A CROSS-DATA SET EVALUATION OF GENETICALLY EVOLVED NEURAL NETWORK ARCHITECTURES

by

Ben Gelman

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Science

Committee:

_____	Dr. Carlotta Domeniconi, Thesis Director
_____	Dr. Zoran Duric, Committee Member
_____	Dr. Huzefa Rangwala, Committee Member
_____	Dr. Sanjeev Setia, Chairman, Department of Computer Science
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Spring Semester 2019
George Mason University
Fairfax, VA

A Cross-Dataset Evaluation of Genetically Evolved Neural Network Architectures

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Ben Gelman
Bachelor of Science
George Mason University, 2015

Director: Dr. Carlotta Domeniconi, Professor
Department of Computer Science

Spring Semester 2019
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my friends and family, who have tirelessly supported and encouraged me.

Acknowledgments

I would like to thank my advisor, Carlotta Domeniconi, for always being there to push me forward. Thank you, Carlotta, for all the support and guidance you gave me whenever I stumbled. I would also like to thank my committee members, Huzefa Rangwala and Zoran Duric, for their time, feedback, and assistance. For Sam Gelman and David Slater, I appreciate all of the time you spent listening and supporting the ideas that made this thesis possible. I extend thanks to Casey Haber and Robert Gove for their advice on graphic design. Finally, I would like to thank Banjo Obayomi for infrastructure support and GPU cluster management.

Table of Contents

List of Tables

List of Figures

Abstract

A CROSS-DATASET EVALUATION OF GENETICALLY EVOLVED NEURAL NETWORK ARCHITECTURES

Ben Gelman

George Mason University, 2019

Thesis Director: Dr. Carlotta Domeniconi

The design of deep neural networks is often colloquially described as an 'art.' Although there are some common, guiding principles such as using convolutions for data with spatial locality or using recurrence for data with temporal characteristics, neural network architectures tend to be manually engineered. Few works currently provide methods to determine optimal architectures and hyper parameters. In this work, we generate empirical evidence for neural network architecture choices. We use a genetic algorithm to evolve 980 neural networks for a variety of common data sets. We analyze the characteristics of the highest performing architectures, compare those traits across data sets, and present a set of generalizable neural network design patterns.

Chapter 1: Introduction

The ubiquity of deep neural networks in recent literature is undeniable. The Sciencedirect database shows a rapid increase in the number of deep learning publications, with 6,837 publications in 2006 ballooning to a massive 16,288 publications in 2016 [1]. Additionally, the interest over time data on Google Trends[1] for "deep learning," displayed in Figure 1.1, shows just how significantly interest has grown in the last few years. This is unsurprising given the success, versatility, and ease of use of deep neural networks.

Deep networks have displayed state-of-the-art effectiveness in applications ranging from image classification [2] to machine translation [3]. Although the theoretical concepts required for deep neural networks are not very new, with even the first convolutional neural network dating back to the 1980s [4], many successes have only occurred relatively recently. This is due in no small part to increasingly massive datasets and changes in computational power. GPUs, in particular, have revolutionized the training of deep neural networks [5–7].

Although the use of specialized hardware has significantly increased the effectiveness of deep neural networks, it has also increased the cost of training and testing a variety of models to the point that it may be prohibitively expensive. Some successful work has even spent hundreds of hours using hundreds of GPUs to train models [8]. This level of investment is not feasible at scale, especially for those who are new to the field or otherwise do not have extensive resources.

Another significant downside of deep neural networks is their lack of transparency. For any given problem, it is not clear what the best neural network architecture would be. Although the universal approximation theorem [9] states that a neural network with just a single hidden layer and a finite number of neurons can approximate any continuous function

[1] trends.google.com

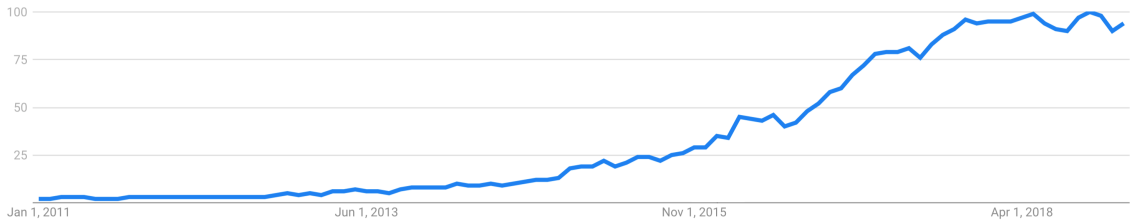


Figure 1.1: Google Trends interest over time data for "deep learning."

in a compact subspace of R^n, there are many practical limitations. The universal approximation theorem does not make any guarantees about the learnability of parameters; in practice, issues such as the quantity of available data and computational time make certain architectures far superior to a neural network with a single hidden layer. The difficulty and lack of clarity in determining these architectures is a barrier to entry for those with limited experience or resources.

Currently, the common methodology for designing neural networks is an experimental, adaptive, and time-consuming approach, evidenced by many hand-designed architectures [10–13]. Experience with a variety of problems plays a large role in the development of effective neural network architectures. Through trial and error, one is eventually able to identify the architectural design choices that help or hinder the problem they are working on. The possible space of viable architectures, however, is enormous. The amount of time and resources required to familiarize one's self with this space is non-trivial, particularly for those who are new to the field.

Genetic algorithms provide an avenue to automatically search this space. With minimal knowledge of deep learning, one can randomly generate and evolve neural networks until an adequate solution is found. While this strategy simplifies the architectural design process, it exacerbates the computational time requirements. With the already extensive time it takes just to train a single deep network, genetically evolving a large population of neural networks over multiple generations compounds the costs. For example, the authors in [8] use a GeForce GTX 980 GPU card to genetically evolve an LSTM, resulting in approximately

200 minutes of training per neural network with 25 generations and a population of 50 networks. If the networks were retrained in every generation, this would take upwards of 170 days, if not for the authors' cluster of 100 GPUs. Dedicating this level of resources to genetically evolve a neural network is simply infeasible for those who are learning or just entering the field.

