \dots d_n}$ (for some $d_1, d_2, \dots, d_n$).

[2]Several popular activation functions like ReLU are not differentiable at certain isolated points. One generally replaces the derivative with its left or right limit in these cases.

In iteration i of MGD, we sample $S_i \subset \mathcal{S}$ of fixed sized $Q := \#S_i$ uniformly at random and update

$$h^i = h^{i-1} - \frac{1}{Q}\alpha^i \sum_{(x,y)\in S_i} \nabla_h l(h,x,y). \tag{3.4}$$

Remark 8. For many machine learning problems, SGD and MGD with small Q empirically require less time and computation than gradient descent to achieve the same accuracy, and are preferred. There is no single definitive explanation for this superior performance. Here are a few arguments.

1. Typical machine learning training sets contain many very similar training examples. Gradient descent requires computing $\nabla_h l(h,x,y)$ for each of these examples each iteration, and is thus highly redundant. In contrast with SGD and MGD only a relatively small number of evaluations of $\nabla_h \log(f_h(x))_y$ are performed before h is changed, limiting redundancy.

2. [BCN18, Theorem 4.7] theoretically compares the rate of convergence of SGD and gradient descent when l is strongly convex in h. While strong convexity is a wildly unrealistic assumption for a neural network, if the network has a smooth activation function and a positive definite Hessian at a local minimum, it is strongly convex in a neighborhood of that minimum. Thus the rate of convergence analysis of [BCN18] applies in such a neighbourhood. The authors of [BCN18] show that under their assumptions, stochastic gradient descent requires a number of iterations of order $\frac{1}{\epsilon}$ to achieve ϵ optimality. In contrast, gradient descent requires order $\log(\frac{1}{\epsilon})$ iterations[B$^+$15, Theorem 3.12]. Since each iteration of gradient descent involves computing the gradient at $\#\mathcal{S}$ training datapoints, the ratio of the work required to achieve ϵ-optimality with gradient descent to the work required to achieve the same optimality with SGD is $(\#\mathcal{S})\epsilon\log(\frac{1}{\epsilon})$. This ratio favors gradient descent for small ϵ, but favors SGD in the when ϵ is moderate and $\#\mathcal{S}$ is very large. This latter situation in common in machine learning.

Remark 9. Some additional comments:

1. The above comparison of gradient descent with SGD also applies to MGD when $Q \ll \#S$. Researchers tend to prefer MGD to SGD because it offers greater opportunities for parallelization[3].

2. It is common[GBCB16, pg. 293] to replace (3.4) with

$$v^i = \rho v^{i-1} + \frac{1}{Q}\sum_{(x,y)\in S_i} \nabla_h l(h,x,y) \tag{3.5}$$
$$h^i = h^{i-1} - \alpha^i v^i, \tag{3.6}$$

where $0 < \rho < 1$. This procedure is known as mini-batch gradient descent with momentum. It is intuitively justified by the observation that it tends to amplify the change in $\{h^i\}$ in a direction if many recent gradients agree about the value of a direction, while it tends to dampen the change in $\{h^i\}$ in a direction if recent gradients disagree. Theoretical arguments for the benefits of momentum are scare, but it often speeds convergence in practice.

3.3 Batch Normalization

Nearly all modern convolutional neural networks use a very effective heuristic called batch normalization[IS15]. We will explain batch normalization in the context of minibatch gradient descent (see Section 3.2) applied

[3]MGD is also more stable than SGD, but SGD's stability can be improved by appropriately scaling down the learning rates relative to MGD.

to a convolutional neural network. Let the minibatch size be Q, and consider a convolutional layer accepting input tensors with d_i channels, width W, and length L (see Section 2.5). Let $A^1, \ldots, A^Q \in \mathbb{R}^{d_i \times L \times W}$ be the Q input tensors passed to the convolutional layer by forward propagation (see Section 2.1) as the ConvNet is applied to the Q datapoints in a minibatch. Define the statistics $\mu, \lambda^2 \in \mathbb{R}^{d_i}$ by

$$\mu_k = \frac{1}{WLQ} \sum_{q=1}^{Q} \sum_{j_w=1}^{W} \sum_{j_l=1}^{L} A^q_{k,j_w,j_l} \tag{3.7}$$

$$\lambda_k^2 = \frac{1}{WLQ} \sum_{q=1}^{Q} \sum_{j_w=1}^{W} \sum_{j_l=1}^{L} (A^q_{k,j_w,j_l} - \mu_i)^2. \tag{3.8}$$

To apply batch normalization to the convolutional layer, we normalize its input using these minibatch statistics. Instead of directly passing $\{A^1, \ldots, A^Q\}$ to the convolutional layer, we instead pass $\{(A^1)', \ldots, (A^Q)'\}$ defined by

$$(A^q)'_{k,:,:} = \alpha_k (A^q_{k,:,:} - \mu_k)/\lambda_k + \beta_k, \tag{3.9}$$

where we have used MATLAB notation, and where $\alpha, \beta \in \mathbb{R}^{d_i}$ are new trainable parameters (thus we expand the hypothesis class by introducing batch normalization). In essence we are normalizing the input tensors to the convolutional layer so that each channel has mean 0 and variance 1 over the minibatch, and then applying a learnable affine transformation specified by α and β. When training is complete and the ConvNet must be applied to test data, μ and λ^2 can be fixed at a weighted average of their values over the last iterations of training. Batch normalization violates the formal framework described previously, because it introduces dependencies between the ConvNet's output at different datapoints in a minibatch. Nevertheless, it is very effective[4].

Despite its effectiveness, batch normalization is poorly understood. One popular theory of is that of "reducing internal covariate shift", which claims that batch normalization improves optimization by ensuring that changes in the weights of earlier layers during optimization do not alter the input distribution seen by subsequent layers too dramatically. According to this theory, batch normalization makes the training of different convolutional layers more independent by controlling the distribution of their inputs. The authors of a recent working paper, [STIM18], present experimental evidence contradicting the internal covariate shift theory. Some of their experiments even suggest that the input distribution seen by convolutional layers during optimization varies more with batch normalization than without. The authors of [STIM18] use further experiments and theoretical analysis of simplified examples to claim that batch normalization actually works by making the gradient of the loss function more Lipshitz, and thus more informative to the optimization algorithm. If this claim is borne out by future work, it may finally provide a foundation for a deeper understanding of batch normalization.

[4]Examples abound. The original batch normalization paper[IS15] demonstrates on standard image classification benchmarks that convolutional neural networks with batch normalization achieve a higher maximum accuracy and require less training time than ConvNets without batch normalization. Also, batch normalization is a key component of the celebrated ResNet[HZRS16].

Chapter 4

Audio Processing

4.1 Introduction

I participated in the 2018 DCASE Freesound General-Purpose Audio Tagging Challenge[FPF+18] hosted on Kaggle. Participants were asked to train an algorithm on a training set of labeled sound clips so that it could accurately label previously unseen sound clips. A standard deep learning approach to this problem is to apply a convolutional neural network to the log spectrograms of the sound clips. I adopted this approach. In this section I justify and describe the short-time Fourier transform, from which the log spectrogram is derived.

4.2 How the Ear Processes Sound

Sounds are pressure waves. When a pressure wave reaches a human ear, it causes the eardrum to vibrate. This vibration is transmitted via a sequence of bones to the cochlea, where it propagates through the cochlear fluid to a structure known as the basilar membrane, different parts of which are sensitive to different frequencies. The frequency-dependent movement of the basilar membrane is transmitted to hair cells, which convert it to electrical impulses, which eventually reach the brain (for more, see Chapter 2 of [SNK11]). This time-varying decomposition of sound into constituent frequencies is the key characteristic of the human auditory system. Thus if we hope to develop an audio classification system partly inspired by how humans process sound, time-frequency decomposition is a good starting point.

4.3 The Short-Time Fourier Transform

The fundamental time-frequency decomposition is the short-time Fourier transform (STFT), which we introduce in this section. Before proceeding, we establish additional notation.

In this section, we will use brackets instead of subscripts to refer to components of vectors in $\mathbb{C}^n$ and matrices in $\mathbb{C}^{n\times n}$. We use the usual inner product: if $f_1, f_2 \in \mathbb{C}^N$ then $\langle f_1, f_2\rangle = \sum_{j=0}^{N-1} f_1(j)\overline{f_2(j)}$ and if $s_1, s_2 \in \mathbb{C}^{N\times N}$ then $\langle s_1, s_2\rangle = \sum_{k=0}^{N-1}\sum_{j=0}^{N-1} s_1(k,j)\overline{s_2(k,j)}$ where $\overline{x}$ denotes the complex conjugate of x. Use the usual norm $\|f\| = \sqrt{\langle f, f\rangle}$. Use periodic boundary conditions, so that for $f \in \mathbb{C}^N$, we have $f(x + jN) := f(x)$ for all $j \in \mathbb{Z}$. For $f_1, f_2 \in \mathbb{C}^N$, define their convolution by $(f_1 * f_2)[t] = \sum_{t'=0}^{N-1} f_1(t')f_2(t-t')$. Let $i \in \mathbb{C}$ denote the imaginary unit. Let $\phi^0, \ldots, \phi^{N-1} \in \mathbb{C}^N$ denote the standard Fourier basis: $\phi^j(k) = \frac{1}{\sqrt{N}}\exp(2\pi i\frac{jk}{N})$. Define the Fourier transform $\mathcal{F} : \mathbb{C}^N \to \mathbb{C}^N$ by

$$\mathcal{F}[f](j) = \langle f, \phi^j\rangle. \tag{4.1}$$

We have Parseval's formula: for all $f_1, f_2 \in \mathbb{C}^N$, $\langle \mathcal{F}(f_1), \mathcal{F}(f_2)\rangle = \langle f_1, f_2\rangle$. For $t, \omega \in \{0, \dots, N-1\}$ define the translation operator T by $T_t[f](j) = f(j-t)$, and the modulation operator by $M_\omega[f](j) = \phi_\omega(j) f(j)$. Note that for $h \in \mathbb{C}^N$ we have

$$\mathcal{F}[T_t h](\omega) = \frac{1}{\sqrt{n}} \sum_{t'=0}^{N-1} h(t'-t) \exp(-2\pi i \frac{\omega}{N} t') = \frac{1}{\sqrt{n}} \sum_{\tau=0}^{N-1} h(\tau) \exp(-2\pi i \frac{\omega}{N}(\tau + t)) = \sqrt{n} M_{-t} \mathcal{F}[h](\omega) \tag{4.2}$$

where we used the change of variable $\tau = t' - t$. Analogously,

$$\mathcal{F}(M_\delta h)[\omega] = \frac{1}{n} \sum_{t'=0}^{N-1} \exp(2\pi i \frac{\delta}{N} t') h(t') \exp(-2\pi i \frac{\omega}{N} t') = \frac{1}{n} \sum_{t'=0}^{N-1} h(t') \exp(-2\pi i \frac{\omega - \delta}{N} t') = \frac{1}{\sqrt{n}} T_\delta \mathcal{F}[h](\omega). \tag{4.3}$$

We can now proceed to the short-time Fourier transform. Let $f \in \mathbb{C}^N$ be a time-domain signal consisting of N measurements of the amplitude of a sound, where consecutive measurements are separated by equal time intervals (in every practical case, $f \in \mathbb{R}^N$, but it is mathematically natural to use $\mathbb{C}^N$). We are interested in the frequency decomposition of f over short time periods. We shall use the window function $g \in \mathbb{C}^N$ to specify the time periods in which we are interested. Then the value of $S[f,g]$, the short-time Fourier transform of f with respect to g, at time and frequency $t, \omega \in \{0, \dots, n-1\}$ is defined by

$$S[f,g](t,\omega) := \frac{1}{\sqrt{N}} \sum_{t'=0}^{N-1} f(t') \overline{g(t'-t)} \exp(-2\pi i \frac{\omega}{N} t'). \tag{4.4}$$

Thus $S[f,g] \in \mathbb{C}^{N \times N}$. To get some insight into the STFT, we establish Proposition 12 .

Proposition 12 ([Grö13, Lemma 3.1.1])**.** *For $f, g \in \mathbb{C}^N$ and $t, \omega \in \{0, \dots, N-1\}$ we have*

$$\begin{aligned} S[f,g](t,\omega) &= \mathcal{F}(f T_t \overline{g})(\omega) && (4.5)\\ &= \exp(-2\pi i \frac{\omega}{N} t)(f * M_\omega \breve{g})(t) && (4.6)\\ &= \langle f, M_\omega T_t g\rangle && (4.7)\\ &= \langle \mathcal{F}[f], T_\omega M_{-t} \mathcal{F}[g]\rangle && (4.8) \end{aligned}$$

where $\breve{g}(t) := \overline{g(-t)}$.

Proof. (4.5) and (4.7) are immediately clear from (4.4). (4.6) becomes obvious when the definitions of convolution and $\breve{g}$ are substituted. (4.8) follows from (4.7) when Parseval's formula, (4.2), and (4.3) are applied. □

Remark 10. (4.5), (4.6), and (4.7) are sometime called the "3 souls" of the short-time Fourier transform[EDM13]. They reveal three different ways to understand it. Assume that the window function g is real, symmetric about 0, nonnegative, and supported on a set of size much smaller than N. Then (4.5) provides a time viewpoint, showing that the short-time Fourier transform can be computed by decomposing the signal f into its constituent frequencies over each of the small time windows specified by pointwise multiplication by translations of g. (4.6) provides a frequency viewpoint, showing that the short-time Fourier transform can also be computed by convolving a signal f with the modulations of g. Finally, (4.7) exhibits a linear-algebraic interpretation of the short-time Fourier transform: one computes it by taking the inner product of the signal f with each of the translations and modulations of the window function g.