Ideally speaking, the characteristics of a data set, and the classification problem one is trying to solve regarding that data set, should determine the most effective neural network architecture. A universal mapping from any given data set to an effective architecture is not currently known. The common practice of hand-designing architectures with a preconceived notion of what techniques should work best does not necessarily inform that mapping. Although search strategies, such as genetic algorithms, may have their own biases, it is at least possible to mechanically tune the trade-off between exploration and exploitation. Thus, search strategies are a good candidate for uncovering information about the relationship between data sets and their effective architectures because you can explore as many variables as you are willing to search through, limited by your computational resources and the robustness of your search strategy.

In this work, we attempt to gain insights into and discover patterns of effective neural network architectures with respect to the problems they are trying to solve. In order to accomplish this, we develop a genetic algorithm that can robustly mutate and crossover deep neural networks; we repeatedly run the genetic algorithm across a variety of data sets, generating a database of effective neural network architectures. We analyze the resulting architectures and how they relate to the characteristics of the data they evolved on.

Our contributions are as follows:

- A genetic algorithm that is able to arbitrarily mutate and crossover layers of a neural network.
- Evaluation of effective neural network architectures and how they relate to a variety of data sets.

- A set of generalizable neural network design patterns.

Chapter 2: Background

Two core concepts that we utilize heavily are neural networks and genetic algorithms. In order to maintain consistent terminology throughout the work, we introduce a few fundamental concepts from each topic.

2.1 Neural Networks

Neural networks are a machine learning algorithm loosely based on the way brains process information [14]. Neural networks combine units of computation, referred to as "neurons," to represent highly complex, nonlinear functions. The specific combination of a neural network's neurons is its "architecture." The connections between neurons are called "weights," and the output of a neuron is the weighted sum of the inbound connections passed through an "activation function." Because a weighted sum is a linear combination, the activation function is generally some non-linear operation that allows the neural network to learn complex relationships. There is a vast amount of literature on activation functions, but we focus our efforts on some of the most common choices: ReLU [15], sigmoid [16], linear, softplus [17], ELU [18], tanh [19], and SELU [20].

Designing a neural network architecture requires making choices about how the network's neurons are connected and which activations to use. There are a variety of strategies for how to connect neurons and apply transformations to the values going through the network, such as recurrence [21], long short-term memory [22], convolutions [23], sequence to sequence [24], and pooling [25], to name a few. We limit the scope of our work to use a sequence of layers, where the layers could be dense, convolutional 1D, or convolutional 2D.

A dense layer is an array of neurons whose "receptive field" is every neuron in the previous layer. The receptive field is the set of neurons that a given neuron takes as

input. A neuron in a dense layer will take the weighted sum of its receptive field, pass that value through an activation function, and then output the result. This is the traditional connection mechanism in neural networks.

Convolutional layers, on the other hand, change the paradigm of full connectivity. The receptive field of a convolutional neuron has a limited size, introducing a concept of spacial locality to the optimization process. Fig. 2.1 displays how a convolutional 1D layer operates on an array of neurons. The convolutional neuron has a receptive field size and a stride length. The stride explains how the receptive field of a convolutional neuron moves across the previous layer of neurons. The notion of connecting the weights to the previous layer is also slightly different; conceptually, it is convenient to consider the weights as connected to each position of the convolutional neuron's receptive field. When a particular position of the receptive field lines up with a neuron from the previous layer during striding, the output of the previous neuron will be multiplied by the weight corresponding to that position. Thus, the weight from a given position in a convolutional neuron's receptive field is often used many times while striding across the previous layer. The convolutional neuron produces an output by passing the weighted sum of the current receptive field through an activation function. Each stride generates one output, so a single convolutional neuron generates as many outputs as there are strides across the previous layer.

With a spatially sensitive receptive field, the notion of a layer's output shape becomes relevant. In the case of dense layers, the output is essentially always "flattened," i.e. the output of a dense layer is a 1 x n array, where n is the number of neurons in the dense layer. Because each neuron in a dense layer is connected to every neuron in the previous layer, the shape of the array has no impact on the results of the neural network. If we expand the concept of the receptive field to cover multiple dimensions, however, we can extract information from the previous layer in a different way. The convolutional 2D layer uses neurons that have a receptive field and stride with two dimensions. Fig. 2.2 displays how a convolutional 2D layer operates on a two dimensional array of neurons. The receptive field of the convolutional neuron simultaneously combines information from two dimensions. When

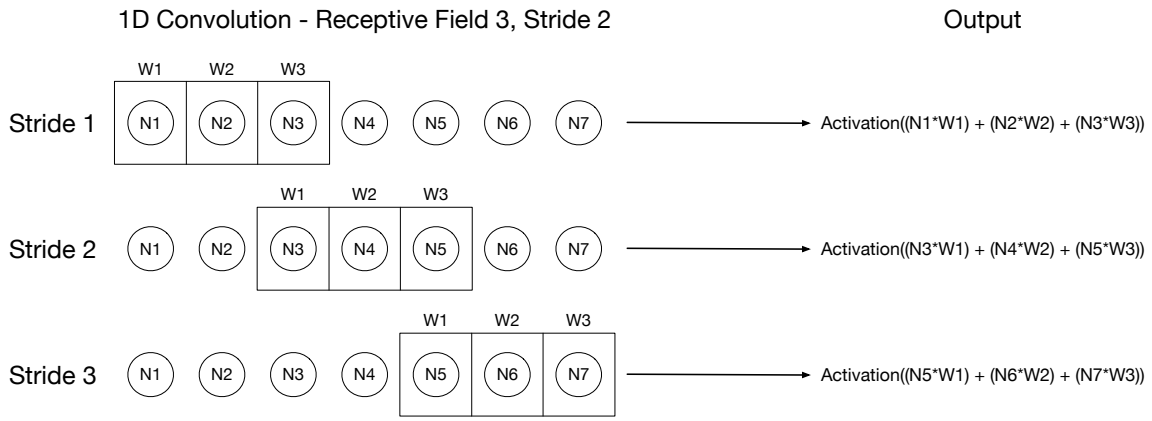


Figure 2.1: A convolutional 1D operation with a receptive field of 3 neurons and a stride of 2 neurons. The convolutional neuron takes the weighted sum of the neurons in its receptive field and passes that through an activation function to generate an output. Each stride generates one output. The weights of the convolutional neuron move with the stride, so W1, W2, and W3 are reused for every weighted sum.