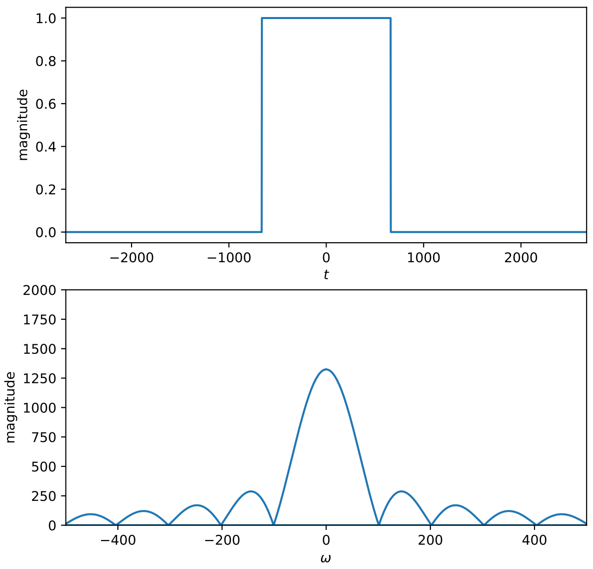

Figure 4.1: The boxcar window (top) and its Fourier transform (bottom). Normalization differs from text.

Remark 11. By (4.7), if g is concentrated around 0, then the value of $S[f, g](t, \omega)$ depends on the value of f only in a small neighborhood of t. (4.8) reveals the corresponding result on frequency concentration: if $\mathcal{F}[g]$ is concentrated around 0 then $S[f, g](t, \omega)$ depends only on $\mathcal{F}[f]$ in a small neighborhood of ω.

Thus the time and frequency resolution of the short-time Fourier transform depends on the time and frequency concentration of the windowing function. Our goal is to break a time-domain signal down into is frequency components over small time intervals. We should thus choose a window with small time support, but which is also relatively concentrated in frequency. An obvious first choice, the scaled characteristic function χ_I of a small interval (called the boxcar window) fails the second criteria (see Figure 4.1). The Hann window has better frequency localization, and thus is a better choice (see Figure 4.2). All of my models use the Hann window.

4.4 Practical Considerations

This section describes modifications to the short-time Fourier transform used in practical implementations.

4.4.1 The Sub-Sampled Short-Time Fourier Transform

Since $S[\cdot, g]$ is a linear transformation from $\mathbb{C}^N$ to $\mathbb{C}^{N \times N}$, the time-frequency grids it outputs are highly redundant. We can make the STFT more efficient by eliminating some this redundancy. Let g be supported on a interval of size $L < N$, and assume for convenience that L divides N. Take $\alpha \in \mathbb{Z}_{++}$ with $\alpha \leq L$. Then the sub-sampled short-time Fourier transform is defined by

$$S_{\alpha,L}[f, g](t, \omega) = \frac{1}{\sqrt{N}} \sum_{t'=0}^{N-1} f(t') \overline{g(t' - \alpha t)} \exp(-2\pi i \frac{\omega}{L} t') \tag{4.9}$$

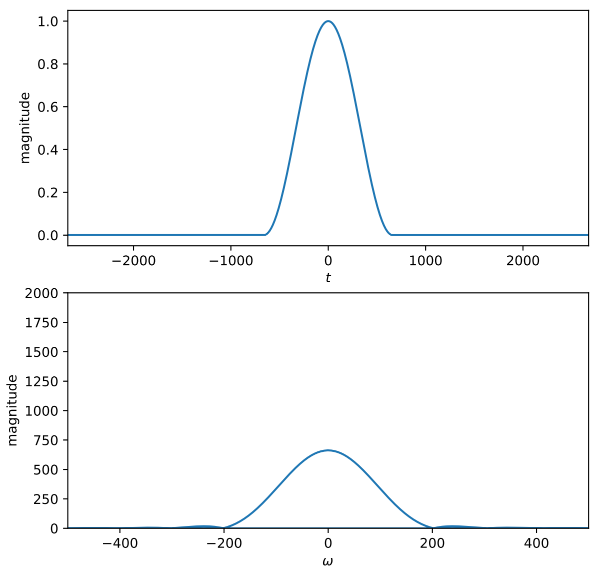

Figure 4.2: The Hann window (top) and its Fourier transform (bottom). Normalization differs from text.

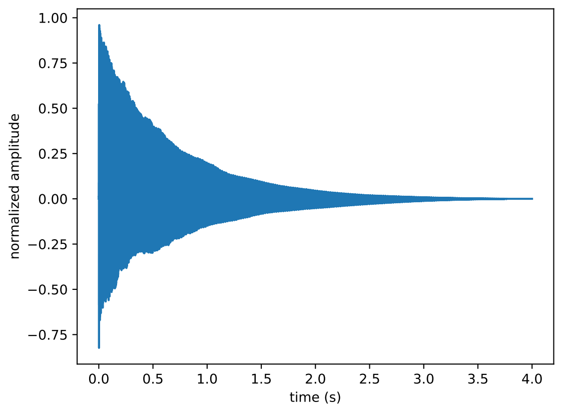

Figure 4.3: Raw amplitude plot of a guitar string pluck. Sample rate is 44.1 kHz.

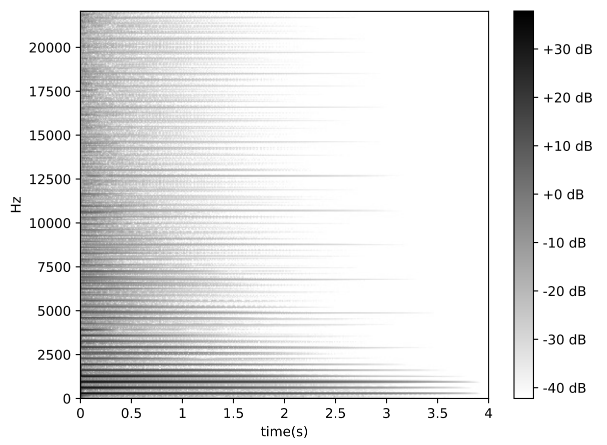

Figure 4.4: The magnitude short-time Fourier transform of a pluck of a guitar string. Sample rate is 44.1 kHz.

for $0 \leq t < \lceil \frac{N}{\alpha} \rceil$ and $0 \leq \omega \leq L$. This version of the sub-sampled short-time Fourier transform uses only a subset of the times and frequencies in the original STFT. Nevertheless, it is invertible, since it amounts to covering $\{0, \ldots, N-1\}$ with overlapping length L intervals and then storing the L-point Fourier transforms of f windowed by g on these intervals.

The contest audio is sampled at 44.1kHz. I use the sub-sampled STFT with $\alpha = 441$ (10ms) and $L = 1323$ (30 ms).

4.4.2 Further Memory Savings

When the Fourier transform is applied to a real signal $f \in \mathbb{R}^n$, half of the output components can be discarded, since

$$\overline{\mathcal{F}[f]}(n-k) = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} \overline{f[j] \exp(-2\pi i j \frac{n-k}{n})} = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} f[j] \exp(2\pi i j) \exp(-2\pi i j \frac{k}{n}) = \mathcal{F}[f](k). \quad (4.10)$$

Since computing the short-time Fourier transform involves repeated computation of the Fourier transform of real signals, this technique can be used to save memory.

Also, experience has demonstrated that the most practically useful information is in magnitudes, not the phases of the short-time Fourier transform. Thus it is conventional to discard phase information. The magnitude of the short-time Fourier transform of a signal is known as its spectrogram.

To reduce the size of the data that must be processed, it is common to shrink the spectrogram of a sound clip by binning its frequencies. Instead of using bins of fixed frequency width, bins of fixed width in the mel scale are used. The mel scale is linear in frequency below a certain critical frequency (1kHz in the implementation I use[MRL+15]) and logarithmic in frequency above that critical frequency. The mel scale was designed so that pitch intervals perceived by humans to have equal width have approximately equal mel width. Typically, one can attain an approximately tenfold reduction in spectrogram size using this method.

4.4.3 Log Magnitude

Finally, human perception of loudness is modeled better by a logarithmic scale than by a linear scale. For this reason, my submission to the DCASE Audio Tagging Challenge makes use of the log spectrograms of the sound clips.

Chapter 5

Freesound General-Purpose Audio Tagging Challenge

Participants in the 2018 DCASE Freesound General-Purpose Audio Tagging Challenge[FPF$^+$18] were given a training dataset of 9474 sound clips ranging in duration from 0.3 seconds to 30 seconds (see Figure 5.1). Each training clip was labeled with one of 41 labels (see Section D). Participants aimed to train machine learning algorithms on the training dataset so that they performed well on held-out test clips. I submitted an ensemble of deep convolutional neural networks, which scored 13th of 558 submissions by individuals and teams. In this section, I explain my submission. See Figures 5.9 and 5.10 for overviews the training and testing pipelines (I describe the details of these pipelines in the subsequent sections of this chapter).

5.1 Why it is Reasonable to Apply Convolutional Neural Networks to Spectrograms

As mentioned in Section 4, my convolutional neural networks accepted as input the (log) spectrogram of the sound clips. One might ask why this is a reasonable approach, given that convolutional neural networks were originally designed for image processing. While part of the answer to this question is "because it works", there are also more satisfying explanations of why convolutional neural networks can be effectively applied to spectrograms.

Speaking generally, both image classification and sound classification are hard pattern recognition problems. To correctly label images of buses, an image recognition algorithm must identify different types of buses from different angles and with different levels of occlusion. Similarly, to correctly label the sounds of buses, a sound classification algorithm must identify the sounds of different types of buses mixed with various interfering background noises. Deep neural networks specialize in this type of problem, in which classification must be invariant to irrelevant transformations of the data (recall from Section 2.4.1 that deep neural networks can map many disjoint input regions to the same output).

More specifically, the rules by which the brain analyzes a visual scene in some ways resemble the rules by which it analyzes an auditory scene. [Bre94, Chapter 1] gives examples, which I repeat here. All else being equal, the brain tends to group nearby regions of an image as corresponding to the same object. An experiment demonstrates an analogous phenomon in auditory scene analysis of time-frequency space. In the experiment, the experimenters play the tones $H_1L_1H_2L_2H_3L_3H_1L_1H_2L_2H_3L_3\ldots$ where the H_i denote high frequency tones and the L_i low frequency tones. There are two ways the subjects can hear these tones: either as a single sequence alternating between high and low tones, or as two separate simultaneous sequences, one of high tones and one of low tones. In the later case, attention can

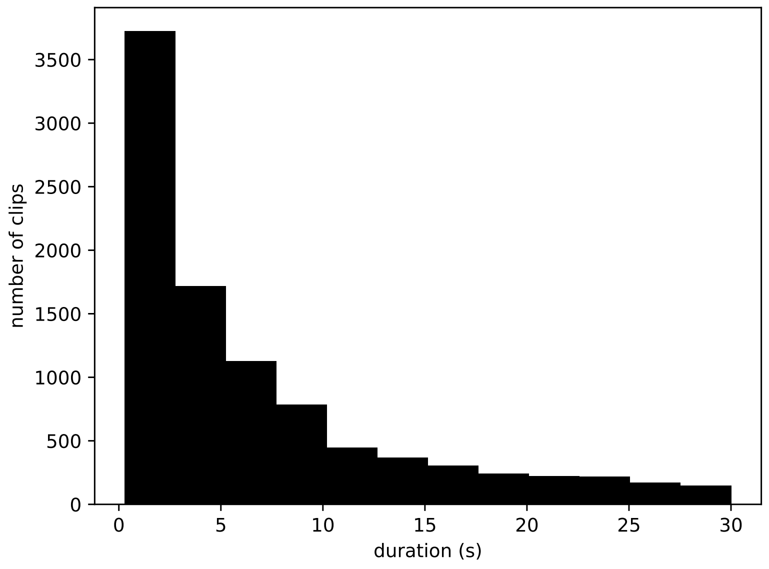

Figure 5.1: Durations of Clips in the training set for the 2018 DCASE Freesound General-Purpose Audio Tagging Challenge. Sample rate is 44.1 kHz.

only be focused on one sequence at a time, and the other is perceived as what the author of [Bre94] calls "vague background". Experimenters found that either increasing the speed of alternation or increasing the frequency separation between the high and low tones made listeners more likely to perceive two separate sequences (see Figure 5.2). This result reveals a tendency of the brain to group together sounds that are nearby in time-frequency space, just as it groups together nearby regions of an image.

Another example of a rule used by the brain in both auditory and visual scene analysis is that of masking. According to this rule, two disjoint regions are more likely to be perceptually grouped if there is evidence that intervening information has been lost. In the case of images, this rule manifests as the familiar logic of occlusion. In the case of auditory scenes, the rule can be demonstrated by experiment. If a listener is played two isolated sounds separated by a gap of silence, the listener is likely to perceive the sounds as separate. However, if the the gap of silence is filled with white noise that could mask a continuous connection between the isolated sounds, then the isolated sounds may be grouped together and perceived as a single auditory phenomenon. The listener may even incorrectly perceive that phenomenon continuing through the noise. See Figure 5.3. Thus the brain uses the same sort of reasoning about masking in its analysis of images and sounds.

One could list more similarities between auditory and visual scene analysis, and interested readers are referred to [Bre94]. Note that there are clearly also many differences between how the brain analyses auditory and visual information. Nevertheless, for our purposes, the existence of some important similarities is sufficient. If you believe that convolutional neural networks are an effective tool for image processing in part because they can learn some of the analysis rules applied to images by the human brain, then it is reasonable to believe that convolutional neural networks will be an effective tool for audio processing, since the brain applies some of the same analysis rules to time-frequency space.