striding, the receptive field moves all the way across the first dimension before incrementing the second dimension. This movement repeats until the second dimension is also complete. Like the 1D convolution, the 2D convolution generates many outputs; however, the output shape of the 2D convolution mirrors both dimensions of the striding. In Fig. 2.2, the receptive field creates two outputs before making a stride in the second dimension, and two outputs after. Thus, the values from the weighted sums in the output are actually organized in a 2x2 shape. Fig. 2.3 visually clarifies this phenomenon. Of particular note is the situation where the layer prior to the convolutional 2D layer has a flattened shape (i.e. $1\ x\ n$). This shape essentially renders the convolutional 2D layer useless. Because we generate neural networks using randomness, we must perform layer reshaping to ensure the validity of convolutional layers. We discuss the details in Chapter 4.

2.2 Genetic Algorithms

Genetic algorithms are a metaheuristic - a general class of techniques that utilize randomness to find effective solutions to optimization problems [26]. Metaheuristics are often used to

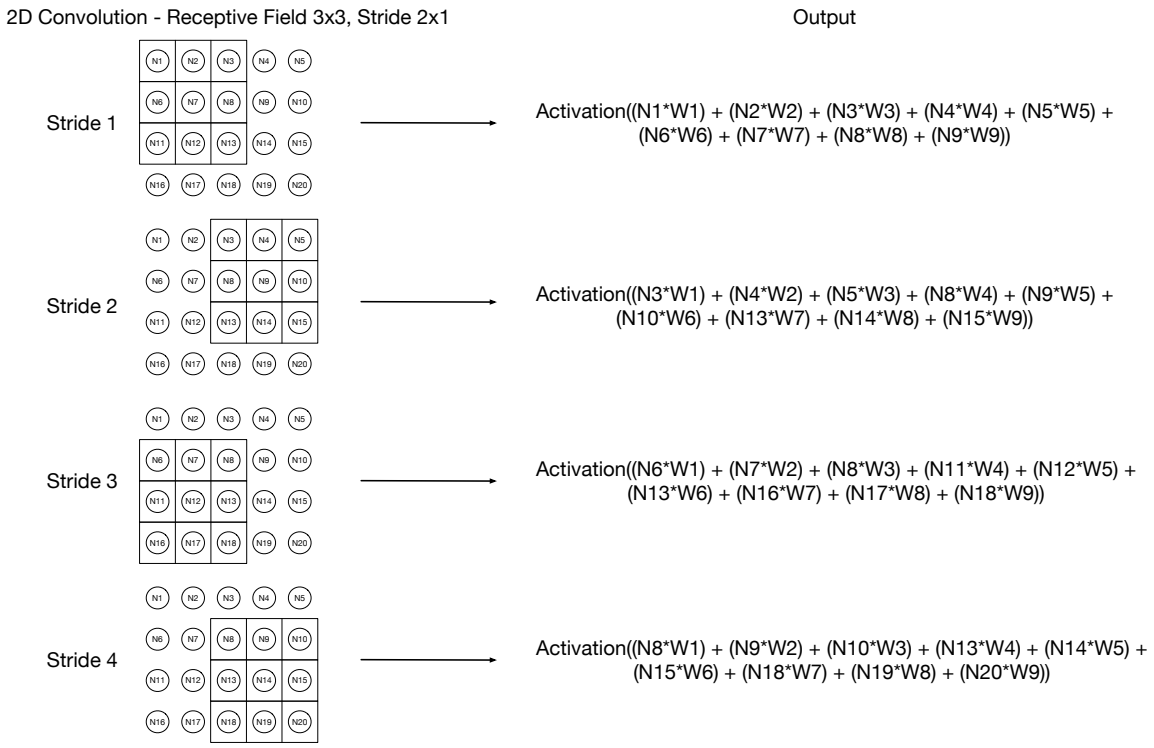


Figure 2.2: A convolutional 2D operation with a 3x3 receptive field and a 2x1 stride. The convolutional neuron takes the weighted sum of the neurons in its receptive field and passes that through an activation function to generate an output. Each stride generates one output, but there are now two dimensions of striding. The convolutional neuron strides along one dimension until it reaches the end, returns to the beginning of that dimension, and then strides once in the second dimension. This process repeats until there are no neurons to stride over in the second dimension. The weights of the convolutional neuron move with the stride, so W1, W2, W3, W4, W5, W6, W7, W8, and W9 are reused for every weighted sum.

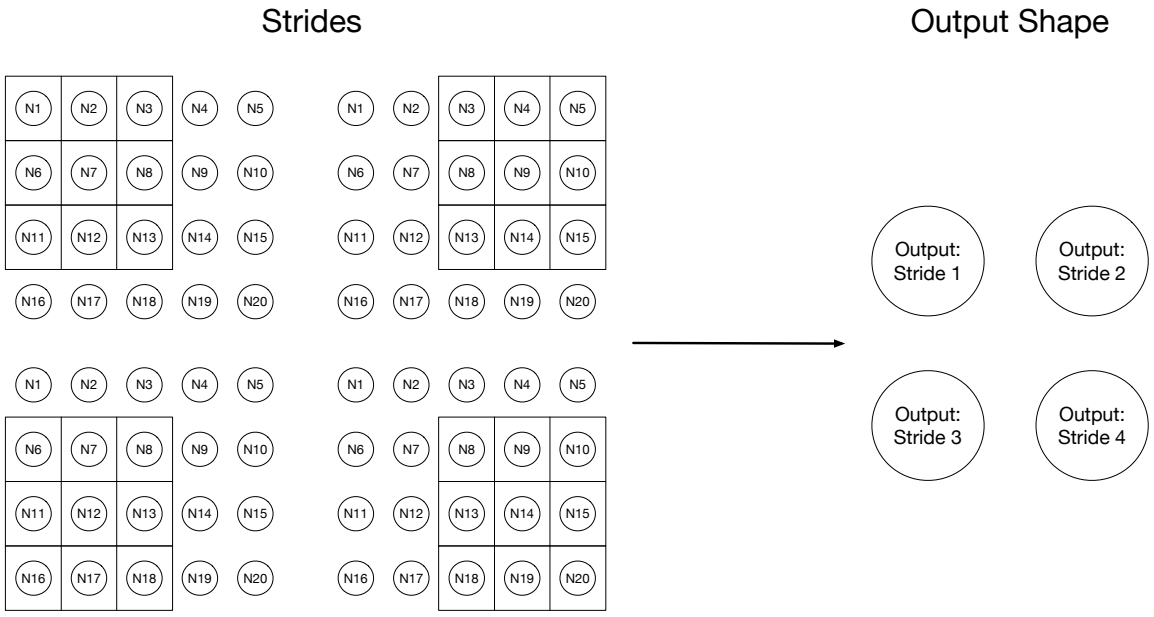


Figure 2.3: The two-dimensional output shape from the 2D convolutional neuron example in Fig. 2.2. The location of the stride determines the location of the output.

solve problems with incomplete information, where one has some procedure to assess the goodness of a given solution, but cannot determine that goodness beforehand. In other words, given a set of candidate solutions, it is not necessarily clear which solutions are the most effective until the evaluation is complete. In the context of learning effective neural network architectures, this is an apt approach: given a candidate set of deep neural networks for a particular data set, it is not entirely clear which are the most effective until they are trained and validated on the data.

Genetic algorithms are based on the idea of natural selection. In natural selection, a population of individuals pass on heritable traits via evolution [27]. Individuals that are best suited to their environment have the highest likelihood of passing on their traits. The set of an individual's traits, called a "chromosome," comprises "alleles," which are the specific instances of each trait [14]. An individual's ability to survive in its environment is its "fitness." Individuals with high fitness are likely to reproduce and create a new "generation" of offspring. In the creation of offspring, individuals can undergo "mutation" and

"crossover." Mutation modifies a single chromosome's alleles, whereas crossover normally exchanges some alleles between two chromosomes. The use of these random processes, along with the survival of the fittest individuals, allows subsequent generations of individuals to become more adept at handling their environment.

In genetic algorithms, an individual's chromosome represents a candidate solution to the given problem. Generally, a genetic algorithm starts by randomly initializing the chromosomes of some initial population. Then, an evaluation function determines the fitness of each individual by assessing how well the individuals perform on the given problem. A selection process must then make several choices: which individuals survive into the next generation, which individuals must be eliminated, and which individuals undergo mutation and crossover. There are a variety of selection strategies [26, 28], but most tend to employ randomness that favors the fittest individuals. The resulting individuals form the next generation, and the algorithm repeats the evaluation and selection procedures until it reaches a stopping criterion. Finally, the genetic algorithm returns the individuals with the highest fitness, regardless of the generation in which they are created.

We apply genetic algorithms to develop effective neural networks for a variety of data sets. In the context of neural networks, a chromosome is one network, and the alleles of that chromosome are the specific traits that constitute the network, such as the number of neurons in the second dense layer, the activation function of the third layer, etc. The specific design and configurations of our genetic algorithm are described in detail in Chapter 4.

Chapter 3: Related Work

The problem of neural network architecture selection is not new. The rise in popularity of neural networks in the late 80s and early 90s prompted a variety of analyses on optimal architecture selection. In 1991, Fogel called out the "artistic" nature of optimal neural network design [29]. Fogel modified Akaike's information criterion (AIC) [30] in order to select the appropriate the number of units in a neural network with a single hidden layer; however, there was no shortage of works at the time that tried to approximate the hidden layer size [31–40]. Despite the non-trivial amount of work on the issue, Fogel considered the hidden layer size selection problem unsolved. The following year, in 1992, Moody and Utans further suggested that comparing validation performance across a search or previously known set of architectures is the most effective option [35]. They claimed that the network optimization should be done with respect to the task at hand, and that it may not be completely possible to design the networks with only a priori knowledge. Moody and Utans did not, however, discount the use of heuristics and principled selection strategies, adding that validation performance can evaluate the usefulness of those approaches. Despite the great detail and sophistication of prior work, the proliferation of deep learning shook the foundation of architecture optimization. Not only were the number of layers in a network called into question, but also the choices of activation functions, layer types such as LSTMs and convolutions, and new forms of regularization such as dropout [41]. The inability of prior work in selection criteria to transfer to these new problems has left a gap in principled deep network architectural design.

Ultimately, Moody and Utans' experimental, adaptive approach to model selection is commonplace, even outside of neural networks. But, the plethora of variables that have arisen in deep learning have only magnified the possible space of architectures. In an effort to exclude human bias, a variety of works have employed search strategies to identify effective

deep learning architectures. Although many of Moody and Utans' contemporaries perform search across single hidden layer sizes, we focus on works targeting deep neural networks. Two topics are of particular relevance: search strategies for identifying effective deep neural network architectures, and heuristics, criteria, or patterns for effective deep neural network design. Although applying a genetic algorithm to search the space of effective deep network architectures is a key element of our work, the ultimate goal is to analyze the resulting networks for generalizable design patterns. Thus, we investigate prior work in both areas.

3.1 Search Strategies

Genetic algorithms have demonstrated success in identifying effective deep neural network architectures for a variety of problems. Miikkulainen et al. develop methods called DeepNEAT and CoDeepNEAT [8], which are genetic algorithms for deep neural network architecture evolution based on NEAT (NeuroEvolution of Augmenting Topologies) [42]. In general, NEAT treats neural networks as a graph of dense neurons, where the edges of the graph represent the receptive fields. Mutations can modify the neurons and edges, while crossover attempts to line up the graph representations and then share edges. DeepNEAT attempts to extend this method by increasing the level of abstraction from individual neurons to various deep network layers like dense, convolutional, and recurrent. Layers that are connected by an edge on the graph will feed data forward and appropriately use the layers' receptive fields. CoDeepNEAT abstracts the graph even further by allowing a node to represent its own subgraph of deep network layers. There is an additional optimization step that co-evolves "blueprints" of subgraphs and the actual deep networks those subgraphs represent.