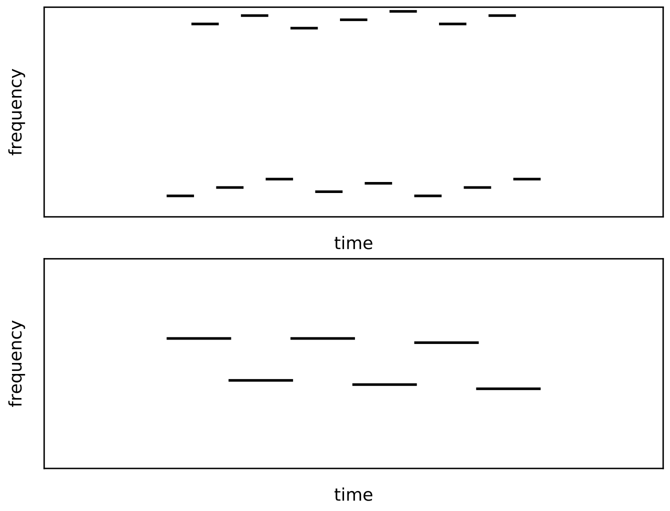

Figure 5.2: Upon hearing the sound described by the top diagram, listeners are likely to perceive two separate sequences of tones, one high and one low. Upon hearing the sound described by the bottom diagram, listeners are likely to perceive a single sequence of tones alternating between high and low.

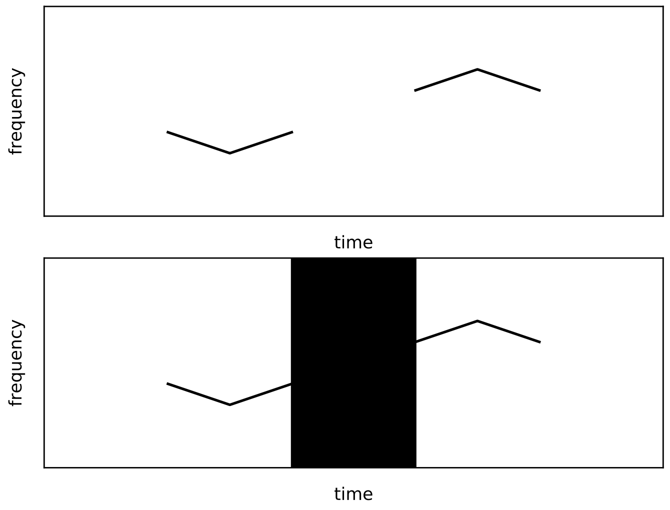

Figure 5.3: A diagram illustrating the masking phenomon. Upon hearing the sound described by the top diagram, the listener perceives two separated frequency glides. Hearing the sound described by the bottom diagram, he or she may perceive only one long frequency glide with some background noise in the middle.

5.2 Architectures

I used two main convolutional neural network architectures: ResNet[HZRS16] and Progressive NASNet[LZS$^+$17].

5.2.1 ResNet

The computation graph of a ResNet (Figure 5.4) consists of stacked subgraphs called residual units (Figure 5.5). A residual unit accepts a single tensor as input. That tensor is passed along two paths: a convolutional path and an identity path. The outputs of the convolutional and identity paths are added to produce the output of the residual unit. The convolutional path consists of series of convolutional layers with batch normalization (see Figure 5.6), while the the identity path merely performs the identity operation: its input is connected directly to its output.

ResNet and ResNet-like architectures have been ubiquitous since a team from Microsoft Research Asia used ResNets to win 1st place in the 2015 Imagenet classification, Imagenet localization, COCO detection, and COCO segmentation challenges[HZRS16]. Below I list a few of the various explanations for ResNet's performance that have been offered.

Identity

The authors of [HZRS16] originally motivated ResNets with the observation that increasing the depth of a convolutional neural architecture by adding stacked convolutional layers increases its performance up to a certain point, but adding too many convolutional layers actually worsens performance. This is surprising, since practical activation functions allow convolutional layers to learn the identity mapping[1]. The functions representable by n stacked convolutional layers are a subset of the functions representable by m stacked convolutional layers when when $m > n$, since the last $m - n$ layers can be set to the identity. That adding additional layers worsens performance suggests that the problem lies with the optimization algorithm, which fails to find the weights that would convert a convolutional layer to an identity mapping. The authors argued the use of residual units enables the optimization algorithm to more easily find the identity mapping, since a residual unit becomes an identity mapping if the weights of the final convolution in its convolutional path are zero.

Variable-Depth Ensembles

Let $\mathcal{M}^k$ represent the operation of the convolutional path of the kth residual unit in a residual network. For convenience, ignore the max layers in Figure 5.4. Let x^0 represent the input to the network and let $x^1, x^2, \ldots$ represent the output each of the residual units. Then we have

$$\begin{aligned}
x^1 &= \mathcal{M}^1(x^0) + x^0 \\
x^2 &= \mathcal{M}^2(x^1) + x^1 \\
&= \mathcal{M}^2(\mathcal{M}^1(x^0) + x^0) + \mathcal{M}^1(x^0) + x^0 \\
x^3 &= \mathcal{M}^3(x^2) + x^2 \\
&= \mathcal{M}^3(\mathcal{M}^2(\mathcal{M}^1(x^0) + x^0) + \mathcal{M}^1(x^0) + x^0) + \mathcal{M}^2(\mathcal{M}^1(x^0) + x^0) + \mathcal{M}^1(x^0) + x^0 \qquad (5.1) \\
&\vdots
\end{aligned}$$

Thus in a network of K residual units, the output of the Kth residual unit is the sum of the outputs of $K + 1$ convolutional networks of depths 0 to K, where the kth network shares all weights not in the kth convolutional path with the $(k-1)$th network. The authors of [WSH16] argue that these $K + 1$ convolutional networks can be thought of as being trained simultaneously and jointly, since the outputs of these networks are summed and fed to the final layers of the network, whose output is ultimately fed to the loss function. Furthermore they argue that during training the networks converge in sequence from shallowest to deepest: when the 1-unit network is still converging, the deeper networks cannot converge since their deeper units are receiving rapidly varying input. Once the 1-unit network has converged, the 2-unit network can converge, and so on. According to this viewpoint, the optimization algorithm can build

[1] Or at least the identity plus an irrelevant constant.

up a ResNet by constructing a series of progressively deeper subnetworks. In contrast, a non-residual network must be trained at full depth all at once. Perhaps relatedly, visualization techniques indicate that the objective function (3.1) tends to be much smoother for ResNets than for other convolutional networks [LXTG17].

5.2.2 Progressive NASNet

Recently, a research team at Google Brain had the idea of viewing the design of neural network architectures as itself an optimization problem to which deep learning could be applied[LZS+17, ZL16]. Broadly speaking, in each iteration of their architecture-design algorithm, a "controller" neural network specifies a distribution over the space of possible architectures for a target problem. Several sample architectures are drawn from this distribution. These sample architectures are trained on the target dataset, and then evaluated on a held-out portion of that dataset. At the end of the iteration, the parameters of the controller architecture are adjusted to increase its probability of producing the sample architectures that achieved the highest evaluation accuracy[2]. Because each iteration of the architecture-design algorithm requires numerous sample architectures to be trained to convergence, architecture search generally requires vast computational resources.

Progressive NASNet (NAS stands for Neural Architecture Search) is one of the most recent and most powerful automatically generated architectures produced by this research program. While the Progressive NASNet architecture was designed by a controller aiming to maximize accuracy on the CIFAR10[KH09] image dataset, experiments demonstrate that convolutional architectures with good performance on one benchmark tend to perform well on other tasks[KSL18], and thus I decided to try Progressive NASNet for the labeling of sound clip spectrograms.

Like ResNet, Progressive NASNet consists of stacked subgraphs called units (see Figures 5.7 and 5.8). Each unit accepts two input tensors and produces two output tensors. When down-sampling is necessary, strided convolutions are used (see Section C).

Progressive NASNet units are considerably more complicated than residual units. Observe in Figure 5.7, that while Output 1 is the result of a complicated series of layers applied to Inputs 1 and 2, Output 2 simply receives Input 1 and forwards it on. Thus like residual units, Progressive NASNet units can be interpreted as combining their own input with the input to the previous layer. Unlike with residual units, with Progressive NASNet units this reasoning can be applied only once, and cannot be repeated to argue that each units uses the input to all previous layers.

Perhaps the most salient feature of the Progressive NASNet unit is that it adds together the output tensors produced by convolutional layers of different kernel sizes, as well as the output tensors produced by max layers. This can be interpreted as an instance of the principle of combining information from different scales and different levels of detail. This principle has been used by a number successful hand-designed architectures (see [HCL+17] for example).

5.3 Activation Function

I used the leaky ReLU activation function given by $\mathrm{LReLU}(x) := \max(x, 0.01x)$. I also experimented with the Swish activation function, given by $\mathrm{SWISH}(x) = \frac{x}{1+\exp(-x)}$, a smooth approximation to ReLU. Recent research[RZL17] suggest convolutional neural networks using SWISH are easier to optimize than those using ReLU or LReLU, but this finding was not borne out in my tests (compare Rows 5.1.2 and 5.1.4 in Table 5.1).

[2]This is an instance of what is known as reinforcement learning.

5.4 Optimization

While many advanced, adaptive variants of minibatch gradient descent have been proposed[Zei12, KB14, TH12], plain minibatch gradient descent combined with momentum and a decreasing learning rate schedule (see Section 3.2) often achieves the state of the art on image-classification benchmarks[ZK16, HLVDMW17]. Unfortunately, the best learning rate schedule can be highly problem-dependent and difficult to determine. Loshchilov and Hutter[LH16] introduced what they call a cosine-annealing learning rate schedule, according to which the learning rate in step $0 \leq t < t_{\max}$ is given by

$$\alpha_t = \frac{1}{2}\alpha_0(1 + \cos(\frac{t}{t_{\max}}\pi)). \tag{5.2}$$

This method achieves performance comparable to that of finely tuned learning rate schedules, but requires only that one specify an initial learning rate α_0 and an anneal duration $t_{\max}$. Furthermore, Loshchilov and Hutter found that cyclically repeating the annealing process while using the the neural network found at $t = t_{\max}$ in one cycle as the initial guess at $t = 0$ in the next cycle produced further performance gains. For these reasons, I used minibatch gradient descent with Loshchilov and Hutter's cyclic cosine-annealing learning rate schedule.

5.5 Data Augmentation

The performance of a neural network depends significantly on the quantity of data upon which it is trained. When huge training datasets are unavailable, it is often beneficial to artificially expand a dataset by synthetically generating data. Typically, this is achieved by applying transformations to the datapoints in the training dataset. The transformations are chosen so that they do not change the correct label of the datapoints to which they are applied, and thus a synthetically generated datapoint can be given the same label as the original datapoint from which it was generated.

5.5.1 Pitch Shifting

The authors of [HCE$^+$17] investigated many data augmentation techniques for convolutional neural networks applied to spectrograms in the context of audio classification. Of the techniques they tried, shifting the pitch of the sound clips was by far the most effective. Consequently, I made use of pitch shifting augmentation. For each clip in a minibatch, with 50% probability I shifted its pitch by a number of semitones drawn uniformly from $\{-2, -1, 1, 2\}$. See Clips 1 for an example.

Comparing rows 5.1.1 and 5.1.2 in Table 5.1 reveals the power of pitch-shifting. The median accuracy of the basic ResNet model trained with pitch-shifting is about 1.5 percentage points higher than the median accuracy of same model without pitch-shifting. The 95% confidence intervals for these median accuracies (computed from 30 runs with and without pitch shifting using the interpolated sign test[GF93, BH93]) are $[0.8230, 0.8329]$ without pitch shifting and $[0.8384, 0.8459]$ with pitch shifting, indicating that there is a statistically significant difference[3].

Conceivably, pitch shifting could be a useful augmentation for only certain categories of sound clip. To test this, I calculated the per-category median accuracy of the basic ResNet model with and without pitch shifting over 30 runs. Table 5.2 shows that while the model trained with pitch shifting was better at classifying most categories, on certain categories its performance deteriorated. As a first attempt to correct this, I trained a model in which pitch shifting was not applied to those categories for which Table 5.2 showed a decrease in accuracy. The model trained by this partial augmentation had a slightly increased median accuracy (0.8467 vs 0.8426), but this was not statistically significant. Moreover, this naive approach does not account for the full complexity of identifying the best categories for augmentation. It might be that augmenting one category increases its accuracy at the cost of dramatically decreasing

[3]Note that these confidence intervals are different from the percentiles reported in Table 5.1.

the accuracy of a second category, so that an optimal augmentation scheme would not augment the first category at all. Future work could consider automatically exploring per-category augmentation schemes.

Guitar

Guitar Shifted 1 Semitone

Guitar Shifted 2 Semitones

Clips 1: When this essay is viewed with a recent version of Adobe Reader with the Adobe Flash Player plugin installed, double-clicking the above buttons plays the original and pitch-shifted versions of a training sound clip from the Guitar category of the DCASE Freesound General-Purpose Audio Tagging Challenge.