We take inspiration from CoDeepNEAT, but we make several significant changes that we fully detail in Chapter 4. One of the primary differences is that we exclude the "neural network as a graph" paradigm, instead treating our networks as a sequence of layers. The robustness of the graph representation massively expands the potential space of viable architectures, but this is a double-edged sword. Although it is possible to discover architectures

that have been otherwise unexplored, the complexity of searching that space vastly increases. Additionally, the human interpretability of the solutions decreases: hand-designed networks that utilize an arbitrary graph representation of a neural network are very rare, many state-of-the-art networks use a sequential model, and known design patterns are often learned or evaluated with sequential models. Because our focus is to locate generalizable design patterns and reduce barriers to entry, we choose to simplify our genetic algorithm to sequential models. The reduced complexity of the search space allows us to allocate our computational time across many data sets, which is vital to evaluating the generalizability of our design patterns.

Although sequential models are common in hand-designed architectures and we choose to focus our evolution along those lines, there is a great diversity in evolutionary techniques nevertheless. Due to the difficulty of implementing crossover between two networks, some works forgo the process entirely [43–46]. Dufourq and Bassett opt for mutating an individual to create one offspring, and then mutating that offspring to create a second offspring [43]. Their mutations can add or replace a current layer with an entirely new, randomly initialized layer, but they prevent the mutation operation from modifying an existing layer's internal parameters. This limits the genetic algorithm's ability to perform exploitation of successful individuals. Real et al. use only one mutation to create an offspring, but their mutation operation allows more fine-grained control over individual parameters [44]. Real et al. deviate from the idea of fully retraining their population in each generation, allowing networks to retain their weights across generations and continue training. This method is highly exploitative of successful individuals, but limits the genetic algorithm's exploration of the space. They offset this issue by using a massive population size of 1000 networks, which are trained on 250 parallel workers. Similarly, Jozefowicz et. al attempt to evolve an LSTM using only mutations to climb towards a more effective architecture. By only using mutation, however, these works limit the ability to balance the exploitation and exploration of their genetic algorithms. As a result, we decide to use a two-tiered mutation

system that allows changes to any parameter in the network while also implementing a simple, yet robust, crossover operation to swap groups of layers between networks. We discuss our mutation and crossover operations in Chapter 4.

Aside from genetic algorithms, some works have utilized other methods to discover neural network architectures. Zoph and Le develop a recurrent neural network to predict effective architectures [47]. The recurrent network outputs sequences of layer parameters, such as number of neurons, receptive field size, and stride length, in order to construct a sequential model. The recurrent network attempts to optimize the validation performance of the predicted architecture, which is a non-differentiable metric. They use reinforcement learning with the validation performance of the predicted architecture as the reward signal in order to train the recurrent network. Although this method is interesting and demonstrates positive results, genetic algorithms provide a more interpretable avenue to control the tradeoff between exploration and exploitation. Because we are attempting to sample many architectures rather than locating a good solution to a single problem, we value the interpretability of the genetic algorithms.

There have also been various efforts to optimize just the hyperparameters for fixed architectures [48, 49]. Gurcan et al. manually build a convolutional architecture for microcalcification detection and then try to optimize the number of neurons and the receptive field sizes of the convolutional layers [48]. They try steepest descent, simulated annealing, and a genetic algorithm. Snoek et al. use Bayesian optimization with Gaussian process priors in order to optimize model hyperparameters [49]. Loshchilov and Hutter show that while Bayesian optimization is very effective on small function budgets, CMA-ES performs better for larger function evaluation budgets [50]. Although these methods are often more efficient at determining model hyperparameters than our general purpose genetic algorithm, they are not able to optimize network architectures. Our genetic algorithm optimizes the architecture and hyperparameters simultaneously, but applying hyperparameter-specific methods afterwards is a potential way to smooth the outputs of the genetic algorithm.

3.2 Design Patterns

The success of search strategies in the previous section demonstrates the viability of limiting manual neural network architecture design; however, those methods do not necessarily alleviate the barriers to entry. Many of the works in the previous section utilize an immense amount of computational resources in order to search for a single, competitive solution to a given problem. Because that is the explicit goal of these works, there are few insights into the space of effective neural network architecture. It is difficult to generalize the design patterns of one unique architecture for a single problem.

The literature on design patterns is limited, but there have been several attempts at a variety of network parameters nonetheless. Karlik and Olgac perform a small scale study of several different activation functions [51]. They use a neural network with a single hidden layer and test 5 different activation functions with hidden layer sizes of 10 neurons and 40 neurons. Between bi-polar sigmoid, uni-polar sigmoid, tanh, conic, and RBF activation functions, they find that tanh obtains the highest accuracy in both neural networks. Karlik and Olgac's study is straightforward and informative, but they do not test the generalizability of their results to other data sets, they do not utilize any deep networks, and their choices of activation functions are relatively uncommon in recent deep learning work. Ramachandran, Zoph, and Le develop a new activation function and compare its performance to ReLU on two different data sets [52]. Although they occasionally outperform ReLU, their activation function contains a hyperparameter that must be set or trained, hindering its usability.

He et al. [53] and Zagoruyko and Komodakis [54] discuss the effects of neural network depth and width on image classification. He et al. use networks as deep as 1000 layers, with each layer having few neurons. They show that this method performs well with fewer parameters than wide networks have. Zagoruyko and Komodakis claim that this level of network depth is unnecessary. By using a much wider, 16-layer network, they obtain similar accuracy and train several times faster, despite having up to two times as many parameters. Because GPUs are optimized for large tensor operations, wide networks are able to perform

computations more efficiently. Very deep networks do not fully utilize the hardware because layers must be processed sequentially. This is a valuable distinction because those without access to powerful GPUs can benefit from very deep networks with fewer parameters.

3.3 Overfitting to Validation

A significant concern that has been rarely addressed in prior work optimizing neural network architectures is the effect of repeated validation. Dwork et al. discuss adaptive data analysis, where model exploration and improvements are chosen based on previous analyses of the same data [55]. When a model is trained on one data set and validated on a separate holdout set, modifications based on the holdout set performance inform the model about the holdout data, even if the model is only ever trained on the training set. This causes the externally informed models to over-report their performance for the reused holdout set, despite not actually performing better across other holdout sets. This phenomenon has been thoroughly investigated [56–60].