5.5.2 Mixup

The winners of the 2018 DCASE Freesound General-Purpose Audio Tagging Challenge[JL] used a special form of data augmentation called mixup[ZCDLP17]. Mixup is motivated by the observation that trained neural networks often exhibit erratic behaviour on artificial datapoints that are convex combinations of true datapoints. Mixup attempts to train a neural network to make sensible predictions on such artificial datapoints, in the hopes that this will result in more sensible behaviour overall. In more detail, whenever a minibatch is needed by minibatch gradient descent, each mixup datapoint in the minibatch is generated as follows. Two random datapoints (x_1, y_1) and (x_2, y_2) are drawn uniformly at random from the original training dataset, and a scalar $\lambda \in [0, 1]$ is drawn at random according to a Beta distribution. The idea is to generate an artificial datapoint that is a convex combination of the two original datapoints whose label is the corresponding combination of the original labels. To facilitate this, we express the original labels $y_1, y_2 \in \mathcal{Y}$ in vector form $\gamma(y_1), \gamma(y_2) \in \mathbb{R}^{\#\mathcal{Y}}$. That is

$$(\gamma(y))_i = \begin{cases} 1 & \text{if } y \text{ is the } i\text{th label in } \mathcal{Y} \\ 0 & \text{otherwise} \end{cases}. \tag{5.3}$$

The mixup datapoint is $(1-\lambda)x_1 + \lambda x_2$ and has vector label $(1-\lambda)\gamma(y_1) + \lambda\gamma(y_2)$. It remains to describe how to calculate the loss for such a datapoint, since (2.45) does not directly apply to vector labels. The loss in (2.45) can be seen as a special case of the Kullback-Leibler divergence, a measure of the distance between probability distributions. If $p \in \Delta^{\mathcal{Y}}$ and $q \in \Delta_+^{\mathcal{Y}}$ (where $\Delta^{\mathcal{Y}}$ and $\Delta_+^{\mathcal{Y}}$ are the set of probability distributions and nonnegative probability distributions over $\mathcal{Y}$ respectively), then the Kullback-Leibler divergence is given by

$$\mathrm{KL}(p, q) := \sum_{\substack{y \in \mathcal{Y} \\ p_y \neq 0}} p_y \log\left(\frac{p_y}{q_y}\right). \tag{5.4}$$

The original loss function (2.45) is then $l(h, x, y) = \mathrm{KL}(\gamma(y), f_h(x))$. We calculate the loss at a mixup datapoint as

$$\mathrm{KL}((1-\lambda)\gamma(y_1) + \lambda\gamma(y_2), f_h((1-\lambda)x_1 + \lambda x_2)). \tag{5.5}$$

Mixup has been successfully used for many different labeling problems, but seems from an intuitive point of view particularly suitable for audio labeling, since the audio scenes we actually hear in the real world are often the superposition of sounds from many different sources. Although the competition winners attribute a large part of their success to mixup augmentation, several other teams used mixup and scored significantly worse than me. In addition, my own experiments with mixup do not show that it is significantly better than pitch shifting. It may be that mixup is only effective in combination with some other aspect of the winner's algorithm.

5.6 Data Balancing

The training dataset for the 2018 DCASE Freesound General-Purpose Audio Tagging Challenge does not contain an equal number of clips of each of the 41 possible categories. The category "Glockenspiel" is the least common in the training set, having only 94 clips, while 17 different categories are tied for most common, with 300 clips. This discrepancy is not huge, and I did not expect it to significantly affect accuracy. After some unsuccessful preliminary experiments with re-weighting the loss associated with clips based on the frequency of their category, I did not use data-balancing for the rest of the competition.

The winning team[JL] used an interesting data-balancing scheme that works by modifying the minibatch gradient descent algorithm (see Section 3.2). Instead of forming a minibatch in each iteration by selecting a subset of the training set uniformly at random, they form a minibatch of size 41 containing one random training clip of each of the 41 possible categories. Intuitively, this approach is appealing because in addition to enforcing balance between the categories, it also forces the optimization algorithm to take into account every category in every iteration. For example, the categories "Fireworks" and "Gunshot or gunfire" are sometimes hard to distinguish. If we include an example of both in every minibatch, then we might expect to decrease the odds of an erroneous step that improves the network's ability to recognize "Fireworks" at the cost of dramatically worsening its ability to detect "Gunshots or gunfire".

After the competition, I experimented with retraining one of my models with this data balancing scheme. Doing so increased the median accuracy. Moreover, the 95% confidence intervals for the model's accuracy without and with the data balancing scheme were $[0.8384, 0.8459]$ and $[0.8461, 0.8537]$ respectively, indicating a statistically significant improvement (the intervals were computed using the same method as in Section 5.5).

5.7 Data Weighting

One idiosyncrasy of this contest's training dataset is that the labels were assigned by users of the Freesound website, and so are not completely reliable. Some of the clips have a "verified" annotation, indicating that the label has been verified by someone besides Freesound users. Thus it is conceivable that there would be value in weighting examples with this annotation higher than examples without it. I attempted this, but found that it did not improve my validation accuracy.

The winners of the contest[JL] employed a special technique to deal with the unreliable data. In each iteration of minibatch gradient descent, they used the gradients at only a subset of the training examples in that iteration's minibatch. The gradients at verified training examples were automatically used. The gradient at an unverified training example was used if the loss function (see (2.45)) evaluated at that training example was less than 0.8 times the maximum loss over all examples in the minibatch. The idea was that unverified training examples with very high loss might be improperly labeled, and so should be excluded.

5.8 Clip Duration

The training data provided to contest participants consisted of sound clips ranging in length from 0.3 seconds to 30 seconds. This presented something of a problem for the use of minibatch gradient descent (see Section 3.2). Although both the ResNet and Progressive NASNet architectures can be trained and tested on input with different L and W dimensions, efficient algorithms implementing convolutional layers require that all tensors in a minibatch have the same L and W (see Section 2.5.1). Since all clips were transformed with the same frequency decomposition, all input tensors had the same L. However, different clip lengths gave them different W. To deal with this issue, whenever I trained a model, I fixed a target width W_τ. Then, in each iteration, whenever the model drew a minibatch, I adjusted the width of the spectrogram of each clip to match W_τ. If a clip had a spectogram with $W > W_\tau$, a random crop of width

W_τ was selected. If a clip had $W < W_\tau$, I increased the width of clip's spectrogram either by padding with silence (50% probability) or by repeatedly duplicating the clip (50% probability).

In addition to being a practical technique to enable efficient minibatch gradient descent, the clip length adjustment procedure can also be seen as an instance of data augmentation (see Section 5.5), since every time a given clip with large W is selected to be used in the current minibatch, a different crop may be used. In this way, the procedure described resembles the cropping augmentation frequently used in image classification (see Section 4.1 of [KSH12]).

5.9 Ensembling

As is common in machine learning competitions, I achieved my best accuracy by averaging the predictions of a number of different models. My final ensemble consisted 14 different convolutional neural networks: 7 ResNets and 7 Progressive NASNets. For both of the architectures, I varied the size of the networks as well as the clip duration (see Section 5.8). Averaging models trained with different clip duration seemed particularly valuable, possibly because the resultant ensemble took into account information on different time scales.

5.10 Software

I implemented the deep neural networks in Pytorch[PGC+17], a Python automatic differentiation framework with a C++/ CUDA backend. I used Librosa[MRL+15], a Python audio library, for audio processing.

5.11 Results

Table 5.1 shows the top-1 prediction accuracies of some of the models I experimented with. I have already discussed the benefits of pitch-shifting data augmentation in Section 5.5 (compare row 5.1.1 with row 5.1.2 and row 5.1.7 with 5.1.8). Generally, the results show that making a ResNet or Progressive NASNet architecture deeper or wider (and thus increasing the number of parameters and floating point operations) makes it more accurate, but that there are diminishing returns. Comparison of rows 5.1.5 and 5.1.6 shows that removal of a downsampling layer from a very deep ResNet also increases accuracy, likely by giving the model access to more precise time and frequency information, but at the cost of an increased number of floating point operations. Row 5.1.10 shows the value of training a network with a longer clip duration. I can think of two possible explanations for this. The first is that certain characteristic patterns of sound that identify the category of a clip are longer than 1 second, so that training with a clip length of only 1 second does not allow the network to learn to identify these patterns. The second explanation is based on the observation that a significant number of the longer sound clips contain regions of either silence or irrelevant background noise. Using a clip duration of only 1 second implies a substantial likelihood that the portion of such a clip chosen for use in a minibatch may be entirely silence or background noise, which may interfere with optimization. Using a longer clip duration may prevent this problem.

Clips 2 provides some examples of mis-classifications by my final ensemble. 06775f3c, whose true category is "Bus", consists mostly of a human voice, and was likely classified as "Laughter" because some training clips of laughter contain human voices. 7c9fb8d6, whose true category is also "Bus", is mostly the white-noise-like sound of the bus itself, which is similar to the white-noise-like sound heard during applause in a large room, explaining the mis-classification. The rising high-pitch firework sound heard in clip 0ed387f3 resembles the sound produced when a piece of paper is torn. The forceful banging heard in clip 38957ab8 has many of the subjective auditory properties of the sound of gunshots, though most humans would probably be able to tell the difference. b1b41c63 is a low note played on a guitar, which I expect many humans would have a hard time distinguishing from the sound of a plucked cello.

The creators of Progressive NASNet[LZS+17] report its superiority over the ResNeXt and ShuffleNet architectures, while the creators of ResNeXt and Shufflenet report that those architectures yield better accuracy than ResNet[XZ17]. My results on the DCASE 2018 dataset do not strongly bear out this hierarchy: the performance of Progressive NASNet and ResNet are similar for architectures that require similar numbers of floating point operations. It is possible that my comparison was not fine-grained enough, and that if I were to experiment with more depths and widths of Progressive NASNet and ResNet, I would find evidence of the superiority of the first over the second. It is also possible that Progressive NASNet is somehow specifically adapted for image processing, and loses some of its edge when applied to the audio processing task.

Row	Name	Median Acc	25 pct	75 pct	Mults ($\times 10^6$)	Params ($\times 10^4$)
5.1.1	ResNet ($C = 60$) no augmentation	0.8279	0.8216	0.8335	51	28
5.1.2	ResNet ($C = 60$) + PS	0.8426	0.8378	0.8468	51	28
5.1.3	ResNet ($C = 60$) + PS + Balance	0.8510	0.8447	0.8553	51	28
5.1.4	ResNet ($C = 60$) + PS + Swish	0.8449	0.8366	0.8495	51	28
5.1.5	ResNet ($C = 60$, $\times 3$ depth) + PS	0.8480	0.8433	0.8528	104	84
5.1.6	ResNet ($C = 60$, $\times 3$ depth,-1 DS) + PS	0.8587	0.8507	0.8628	331	84
5.1.7	Prog. N. ($C = 48$) no augmentation	0.8380	0.8364	0.8470	215	222
5.1.8	Prog. N. ($C = 48$) +PS	0.8549	0.8499	0.8576	215	222
5.1.9	Prog. N. ($C = 96$) + PS	0.8595	0.8582	0.8632	755	838
5.1.10	Prog. N. ($C = 96$) + PS + clip len.=5sec	0.8822	0.8814	0.8822	755	838
5.1.11	Final Ensemble	0.9138	-	-	-	-

Table 5.1: Median Acc: median accuracy; 25 pct: 25th percentile accuracy; 75 pct: 75th percentile accuracy; Mults: approximate number of floating point multiplies required to apply the trained network to a 900ms sound clip; Params: number of parameters in the neural architecture. PS: Pitch shifting (see Section 5.5); Prog. N.: Progressive NASNet. Note that C refers to the number of channels in the first residual or progressive NAS unit. "$\times n$ depth" indicates that each unit in the original architecture has been replaced by n stacked copies in the new architecture . -1 DS indicates that the first downsample, whether it works by striding or pooling, has been removed from the architecture. Balance indicates that the data balancing scheme described in Section 5.6 is used. Clip len. indicates that a clip length other than the default 1 second has been used. The results in rows 5.1.1, 5.1.2, and 5.1.3 are from 30 runs each. Results in rows 5.1.4 to 5.1.10 are from 24 hours worth of runs each. Each run consisted of 2 cycles of the cyclic cosine annealing procedure described in Section 5.4. Each cycle consisted of 100 epochs. Each epoch consisted of a number of iterations equal to the dataset size divided by the minibatch size.

True Category	Predicted Category	Clip
Bus	Laughter	06775f3c
Bus	Applause	7c9fb8d6
Fireworks	Tearing	0ed387f3
Knock	Gunshot_or_gunfire	38957ab8
Acoustic_guitar	Cello	b1b41c63

Clips 2: Examples of mis-classified clips. When this essay is viewed with a recent version of Adobe Reader with the Adobe Flash Player plugin installed, double-clicking a clip name will play it.

	With Pitch Shifting			Without Pitch Shifting			
	Median	Conf Int L	Conf Int R	Median	Conf Int L	Conf Int R	Median Diff
Overall	0.8426	0.8384	0.8459	0.8279	0.823	0.8329	0.0147
Acoustic_guitar	0.8333	0.8056	0.8333	0.7778	0.75	0.8333	0.0555
Applause	1	1	1	1	1	1	0
Bark	0.8696	0.8261	0.913	0.8696	0.8261	0.8696	0
Bass_drum	0.913	0.8696	0.913	0.8696	0.8261	0.913	0.0434
Burping_or_eructation	1	0.9615	1	0.9615	0.9615	0.9615	0.0385
Bus	0.7	0.65	0.75	0.7	0.65	0.7	0
Cello	0.9545	0.9318	0.9545	0.9318	0.9091	0.9545	0.0227
Chime	0.7083	0.7083	0.75	0.6667	0.6667	0.6667	0.0416
Clarinet	0.9778	0.9556	0.9778	0.9556	0.9333	0.9749	0.0222
Computer_keyboard	0.7619	0.7143	0.8034	0.7143	0.6667	0.7619	0.0476
Cough	0.9583	0.9167	0.9583	0.9167	0.9167	0.9583	0.0416
Cowbell	0.9706	0.9412	0.9706	0.9412	0.9412	0.9706	0.0294
Double_bass	0.9375	0.9375	0.9688	0.9375	0.9062	0.9375	0
Drawer_open_or_close	0.5417	0.4583	0.5833	0.625	0.5054	0.6667	-0.0833
Electric_piano	0.7692	0.7357	0.8077	0.7692	0.7308	0.7692	0
Fart	0.9167	0.9167	0.9583	0.875	0.8333	0.9113	0.0417
Finger_snapping	0.963	0.9259	0.963	0.963	0.9259	0.963	0
Fireworks	0.6731	0.6204	0.7258	0.6538	0.6154	0.6923	0.0193
Flute	0.9659	0.9545	0.9773	0.9545	0.9545	0.9773	0.0114
Glockenspiel	0.5417	0.5	0.5833	0.5417	0.5	0.5833	0
Gong	0.9	0.9	0.9333	0.9	0.8667	0.9	0
Gunshot_or_gunfire	0.6078	0.5686	0.6641	0.6471	0.6275	0.6812	-0.0393
Harmonica	0.8148	0.7778	0.8519	0.7778	0.7407	0.8148	0.037
Hi-hat	0.8438	0.8165	0.875	0.8125	0.8125	0.871	0.0313
Keys_jangling	0.7391	0.7391	0.7826	0.7391	0.7013	0.7826	0
Knock	0.8125	0.8125	0.8438	0.8438	0.8125	0.8438	-0.0313
Laughter	0.9032	0.9032	0.9355	0.9032	0.871	0.9355	0
Meow	0.875	0.875	0.9113	0.8333	0.7917	0.875	0.0417
Microwave_oven	0.75	0.672	0.75	0.7083	0.6667	0.7446	0.0417
Oboe	0.9706	0.9412	0.9706	0.9706	0.9412	0.9706	0
Saxophone	0.9659	0.9333	0.9659	0.9432	0.9205	0.9545	0.0227
Scissors	0.35	0.35	0.4	0.35	0.35	0.4	0
Shatter	0.875	0.8333	0.875	0.8333	0.7917	0.8333	0.0417
Snare_drum	0.9286	0.8929	0.9643	0.8929	0.8571	0.8929	0.0357
Squeak	0.3125	0.2917	0.375	0.25	0.2083	0.3333	0.0625
Tambourine	0.9688	0.9688	0.9688	0.9375	0.9375	0.9688	0.0313
Tearing	0.8864	0.8636	0.9091	0.9091	0.8636	0.9545	-0.0227
Telephone	0.6923	0.6667	0.6923	0.641	0.6154	0.6634	0.0513
Trumpet	0.9667	0.9667	1	0.9333	0.9333	0.9667	0.0334
Violin_or_fiddle	0.9713	0.9655	0.977	0.9655	0.954	0.977	0.0058
Writing	0.7083	0.6304	0.7083	0.75	0.7083	0.7917	-0.0417

Table 5.2: Per-category median accuracy of a ResNet ($C = 60$) model trained with and without pitch shifting augmentation. 30 runs were performed both with and without pitch shifting. The left and right endpoints of a 95% confidence interval calculated using the interpolated sign test are also given (see [GF93] and [BH93]). The values in the final column are the differences between the median accuracies with and without augmentation.

5.12 Other Attempts

Over the course of the competition, I attempted many variants of my core strategy, many of which turned out to be unsucessful. I describe a few here.

- Following Zeghidour ConvNet[ZUS$^+$18], I incorporated the short-time Fourier transform into the neural architecture to achieve end-to-end training. That is, I introduced a special initial convolutional layer that accepted an unprocessed one-dimensional amplitude signal and, with its initial weights, closely modeled the short-time Fourier transform. I allowed these weights to be learned during training, in the hopes of discovering a time-frequency representation more suitable that the STFT to the problem at hand. Unfortunately, models produced by this procedure proved inferior to those found by using a fixed short-time Fourier transform. I suspect that this may be because allowing the time frequency transformation to be trainable means having a trainable convolution with a very large kernel size. Experience demonstrates that the most easily trained convolutional neural networks have small or moderately sized kernels.

- I attempted to account for the unreliability of some of the unverified sound clips using virtual adversarial training (VAT)[MMKI17]. My procedure was to train one model, and after training, identify the unverified sound clips (see Section 5.7) for which the trained model had a high loss, and erase the label on these sound clips. I then trained a second model on the modified training data. For sound clips with an erased label, I used virtual adversarial loss instead of the loss of (2.45). Virtual adversarial loss encourages the predicted labels of the unlabeled sound clips to be stable to perturbations of the data. This approach is similar in spirit to the one used by the winners and described in Section 5.7, but unfortunately it did not improve the accuracy of my model. Virtual adversarial training is most appropriate when the unlabeled datapoints fit into one of the label categories. Some of the unverified datapoints in the 2018 DCASE training set do not actually fit into any of the label categories. If there are enough of these points, this could explain the lack of improvement from the use of virtual adversarial training.

- I attempted additional data augmentation by randomly mixing the sound clips with background noise from the BBC sound effects library[4], while being sure that none of the background noises contained sounds from the list of possible labels (See Section D). In contrast to pitch-shifting augmentation, this background-noise augmentation slowed training without providing a noticeable benefit to validation accuracy.

- The authors of [MR17] describe a transfer-learning procedure by which they improved the performance of their neural network model on a speech recognition task by first pre-training it for a general audio classification task. They used their pre-trained network (modified to use the correct number of categories) as an initial guess while training for the speech recognition task. I attempted the opposite transfer, first training on the Google Speech Commands[War17] dataset, and then transferring to the contest dataset. Unfortunately this yielded no appreciable benefit. It may be that transfer learning works better when the transfer is from a general task like audio classification to a more specific task like speech recognition than in the opposite direction.

[4]http://bbcsfx.acropolis.org.uk

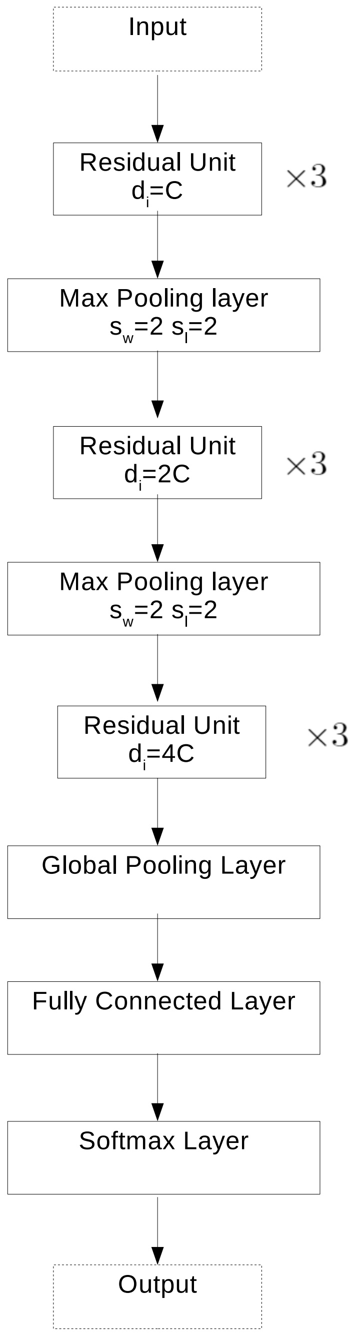

Figure 5.4: An example residual network. See Section C for layer descriptions. C is a parameter. d_i indicates the number of input channels of a given residual unit. The number of output channels of a residual unit is always equal to the d_i of the next unit. $(s_w, s_l) = (2, 2)$ are the strides of the max pooling layers (see Section C). All convolutional layers in residual units use the same nonlinear activation function, while the fully connected layer uses the identity as its activation function.

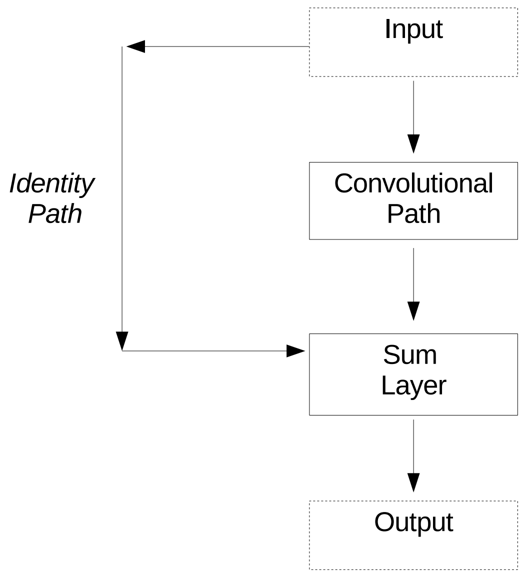

Figure 5.5: Residual unit

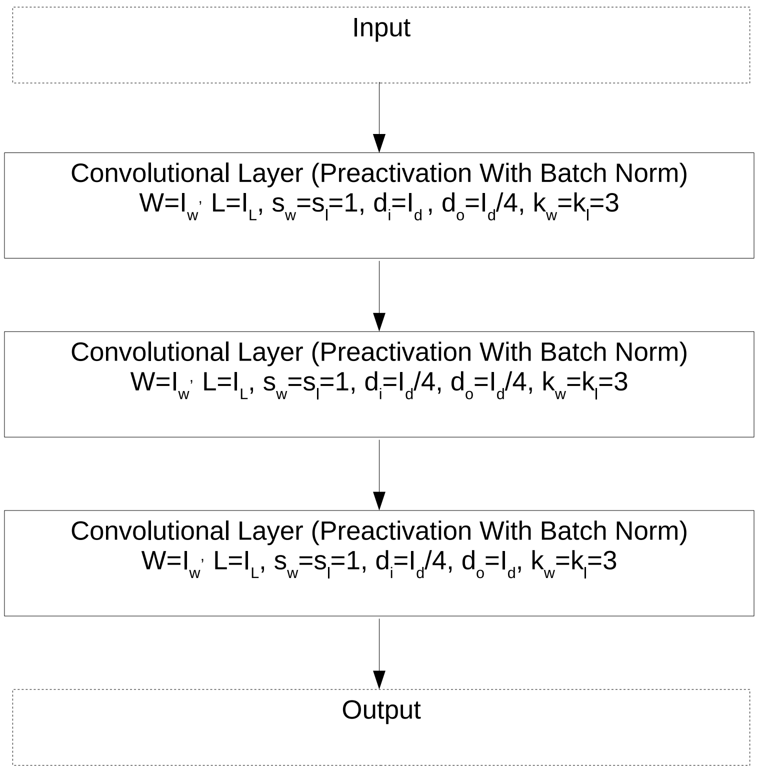
Input
Convolutional Layer (Preactivation With Batch Norm)
W=I_w, L=I_L, s_w=s_l=1, d_i=I_d , d_o=I_d/4, k_w=k_l=3
Convolutional Layer (Preactivation With Batch Norm)
W=I_w, L=I_L, s_w=s_l=1, d_i=I_d/4, d_o=I_d/4, k_w=k_l=3
Convolutional Layer (Preactivation With Batch Norm)
W=I_w, L=I_L, s_w=s_l=1, d_i=I_d/4, d_o=I_d, k_w=k_l=3
Output

Figure 5.6: A bottleneck convolutional path. The path accepts as input a tensor of dimensions (I_W, I_L, I_d) and produces as output a tensor with the same dimensions. All convolutions have a kernel size of $(k_w, k_l) = (3, 3)$ and are unstrided $((s_w, s_l) = (1, 1))$ Note that unlike the standard convolutional layers described in the text, modern ResNets use pre-activation, according to which the activation function is applied to the input tensor before — not after— the convolution.

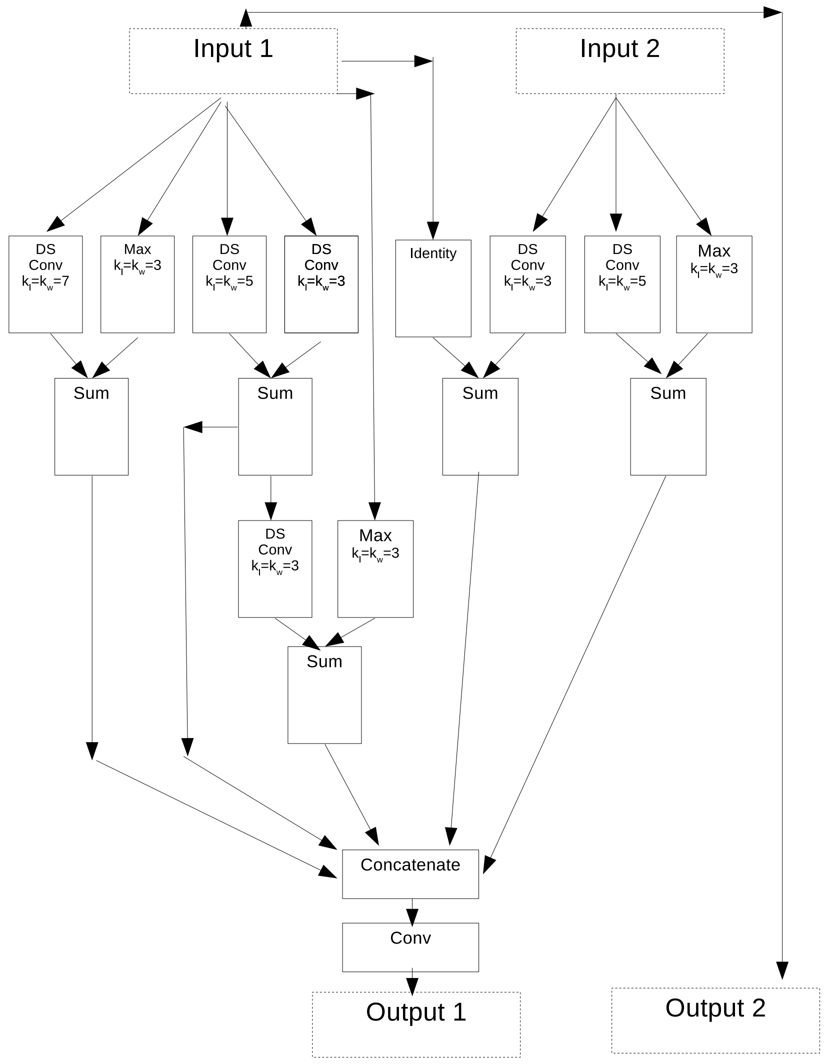

Figure 5.7: Progressive NASNet Unit. DS Conv stands for Depthwise Separable Convolution. See Section C for definitions of the layer types. In each pooling and convolutional layer, (k_w, k_l) indicates the kernel size. All convolutional layers used batch normalization. Each Progressive NASNet unit has parameters d_i and d_o, specifying the number of input and output channels. All layers produce d_i output channels except for the final convolution, which produces d_o. If $d_i \neq d_o$, an extra convolution is added before output 2 to generate an appropriate number of output channels.