Over-reporting validation performance is a critical issue in the genetic evolution of neural network architectures. Despite the many different approaches to genetically evolving neural network architectures, using validation performance to determine an individual's fitness is a nearly universal strategy. That fitness becomes an important selection criteria for creating offspring in the genetic algorithm. By using the same validation set in every generation of the genetic algorithm, the evolution inevitably favors individuals that over-report their performance for the particular validation set, even if those networks do not actually generalize better. We take several steps towards mitigating the effects of repeated validation by changing validation sets and performing multiple-fold validation. We discuss the specifics in Chapter 4.

Chapter 4: Methodology

4.1 Genetic Algorithm

Although we take inspiration from prior works in optimizing deep networks with genetic algorithms, we make a variety of different design choices in our own method. Our genetic algorithm focuses on assembling a sequence of layers using the three layers we introduce in Chapter 2. In our context, an individual's chromosome is a neural network; the alleles are a variety of layer-level and network-level parameters. Our algorithm has the ability to initialize random networks, mutate any layer-level or network-level parameters, and crossover consecutive layers between networks.

The random initialization procedure works by using a set of configurable initialization parameters. For every allele with a discrete variable in an individual's chromosome, there is a minimum and maximum starting value. There is a uniform random chance of selecting any value from that range. For every categorical variable, there is a uniform random chance of selecting one of the values.

The mutation operation works by changing parameters within layers. The dense layer is able to mutate the number of neurons and the activation function; the receptive field is always the entire previous layer. The 1D convolutional layer (conv1D layer) is able to mutate the number of neurons, one dimension of its receptive field, the receptive field's stride, and the activation function. The 2D convolutional layer (conv2D layer) is able to mutate the number of neurons, both dimensions of the receptive field, both dimensions of the receptive field's stride, and the activation function. Besides for layer parameters, there are also network-wide parameters that cannot be attributed to any single layer: the batch size of the input data, the ability to append a new layer to the end of the network, and the ability to delete a random layer. Fig. 4.1 displays all the mutable parameters in the genetic

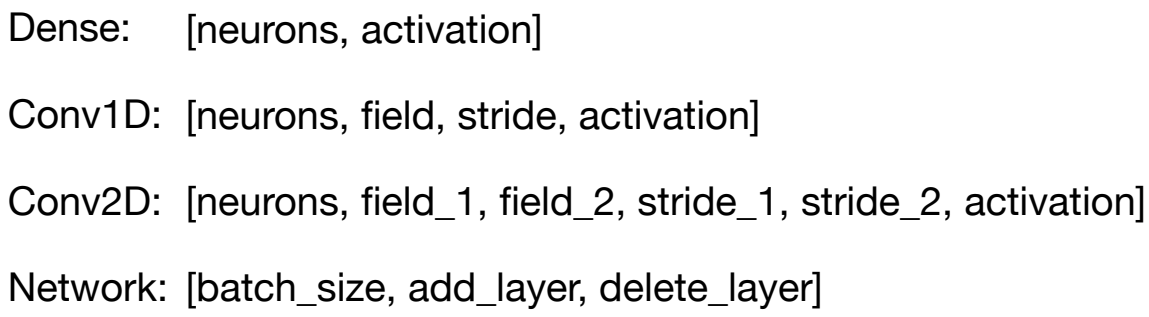


Figure 4.1: The layer-level and network-level mutable parameters.

algorithm.

The crossover operation in our genetic algorithm works by swapping layer positions between two networks. The crossover operation selects a layer position in two different networks and then swaps the layers. Fig. 4.2 demonstrates a basic crossover between two networks. We focus on layer-level crossover due to the unintuitive nature of crossing within-layer parameters. If, for example, the crossover were to choose a conv1D layer and a dense layer, it is not clear how to transfer the receptive field or stride parameters to the dense layer.

Even with limiting the scope to layer-level crossover, however, we still run into functional issues due to the nature of data shaping. The parameters of each layer dictate the output shape of that layer, and random combinations of layers can create nonsensical data flow. In the case of a dense layer followed by a conv2D layer, the dense layer outputs a one-dimensional shape, which completely invalidates the information a conv2D layer is able to learn with its second dimension. As a result, we perform a "squaring" operation, where we attempt to reshape the data into the squarest shape. Fig. 4.3 show an example of the squaring operation, which works by computing the factors of a given number and then taking the two closest factors. Although we recognize that this is a design decision that may influence the genetic algorithm's optimization process, we believe it is a more robust option than excluding the reshape operation. Without the reshaping, any dimensional flattening of the data would be irrevocable. Not only would this significantly reduce the use of conv2D

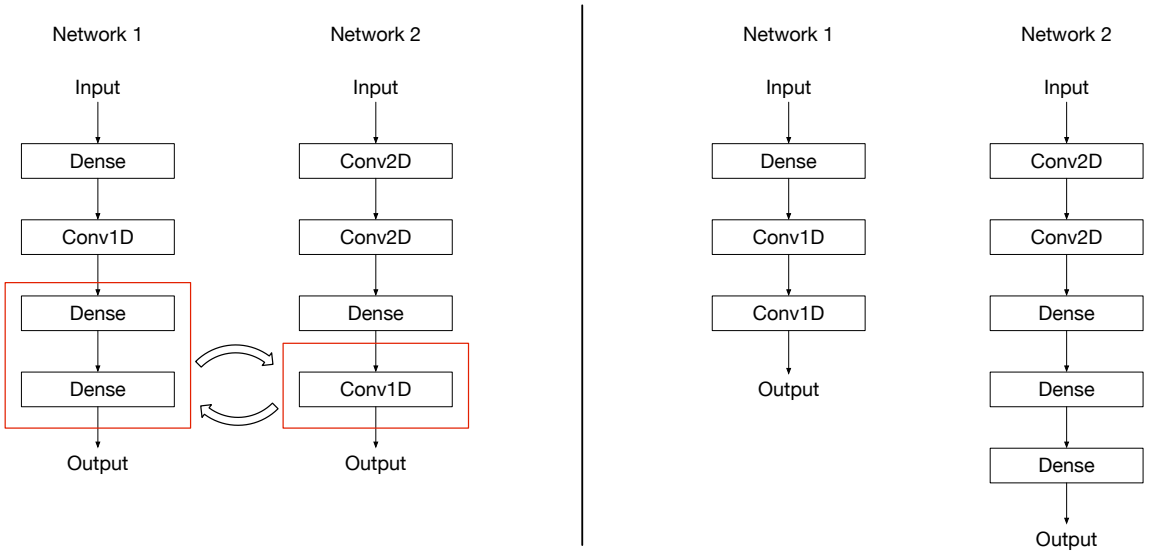


Figure 4.2: An example crossover operation between two networks. The left side of the image is before the crossover; the right side of the image is after the crossover.

layers, it could also reduce the likelihood of seeing deeper networks because the data would necessarily become flatter with more layers. To elaborate, conv2D layers with a large stride in the second dimension can result in the flattening of that dimension. Thus, even if a dense layer did not flatten the data, conv2D layers could accomplish that quickly on their own. Additionally, we flatten the data before a conv1D layer. Dense layers do not require reshaping because their receptive field is always every neuron in the previous layer.

Given our definitions of the mutation and crossover operations, we still need to assign probabilities for when they occur. In order to increase the robustness of the genetic algorithm's optimization process, we use a two-tiered system for mutation probabilities. There is a global mutation probability to determine whether a given network will undergo any mutation at all. When this check is passed, there is an individual mutation probability for every mutable parameter in the network. This includes both layer-level and network-level parameters, which are all displayed in figure Fig. 4.1. Thus, whenever the global mutation probability succeeds, any combination of the individual mutable parameters may be chosen for the resulting mutation.

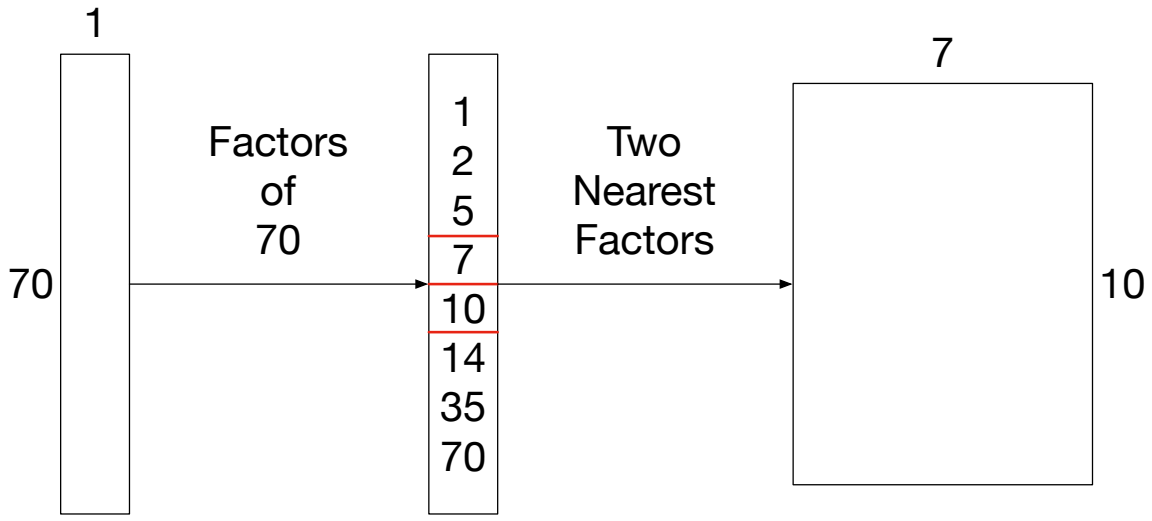


Figure 4.3: A squaring operation working on a flattened array of 70 units. The factors are sorted and the middle two entries (the two closest factors) are selected as the new dimensions for the data.

There is yet another caveat with mutation: discrete variables require boundaries on the magnitude of the mutation. When the two-tiered probabilities for a given allele with a discrete value succeed, our algorithm modifies the allele using a minimum and maximum magnitude specification with a uniform random chance of selecting any value in the range. Categorical variables do not require a magnitude specification; mutations will select any of the values with uniform random chance.

In the crossover operation, there is a single global crossover probability. When the crossover probability succeeds, there is a uniform random chance to select layer positions in the two networks chosen for crossover. The crossover is then performed according to Fig. 4.2, and the resulting networks undergo any reshaping if necessary. There are no magnitude parameters for crossover.

With the initialization, mutation, and crossover procedures fully defined, we can discuss the genetic algorithm's selection procedures. The algorithm first begins by initializing a configurable population size. Each individual calculates its fitness by training and validating on the given data set. We use the validation set accuracy as the individual's fitness for all

data sets. The algorithm then creates a generation of offspring using a process called tournament selection [61]. Tournament selection works by randomly selecting a number of individuals, equal to a configurable tournament size, from the current generation. From the individuals that are chosen to participate in the tournament, the individual with the highest fitness is selected as the winner. The winning individual is added to the offspring. The tournament selection repeats until the number of offspring is equal to the configurable population size. When the tournament size is very high, the offspring likely comprise the few fittest individuals because they are more likely to participate in many of the tournaments. A low tournament size provides additional opportunities for less fit individuals to be added to the offspring.

After the tournament selection is complete, we randomly split the offspring into pairs of two in preparation for crossover. Each pair is checked against the global crossover probability, and the individuals in the pair are crossed if the probability succeeds. Each of the resulting individuals are then checked by the two-tiered mutation probabilities, and the individuals' alleles are mutated wherever the probabilities succeed. The resulting individuals constitute the next generation in the genetic algorithm. The tournament selection, crossover, and mutation procedures are repeated until a configurable number of generations is reached. In every generation, we perform a new random split of the data into train and validation sets. Thus, every individual in a single generation trains on the same splits, but surviving individuals will be forced to train and validate on different data in the following generation. If an individual has managed to overfit to the validation set of one generation, that individual and its offspring will have to deal with different data in the next generation, helping to mitigate the long-term effects of over-reporting performance on the genetic algorithm.