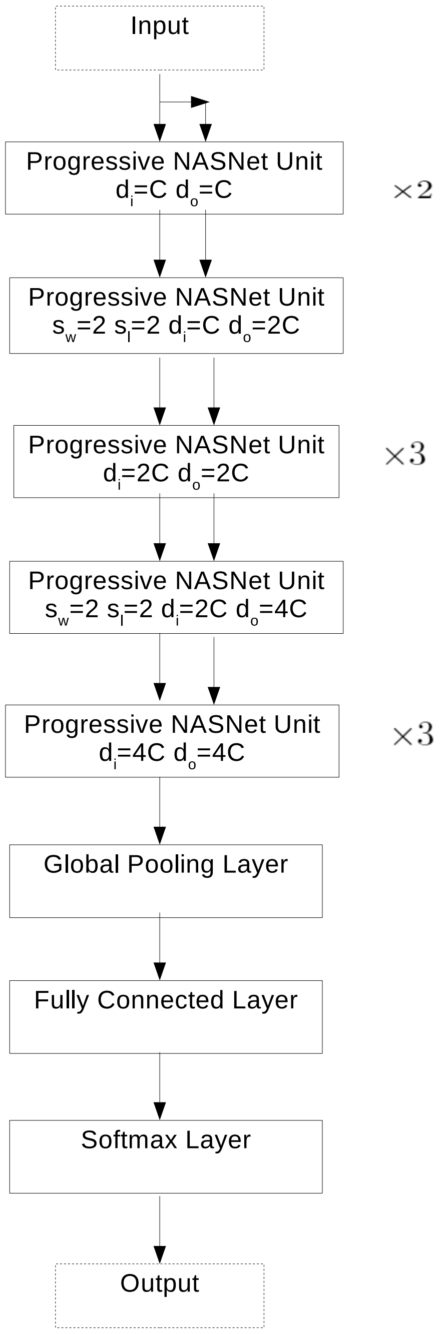

Figure 5.8: A Progressive NASNet Architecture. C is a parameter. For each progressive NASNet unit, d_i indicates the number of input channels, d_o indicates the number of output channels, and s_w, s_l indicates the stride used to downsample the input tensors received. All convolutions in progressive NASNet units have the same nonlinear activation function. The Fully connected layer has the identity as its activation function.

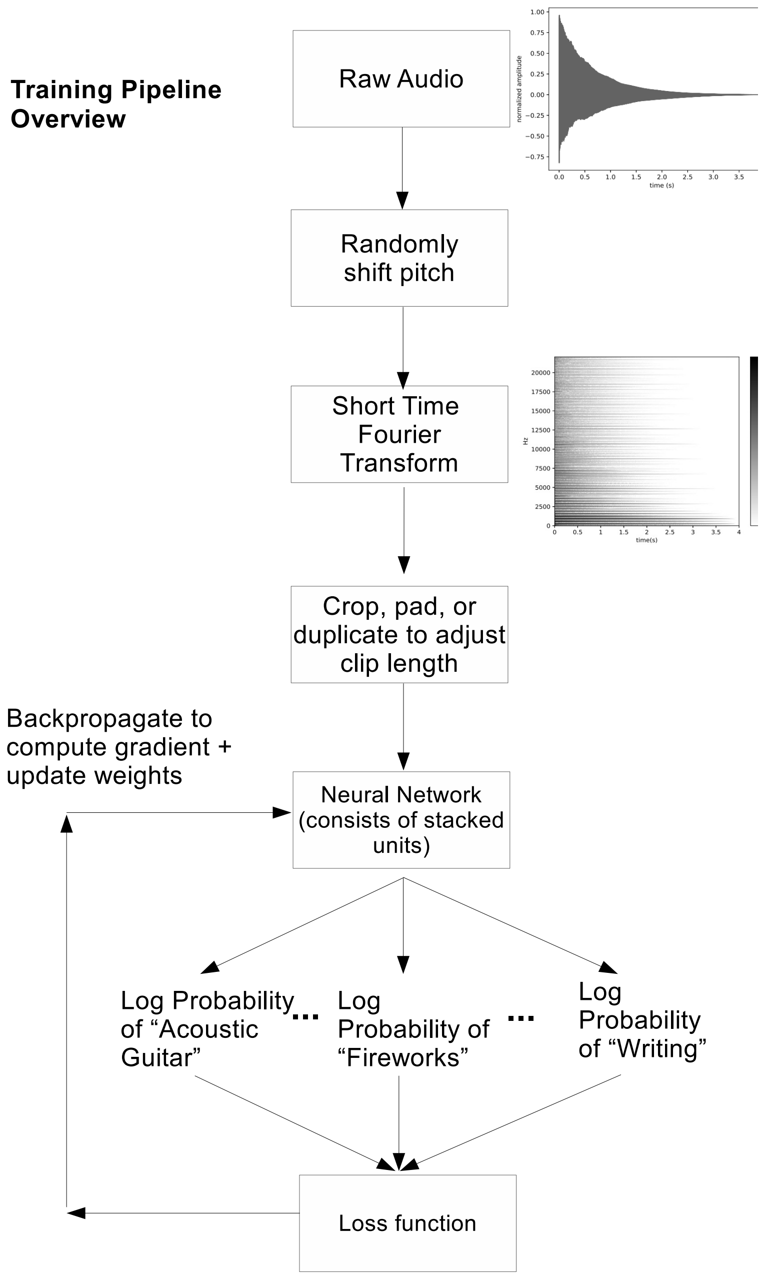
Training Pipeline Overview
Raw Audio
Randomly shift pitch
Short Time Fourier Transform
Crop, pad, or duplicate to adjust clip length
Backpropagate to compute gradient + update weights
Neural Network (consists of stacked units)
Log Probability of “Acoustic Guitar”
...
Log Probability of “Fireworks”
...
Log Probability of “Writing”
Loss function
normalized amplitude
time (s)
Hz
time(s)

Figure 5.9: An overview of the pipeline by which the neural networks were trained.

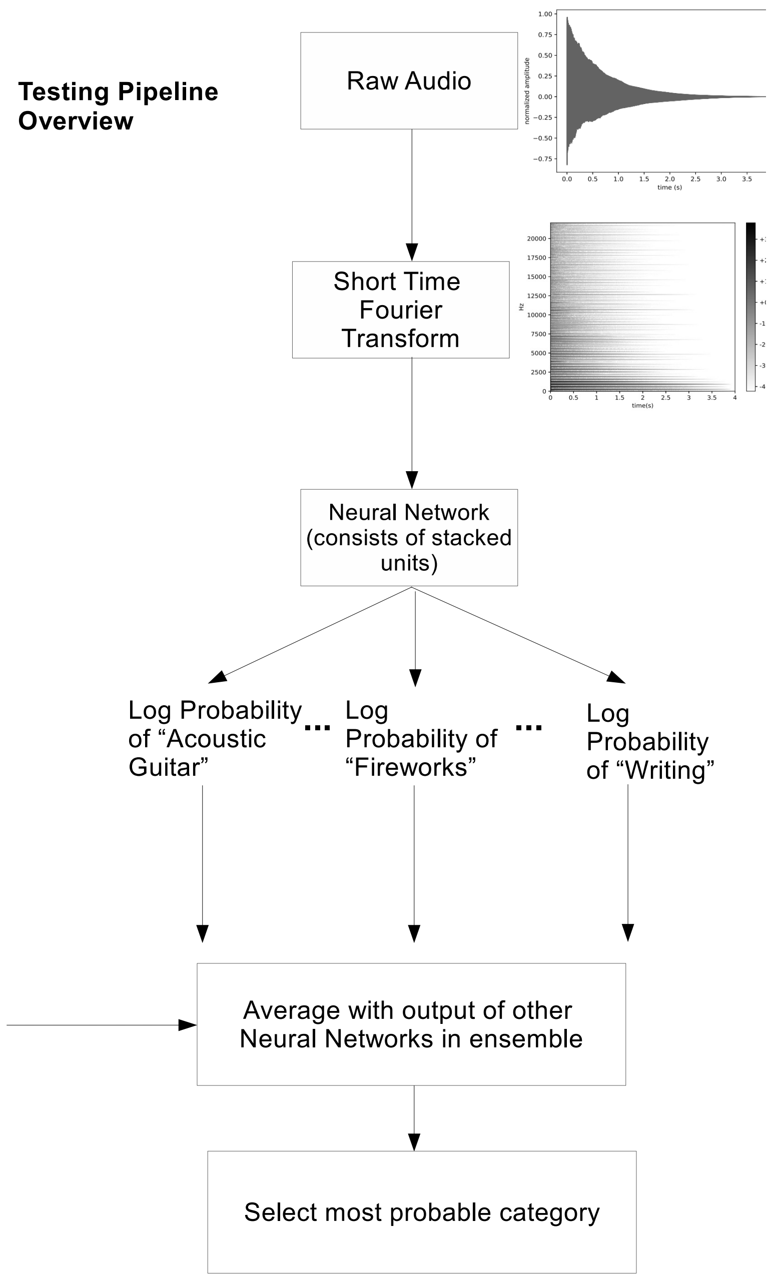
Testing Pipeline Overview
Raw Audio
Short Time Fourier Transform
Neural Network (consists of stacked units)
Log Probability of “Acoustic Guitar”
...
Log Probability of “Fireworks”
...
Log Probability of “Writing”
Average with output of other Neural Networks in ensemble
Select most probable category

Figure 5.10: An overview of the pipeline by trained neural networks were used to label test sound clips.

Chapter 6

Conclusions

In the first part of this essay, I surveyed some of the theory of neural networks, illustrating the various ways neural networks derive their power from their compositional structure. In the case of simple shallow neural networks, which are linear combinations of pre-compositions of a fixed function with variable affine functions, this compositional structure enables the use the use of the chain rule to show that simple shallow neural networks can efficiently approximate any polynomial and hence any continuous function. In the case of deep networks, I cited results showing that because deep neural networks are more compositional than shallow ones, (a) deep neural networks can represent more complex and interesting functions than shallow networks with a given neuron budget (for the right definitions of “complex” and “interesting”); and (b) in certain restricted cases (ReLU-nets and networks whose compositional structure conforms to that of the target function) deep neural networks have better order of approximation than shallow networks. While these results are impressive, they are also somewhat disjointed. As I see it, what is required is a theory that specifies a very general class of interesting functions and shows how the compositional structure of deep neural networks makes them particularly suited for approximating functions of this class. Obviously, it is far easier to describe what is needed than to accomplish it.

In the second part of this essay, I described my entry into an audio-labeling machine learning competition. My experience in this competition demonstrates that in the application of deep neural networks to machine learning, although selecting a good architecture is important, choosing the right data processing techniques is also vital. Pitch-shifting data augmentation was key to performance my models, and without it I would not have placed in the top 3%. Furthermore, from the winners’ description of their algorithm it seems that what distinguished them from other competitors was their methods of augmenting, balancing, and masking the data[1]. One avenue for future work might be the partly or fully automated design of such data processing schemes.

[1]Though in my experiments, only their data balancing scheme improved the accuracy of my models.

Appendix A

Description of Backward Propagation

Definition A.1. A computation graph is *scalar* if it has only one output vertex and that output vertex has range $\mathbb{R}$. Denote the output vertex of a scalar computation graph by v_{out}.

Definition A.2. The function F induced by a scalar computation graph (see Definition 3) can be differentiated via a recursive procedure called *backward propagation.* For vertexes v and w with $v \in \text{in}(w)$ denote by $\frac{\partial w}{\partial v}[z] : \mathcal{R}_v \to \mathcal{R}_w$ the linear function representing the partial derivative at $z \in \mathcal{R}_{\text{in}(w)}$ of c_w with respect to the input it receives from vertex v. Then define for each vertex v and each input to the graph $x \in \mathcal{D}$ the linear function $\frac{\partial F}{\partial v}[x] : \mathcal{R}_v \to \mathbb{R}$ given by

$$\frac{\partial F}{\partial v}[x] := \begin{cases} 1 & \text{if } v = v_{\text{out}} \\ \sum_{w \in \text{out}(v)} \frac{\partial F}{\partial w}[x] \circ \frac{\partial w}{\partial v}\left[F_{\text{in}(w)}(x)\right] & \text{otherwise} \end{cases}. \tag{A.1}$$

By chain rule $\frac{\partial F}{\partial v}$ is the derivative of the F with respect to the output of c_v. Letting Dc_v denote the total derivative of c_v with respect to all its inputs, the total derivative of F is then given (again by chain rule) by

$$DF[x] = \bigoplus_{v \in V_{\text{in}}} \frac{\partial F}{\partial v}[x] \circ Dc_v[x_v]. \tag{A.2}$$

Appendix B

Additional Proofs

B.1 Non-Smooth Activation Functions

Here we show that the requirements of Proposition 2 can weakened to demand only a continuous activation function. We first prove an auxially result.

Proposition B.1. *If $f \in \operatorname{cl}\mathrm{SSNN}_1(\sigma)$ then for all $\lambda, \theta \in \mathbb{R}$, $f(\lambda \cdot +\theta) \in \operatorname{cl}\mathrm{SSNN}_1(\sigma)$*

Proof. Follows since if we have $\mathrm{SSNN}_1 \supset \{g_n\} \to f$ uniformly on compact sets then $\mathrm{SSNN}_1 \supset \{g_n(\lambda \cdot +\theta)\} \to f(\lambda \cdot +\theta)$ uniformly on compact sets. □

We are now able to weaken the requirements of Proposition 2.

Proposition 3 (Proposition 3.7 in [Pin99])**.** *Apply the topology of uniform convergence on compact sets to $C(\mathbb{R})$. Assume $\sigma \in C(\mathbb{R})$ is not a polynomial. Then all polynomials lie in $\operatorname{cl}\mathrm{SSNN}_1(\sigma)$.*

Proof. For any $\phi \in C^\infty(\mathbb{R})$ with compact support[1] define the function $\sigma_\phi : \mathbb{R} \to \mathbb{R}$ by

$$\sigma_\phi(x) := \int_{\mathbb{R}} \sigma(x-t)\phi(t)\,\mathrm{d}t. \tag{B.1}$$

We claim that $\sigma_\phi \in \operatorname{cl}\mathrm{SSNN}_1(\sigma)$. We will prove the claim by Riemann integration. Let $[-R, R]$ contain the support of ϕ, and for $m > 0$ let $\delta^m := \frac{2R}{m}$, let $t_i^m := -R + i\delta^m$, and let $r_m : \mathbb{R} \to \mathbb{R}$ be given by

$$r_m(x) = \sum_{i=1}^{m} \sigma(x - t_i^m)\phi(t_i^m)\delta^m. \tag{B.2}$$

Note that $r_m \in \mathrm{SSNN}_1(\sigma)$, so to prove the claim it suffices to show that the r_m converge to σ_ϕ uniformly on an arbitrary compact set $K \subset \mathbb{R}$. We have

$$|\sigma_\phi(x) - r_m(x)| \tag{B.3}$$

$$= |\int_{-R}^{R} \sigma(x-t)\phi(t)\,\mathrm{d}t - \sum_{i=1}^{m} \sigma(x - t_i^m)\phi(t_i^m)\delta^m| \tag{B.4}$$

$$\leq \sum_{i=1}^{m} \left(\int_{t_{i-1}}^{t_i^m} |\sigma(x-t)||\phi(t) - \phi(t_i^m)|\,\mathrm{d}t \right) + \sum_{i=1}^{m} \left(\int_{t_{i-1}}^{t_i^m} |\sigma(x-t) - \sigma(x - t_i^m)||\phi(t_i^m)|\,\mathrm{d}t \right) \tag{B.5}$$

$$\leq 2R \sup_{z \in K + [-R,R]} |\sigma(z)| \sup_{\substack{t_a, t_b \in [-R,R] \\ |t_a - t_b| \leq \delta^m}} |\phi(t_a) - \phi(t_b)| + 2R \sup_{t \in [-R,R]} \phi(t) \sup_{\substack{z_a, z_b \in K + [-R,R] \\ |z_a - z_b| \leq \delta^m}} (\sigma(z_a) - \sigma(z_b)). \tag{B.6}$$

[1]The support of a function is the closure of the set of points at which it is nonzero.

Both terms go to zero as $m \to \infty$ by the uniform continuity of the continuous functions ϕ and σ on the compact sets $[-R, R]$ and $K + [-R, R]$. So we have proven the claim. It follows from Proposition B.1 that $\operatorname{cl}\mathrm{SSNN}_1(\sigma_\phi) \subset \operatorname{cl}\mathrm{SSNN}(\sigma)$.

Since $\sigma_\phi \in C^\infty(\mathbb{R})$[Fol13, Proposition 9.3b] for any $\phi \in C^\infty(\mathbb{R})$, we can apply the same technique as in the proof of Proposition 2 to show that for all $\phi \in C^\infty(\mathbb{R})$, all $k \geq 0$, and all $\theta \in \mathbb{R}$,

$$(t \mapsto t^k \sigma_\phi^{(k)}(\theta)) \in \operatorname{cl}\mathrm{SSNN}_1(\sigma_\phi). \tag{B.7}$$

There exists a family $\{\phi_n\}_{n\in\mathbb{Z}_+} \subset C^\infty(\mathbb{R})$ with compact support such that $\sigma_{\phi_n} \to \sigma$ uniformly on compact sets as $n \to \infty$[Fol13, Proposition 8.14b]. Now suppose to get a contradiction that for some $k \geq 0$ we have that the monomial t^k is not in $\operatorname{cl}\mathrm{SSNN}_1(\sigma)$. Then since $\operatorname{cl}\mathrm{SSNN}_1(\sigma_{\phi_n}) \subset \operatorname{cl}\mathrm{SSNN}_1(\sigma)$ for all n we have that the monomial t^k is not in $\operatorname{cl}\mathrm{SSNN}_1(\sigma_{\phi_n})$ for all n. But this implies by (B.7) that $\sigma_{\phi_n}^{(k)}(\theta) = 0$ for all θ for all n, so that σ_{ϕ_n} is a polynomial of degree less than k for all n. But since $\sigma_{\phi_n} \to \sigma$ uniformly on compact sets as $n \to \infty$, we have that σ is a polynomial of degree less than k, a contradiction. So $\operatorname{cl}\mathrm{SSNN}_1(\sigma)$ contains all the monomials and thus all the polynomials as desired. $\square$

B.2 Ridge Polynomials

In this section we show that the ridge polynomials of degree less than k span the polynomials of degree less than k. This was used in Proposition 4, and Proposition 6.

B.2.1 Multi-indexes

Definition B.1. If $\alpha \in \mathbb{Z}_+^n$ call α a *multi-index* and let $|\alpha| := \sum_{i=1}^n \alpha_i$ and $\alpha! = \prod_{i=1}^n \alpha_i!$. Furthermore denote

$$x^\alpha := x_1^{\alpha_1} \dots x_n^{\alpha_n} \qquad \text{for } x \in \mathbb{R}^n \tag{B.8}$$

$$D^\alpha := \frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1} \dots \partial x_n^{\alpha_n}}. \tag{B.9}$$

If $p : \mathbb{R}^n \to \mathbb{R}$ is a polynomial of degree k in n variables given by $p(x) = \sum_{|\alpha|\leq k} b_\alpha x^\alpha$, define the differential operator

$$p(D) = \sum_{|\alpha|\leq k} b_\alpha D^\alpha. \tag{B.10}$$

B.3 Polynomials

Definition B.2. A polynomial $p : \mathbb{R}^n \to \mathbb{R}$ is *homogeneous* of degree k if it is of the form $p(x) = \sum_{|\alpha|=k} b_\alpha x^\alpha$. Denote by H_k the vector space of such polynomials.

H_k is a vector space of dimension $\binom{n-1+k}{n-1}$. This follows from a combinatorial argument: consider $n - 1 + k$ slots into which $n - 1$ dividers can be placed to generate n regions. Each choice of where to put the dividers maps bijectively to a monomial in a basis for H_k, where the number of empty slots in the jth region corresponds to the exponent of x_j in the monomial.

Definition B.3. A function $p : \mathbb{R}^n \to \mathbb{R}$ is a *ridge monomial* if it is of the form

$$p(x) = \langle x, a\rangle^k \tag{B.11}$$

for some $a \in \mathbb{R}^n$ and $k \in \mathbb{Z}_+$. A *ridge polynomial* is a linear combination of ridge monomials.

Proposition B.2 (First part of Theorem 4.1 in [Pin99] and Proposition 5.1 in [Pin15])**.** *The ridge polynomials of degree k span the homogeneous polynomials of degree k*

Proof. First note that if α_1, α_2 are multi-indexes with $k = |\alpha_1| = |\alpha_2|$ and p_{α_2} is the monomial given by $p_{\alpha_2}(x) := x^{\alpha_2}$ then

$$D^{\alpha_1} p_{\alpha_2} = \begin{cases} \alpha_1! & \text{if } \alpha_1 = \alpha_2 \\ 0 & \text{otherwise} \end{cases}. \tag{B.12}$$

Since the monomials of degree k form a basis for H_k, the functions of the form D^α with $|\alpha| = k$ form a basis for the linear functionals on H_k (where the constant functions are identified with their value in $\mathbb{R}$). Thus any linear functional on H_k can be written as $q(D)$ where $q \in H_k$. Now consider applying a linear functional defined by $q \in H_k$ to a ridge monomial:

$$q(D)(\langle \cdot, a\rangle^k) = k! q(a). \tag{B.13}$$

A vector lies in a closed subspace if and only if all continuous linear functionals that vanish on the subspace also vanish on the vector[Fol13, Theorem 5.8 pg 159]. Consider a linear functional that vanishes on the ridge polynomials of degree k, viewed as a subspace of the homogeneous polynomials of degree k. By (B.13) the polynomial q corresponding to the linear functional is everywhere 0. Hence the linear functional vanishes everywhere. Hence the ridge polynomials span the homogeneous polynomials. $\square$

Proposition B.3 (Middle of proof of Theorem 4.1 in [Pin99])**.** *Let $n, k \geq 1$ Let $r = \binom{n-1+k}{n-1}$. There exist fixed vectors $a^1, \ldots, a^r \in \mathbb{R}^n$ such that for every polynomial p in n variables of degree less than on equal to k there are univariate polynomials $\pi_1, \ldots, \pi_r$ of degree less than or equal to k satisfying*

$$p(x) = \sum_{i=1}^{r} \pi_i(\langle x, a^i\rangle). \tag{B.14}$$

Proof. By Proposition B.2, there exists $a^1, \ldots, a^r$ such that every homogeneous polynomial p of degree k lies in $\operatorname{span}(\{\langle \cdot, a^i\rangle^k\}_{i=1}^r)$.

Now we claim that for $s < k$, every homogeneous polynomial of degree s lies in $\operatorname{span}(\{\langle \cdot, a^i\rangle^s\}_{i=1}^r)$ (for the same choice of $a^1, \ldots, a^r$). To get a contradiction, suppose the claim is not true. Then there must be a nontrivial linear functional on H_s that vanishes on $\{\langle \cdot, a^i\rangle^s\}_{i=1}^r$[Fol13, Theorem 5.8 pg 159]. So by the proof of Proposition B.2, there must be a $q \in H_s$ that vanishes on $A := \{a^1, \ldots, a^r\}$. Let $p \in H_{k-s}$ be nonzero. Then $pq \in H_k$ is a nonzero polynomial[2] and corresponds to a nonzero linear functional $pq(D)$ on H_k that vanishes on $\{(\langle \cdot, a_1\rangle)^k, \ldots, \langle \cdot, a_r\rangle)^k\}$. This contradicts our choice of A. So we have proven the claim.

Since the homogeneous polynomials of degree less than or equal to k span the algebraic polynomials of degree less than or equal to k, we have that every polynomial of degree less than or equal to k can be written as a linear combination of the terms of the form $\{a^i, x\}^s$ for $0 \leq s \leq k$ and $1 \leq i \leq r$. This implies the desired result. $\square$

Proposition B.3 is the result we require. It is more precise than was needed in Section 2.3.1, but the precision was necessary in Section 2.3.2.

[2]Since the multivariate polynomials are an integral domain.

Appendix C

Additional Layer Types for Convolutional Neural Networks

Definition C.1. Let v be a vertex in a computation graph with indegree $\deg_{\text{in}} \in \mathbb{Z}_{++}$. Call v a *sum layer* if for all $u \in \text{in}(v)$ $\mathcal{R}_u = \mathbb{R}^{d\times L\times W} = \mathcal{R}_v$ for some $d, L, W \in \mathbb{Z}_{++}$ and

$$c_v(x_1, x_2, \ldots, x_{\deg_{\text{in}}}) = \sum_{i=1}^{\deg_{\text{in}}} x_i. \tag{C.1}$$

Definition C.2. Let v be a vertex in a computation graph with indegree $\deg_{\text{in}} \in \mathbb{Z}_{++}$. Call v a *concatenation layer* if for all $u \in \text{in}(v)$ we have $\mathcal{R}_u = \mathbb{R}^{d_u\times L\times W}$ for some $d_u, L, W \in \mathbb{Z}_{++}$ and $\mathcal{R}_v = \mathbb{R}^{(\sum_{u\in\text{in}(v)} d_u)\times L\times W}$ and c_v operates by concatenating the input tensors along their first dimension to produce the output tensor.

Definition C.3. Let v be a vertex in a computation graph. Call v a *max layer* or *max pooling layer* if there are $d, W, L, s_w, s_l, k_w, k_l \in \mathbb{Z}_{++}$ such that $\mathcal{D}_v = \mathbb{R}^{d\times L\times W}$ and $\mathcal{R}_v = \mathbb{R}^{d\times\lceil\frac{L}{s_L}\rceil\times\lceil\frac{W}{s_w}\rceil}$ and

$$c_v(x)_{i,j_w,j_l} := \max\{x_{i,1+s_w(j_w-1)+p_w,1+s_l(j_l-1)+p_l}\}_{\lfloor\frac{k_w-1}{2}\rfloor\le p_w\le\lceil\frac{k_w-1}{2}\rceil,\lfloor\frac{k_l-1}{2}\rfloor\le p_l\le\lceil\frac{k_l-1}{2}\rceil} \tag{C.2}$$

where again we use zero-padding boundary conditions.

Note that in Section 5 in our description of Progressive NASNet we have occasion to discuss strided convolutional layers with strides $s_w, s_l \in \mathbb{Z}_{++}$. Such a layer is equivalent to a standard convolutional layer followed by a max layer with $k_w = k_l = 1$ and the given strides s_w, s_l. Of course, in practice strided convolutional layers are implemented more efficiently.

Definition C.4. Let v be a vertex in a computation graph. Call v a *global pooling layer* if there exist $d, W, L \in \mathbb{Z}_{++}$ such that v has vertex domain $\mathbb{R}^{d\times W\times L}$, vertex range $\mathbb{R}^d$, and vertex function $c_v : \mathbb{R}^{d\times W\times L} \to \mathbb{R}^d$ given by

$$c_v(A)_i = \frac{1}{WL}\sum_{j_w=1}^{W}\sum_{j_l=1}^{L} A_{i,j_w,j_l}. \tag{C.3}$$

Definition C.5. Let v be a vertex in a computation graph. Call v a *flattening layer* if there exist $d, W, L \in \mathbb{Z}_{++}$ such that v has vertex domain $\mathbb{R}^{d\times W\times L}$, vertex range $\mathbb{R}^{dWL}$, and if the vertex function c_v simply rearranges its input into a single column using row-major order.

Definition C.6. Let v be a vertex in a computation graph. Call v a *fully connected layer* with activation function σ if there exists $d_i, d_o \in \mathbb{Z}_{++}$ and $A \in \mathbb{R}^{d_o \times d_i}$ such that the v has vertex domain $\mathbb{R}^{d_i}$, vertex range $\mathbb{R}^{d_o}$, and vertex function c_v defined by

$$c_v(x)_k = \sigma(Ax)_k \tag{C.4}$$

for $1 \leq k \leq d_0$. The matrix A is called the *weight matrix.*

Definition C.7. Let v be a vertex in a computation graph. Call v a *softmax* layer if there exists $d \in \mathbb{Z}_{++}$ such that v has vertex domain and range $\mathbb{R}^d$ and vertex function given by

$$c_v(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{d} \exp(z_j)}. \tag{C.5}$$

C.1 Depthwise Separable Convolutions

To compute a single entry in the output tensor of a convolutional layer, one convolves across the channel dimension (index p in (2.41)) and across the depth dimensions (indexes q_w and q_l in (2.41)). Deep learning researchers at Google have argued that in trained ConvNets, the channel-wise and depth-wise aspects of a convolutional layer are sufficiently independent that it is valuable to impose this independence explicitly[Cho17]. That is, instead of convolving with the tensors $\{T^1, \ldots, T^{d_o}\} \subset \mathbb{R}^{d_i \times k_w \times k_L}$, we use depth-wise weight tensors $\{D^1, \ldots, D^{d_i}\} \subset \mathbb{R}^{k_w \times k_l}$ and channel-wise weight tensors $\{C^1, \ldots, C^{d_o}\} \subset \mathbb{R}^{d_i}$ and replace (2.41) with

$$S^d_{i,j_w,j_l} := \sum_{q_w, q_l \in \mathbb{Z}} D^i_{q_w,q_l} A_{i,j_w-q_w,j_l-q_l} \tag{C.6}$$

$$c_v(A)_{i,j_w,j_l} := \sigma\left(\sum_{p=1}^{d_i} C^i_p S^d_{p,j_w,j_l} + b_i\right). \tag{C.7}$$

Convolutions decomposed this way are said to be *depthwise-separable.* Such convolutions have been used in a number of very successful ConvNet architectures[Cho17, HZC+17].

Appendix D

Freesound General-Purpose Audio Tagging Challenge Labels

The aim of the competition was to correctly label each sound clip. The possible labels are listed below.

Acoustic guitar	Applause	Bark	Bass drum	Burping or eructation
Bus	Cello	Chime	Clarinet	Computer keyboard
Cough	Cowbell	Double bass	Drawer open or close	Electric piano
Fart	Finger snapping	Fireworks	Flute	Glockenspiel
Gong	Gunshot or gunfire	Harmonica	Hi-hat	Keys jangling
Knock	Laughter	Meow	Microwave oven	Oboe
Saxophone	Scissors	Shatter	Snare drum	Squeak
Tambourine	Tearing	Telephone	Trumpet	Violin or fiddle
Writing				

Appendix E

List of Symbols

$\lceil x \rceil$	Ceiling of x: the smallest integer greater than or equal to x.
$\mathrm{cl}(S)$	Closure of the set S: the limits of sequences in S
$\#S$	Number of elements in the set S
$\overline{x}$	Complex conjugate of x. If $x = a + bi$ then $\overline{x} = a - bi$.
$C(A)$	The set of continuous real-valued functions on A
$C^n(A)$	The set of continuous real-valued functions on A whose partial derivatives up to order n exist and are continuous.
$C^\infty(A)$	The set of continuous real-valued functions on A whose partial derivatives of all orders exist and are continuous.
$\cdot$	Used to specify a function with an implicit domain and range. For example, if $a \in \mathbb{R}^n$ then $\langle \cdot, a \rangle$ refers to the function $f : \mathbb{R}^n \to \mathbb{R}$ given by $f(x) = \langle x, a \rangle$.
$\Delta^{\mathcal{Y}}$	If $\mathcal{Y}$ is a finite set, then Δ^Y is the set of probability distributions over $\mathcal{Y}$.
$\Delta^{\mathcal{Y}}_+$	The set of positive probability distributions over $\mathcal{Y}$.
$\langle x, y \rangle$	Inner product. Unless otherwise specified, this is the standard real inner product $\langle x, y \rangle = \sum_i x_i y_i$.
$\lfloor x \rfloor$	Floor of x: largest integer less than or equal to x.
$\mapsto$	Also used to specify a function with an implicit domain and range. For example $x \mapsto x^2$ refers to the function $f : \mathbb{R} \to \mathbb{R}$ given by $f(x) = x^2$.
$A_{i,:,:}$	(MATLAB notation) If $A \in \mathbb{R}^{d \times W \times L}$, then $A_{i,:,:} := (A_{i,j,k})_{j \in \{1,\ldots,W\}, k \in \{1,\ldots,L\}} \in \mathbb{R}^{W \times L}$.
$\mathbb{R}$	The real numbers
$\mathbb{R}_+$	The non-negative real numbers
$\mathbb{R}_{++}$	The positive real numbers
$\mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$	The set of d-dimensional rectangular arrays of real numbers such that dimension i has size n_i.
LReLU	The function defined by $\mathrm{LReLU}(x) = \max(x, 0.01x)$
ReLU	The function defined by $\mathrm{ReLU}(x) := \max(x, 0)$
$\mathbb{Z}$	The integers
$\mathbb{Z}_+$	The non-negative integers
$\mathbb{Z}_{++}$	The positive integers

Bibliography

[B⁺15] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[BBL02] Thomas Bagby, Len Bos, and Norman Levenberg. Multivariate simultaneous approximation. *Constructive approximation*, 18(4):569–577, 2002.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BCN18] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[BH93] Rudolf Beran and Peter Hall. Interpolated nonparametric prediction intervals and confidence intervals. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 643–652, 1993.

[Boo81] Conte & Boor. *Elementary Numerical Analysis; an Algorithmic Approach.* McGraw Hill., 1981.

[Bre94] Albert S Bregman. *Auditory scene analysis: The perceptual organization of sound.* MIT press, 1994.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[CCL95] Tianping Chen, Hong Chen, and Ruey-wen Liu. Approximation capability in $C(\mathbb{R}^n)$ by multilayer feedforward networks and related problems. *IEEE Transactions on Neural Networks*, 6(1):25–30, 1995.

[CFS16] Keunwoo Choi, George Fazekas, and Mark Sandler. Automatic tagging using deep convolutional neural networks. *arXiv preprint arXiv:1606.00298*, 2016.

[Cho17] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint*, pages 1610–02357, 2017.

[CWV⁺14] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[DDS⁺09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[EDM13] Gianpaolo Evangelista, Monika Dörfler, and Ewa Matusiak. Arbitrary phase vocoders by means of warping. *Musica/Tecnologia*, 7:91–118, 2013.

[ESC+12] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

[Fol13] Gerald B Folland. *Real analysis: modern techniques and their applications.* John Wiley & Sons, 2013.

[FPF+18] Eduardo Fonseca, Manoj Plakal, Frederic Font, Daniel PW Ellis, Xavier Favory, Jordi Pons, and Xavier Serra. General-purpose tagging of freesound audio with audioset labels: Task description, dataset, and baseline. *arXiv preprint arXiv:1807.09902*, 2018.

[Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[GAG+17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[GEF+17] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 776–780. IEEE, 2017.

[GF93] Jean D Gibbons and Jean D Gibbons Fielden. *Nonparametric statistics: An introduction.* Number 90. Sage, 1993.

[Grö13] Karlheinz Gröchenig. *Foundations of time-frequency analysis.* Springer Science & Business Media, 2013.

[HACN90] Feng-hsiung Hsu, Thomas Anantharaman, Murray Campbell, and Andreas Nowatzyk. A grandmaster chess machine. *Scientific American*, 263(4):44–51, 1990.

[HCE+17] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. Cnn architectures for large-scale audio classification. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 131–135. IEEE, 2017.

[HCL+17] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

[HGN90] RM Hyatt, Albert E Gower, and Harry L Nelson. Cray blitz. In *Computers, Chess, and Cognition*, pages 111–130. Springer, 1990.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[Hor93] Kurt Hornik. Some new results on neural network approximation. *Neural networks*, 6(8):1069–1072, 1993.

[HWT$^+$15] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[HZC$^+$17] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[JL] Il-Young Jeong and Hyungui Lim. Audio tagging system for dcase 2018: Focusing on label noise, data augmentation and its efficient learning.

[KB09] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KH09] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KSL18] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? *arXiv preprint arXiv:1805.08974*, 2018.

[KWM16] David B Kirk and W Hwu Wen-Mei. *Programming massively parallel processors: a hands-on approach (3rd edition)*. Morgan kaufmann, 2016.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBD$^+$89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[LCJB$^+$89] Yann Le Cun, Lionel D Jackel, Brian Boser, John S Denker, Henry P Graf, Isabelle Guyon, Don Henderson, Richard E Howard, and William Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46, 1989.

[LGP$^+$90] Douglas B Lenat, Ramanathan V. Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. Cyc: toward programs with common sense. *Communications of the ACM*, 33(8):30–49, 1990.

[LH16] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[LTHS88] Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

[LXTG17] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

[LZS$^+$17] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017.

[Mai10] VE Maiorov. Best approximation by ridge functions in l p-spaces. *Ukrainian Mathematical Journal*, 62(3):452–466, 2010.

[MKS$^+$15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[MMKI17] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *arXiv preprint arXiv:1704.03976*, 2017.

[Mon17] Guido Montúfar. Notes on the number of linear regions of deep neural networks. 2017.

[MP43] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[MP16] Hrushikesh N Mhaskar and Tomaso Poggio. Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications*, 14(06):829–848, 2016.

[MPCB14] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932, 2014.

[MR17] Brian McMahan and Delip Rao. Listening to the world improves speech command recognition. *arXiv preprint arXiv:1710.08377*, 2017.

[MRL$^+$15] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, pages 18–25, 2015.

[Pet98] Pencho P Petrushev. Approximation by ridge functions and neural networks. *SIAM Journal on Mathematical Analysis*, 30(1):155–189, 1998.

[PGC$^+$17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[Pin99] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8:143–195, 1999.

[Pin15] Allan Pinkus. *Ridge functions*, volume 205. Cambridge University Press, 2015.

[R$^+$76] Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1976.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 2017.

[S$^+$04] Richard P Stanley et al. An introduction to hyperplane arrangements. *Geometric combinatorics*, 13:389–496, 2004.

[SGB$^+$15] Dan Stowell, Dimitrios Giannoulis, Emmanouil Benetos, Mathieu Lagrange, and Mark D Plumbley. Detection and classification of acoustic scenes and events. *IEEE Transactions on Multimedia*, 17(10):1733–1746, 2015.

[SHM$^+$16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[SNK11] Jan Schnupp, Israel Nelken, and Andrew King. *Auditory neuroscience: Making sense of sound*. MIT press, 2011.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization?(no, it is not about internal covariate shift). *arXiv preprint arXiv:1805.11604*, 2018.

[Tel15] Matus Telgarsky. Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*, 2015.

[TH12] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[TSF$^+$16] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[War17] Pete Warden. Speech commands: A public dataset for single-word speech recognition. *Dataset available from http://download. tensorflow. org/data/speech_commands_v0*, 1, 2017.

[WH60] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[WSC$^+$16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[WSH16] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv preprint arXiv:1611.10080*, 2016.

[XZ17] Mengxiao Lin Jian Sun Xiangyu Zhang, Xinyu Zhou. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv:1707.01083*, 2017.

[Yar17] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[Yar18] Dmitry Yarotsky. Optimal approximation of continuous functions by very deep relu networks. *arXiv preprint arXiv:1802.03620*, 2018.

[ZCDLP17] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[Zei12] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[ZL16] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[ZUS$^+$18] Neil Zeghidour, Nicolas Usunier, Gabriel Synnaeve, Ronan Collobert, and Emmanuel Dupoux. End-to-end speech recognition from the raw waveform. *arXiv preprint arXiv:1806.07098*, 2018.