The best individuals from any generation in the algorithm are stored in a "hall of fame," whose size is configurable. Every individual in the hall of fame trains and validates on 3 different splits of the data, and their validation accuracies are averaged. By using 3 different splits, we mitigate the likelihood of a model overfitting to a single, final validation set. The 5

individuals with the highest average validation accuracy are stored in a database of effective architectures. The rest of the networks in the hall of fame are discarded. Allowing additional space in the hall of fame for networks that can be discarded also helps mitigate the issues with repeated validation. If one of the networks in a given generation of the algorithm over-reports its performance and manages to enter the hall of fame, our 3-split validation method and the ability to discard networks from the hall of fame prevents that network from entering the database of effective architectures.

Because the genetic algorithm may get stuck in local optima, it is possible that a single run of the genetic algorithm does not capture an encompassing view of possible effective neural network architectures. In order to understand the variance of the genetic algorithm, we run it on the same data set many times. We also vary some of the genetic algorithm's configurations. Because it takes many runs of the genetic algorithm to analyze the variance of a single change in the configurations, we only test a few changes. Fig. 4.4 shows all of the genetic algorithms configurations and variations that we test.

In the following section, we review the characteristics of every data set and note any data-specific configurations.

Genetic Algorithm Parameters
Hall of Fame Size: 10
Population: 20, 25, 35, 50
Tournament Size: 2, 4, 5
Global Crossover Probability: 0.25, 0.3, 0.4
Global Mutation Probability: 0.8, 0.9
Number of Generations: 25, 30, 40, 45, 50, 55
Training Epochs: Data Dependent

Initialization Parameters
Dense Neuron Minimum: 4, 8
Dense Neuron Maximum: 24, 32
Conv1D Neuron Minimum: 4, 8
Conv1D Neuron Maximum: 24, 32
Conv1D Receptive Field Minimum: 2
Conv1D Receptive Field Maximum: 6
Conv1D Stride Minimum: 1
Conv1D Stride Maximum: 2
Conv2D Neuron Minimum: 4, 8
Conv2D Neuron Maximum: 24, 32
Conv2D Receptive Field Minimum: 2
Conv2D Receptive Field Maximum: 4
Conv2D Stride Minimum: 1
Conv2D Stride Maximum: 2
Batch Size Minimum: 1
Batch Size Maximum: 128
Number of Layers Minimum: 1
Number of Layers Maximum: 4

Mutation Probabilities
Dense Neuron: 1.0
Dense Activation: 0.5, 0.75
Conv1D Neuron: 1.0
Conv1D Receptive Field: 0.5, 0.9
Conv1D Stride: 0.5, 0.75
Conv1D Activation: 0.5, 0.75
Conv2D Neuron: 1.0
Conv2D Receptive Field: 0.5, 0.9
Conv2D Stride: 0.5, 0.75
Conv2D Activation: 0.5, 0.75
Batch Size: 0.5
Add Layer: 0.7, 0.75
Delete Layer: 0.65, 0.7, 0.8

Mutation Boundaries
Dense Neuron Minimum: -10, -15
Dense Neuron Maximum: 10, 15
Conv1D Neuron Minimum: -10, -15
Conv1D Neuron Maximum: 10, 15
Conv1D Receptive Field Minimum: -2
Conv1D Receptive Field Maximum: 2
Conv1D Stride Minimum: -1
Conv1D Stride Maximum: 1
Conv2D Neuron Minimum: -10, -15
Conv2D Neuron Maximum: 10, 15
Conv2D Receptive Field Minimum: -2
Conv2D Receptive Field Maximum: 2
Conv2D Stride Minimum: -1
Conv2D Stride Maximum: 1
Batch Size Minimum: -10
Batch Size Maximum: 10

Figure 4.4: The full list of genetic algorithm configurations and tested values.

Chapter 5: Data

We apply our genetic algorithm to four different data sets: MNIST handwritten digits [62], CIFAR10 small image classification [63], IMDB movie reviews sentiment classification[1], and Reuters newswire topics classification[2]. In order to compare within and across data set types, we select two image classification problems (MNIST and CIFAR10) and two text classification problems (IMDB and Reuters). We train our neural networks using a cluster of 4 NVIDIA Tesla P4s[3]. Table 5.1 displays the amount of computation time, in days, we dedicate to each data set. We allocate the time qualitatively depending on various factors such as the size of data, variance of the genetic algorithm's output, and the number of configurations that we want to test. In total, we spend 107.77 GPU days evolving 980 neural networks across four data sets.

MNIST is an image classification data set with 60,000 training images and 10,000 test images. The images are 28x28x1 grayscale pictures of handwritten digits. There are 10 classes, one for each digit from 0 - 9. For training, we test 3 and 4 epochs.

CIFAR10 is an image classification data set with 50,000 training images and 10,000 test images. The images are 32x32x3 color pictures of various objects. There are 10 classes, one for each type of object: airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck. For training, we test 3 and 4 epochs.

IMDB is a text classification data set with 25,000 training movie reviews and 25,000 test movie reviews. Each review is encoded as a sequence of words. The reviews are labeled as either positive or negative, representing the general sentiment of the review. For training, we test 4, 6, 10, and 15 epochs. Additionally, we have to make a decision about formatting

[1] https://keras.io/datasets/#imdb-movie-reviews-sentiment-classification
[2] https://keras.io/datasets/#reuters-newswire-topics-classification
[3] https://images.nvidia.com/content/pdf/tesla/184457-Tesla-P4-Datasheet-NV-Final-Letter-Web.pdf

Table 5.1: A table of total genetic algorithm runtimes across each data set. The measurements are in GPU days, which is the sum of computation time across all active GPUs.

Data	**GPU Days**	**Number of Evolved Networks**
MNIST	15.94	250
CIFAR10	26.95	320
IMDB	24.83	205
Reuters	40.05	205
Total	107.77	980

the input data. Because the IMDB data is formatted as a sequence of words, we need a computationally tractable representation for the neural networks. In order to do this, we choose to use a word embedding that is learned at training time, similar to Word2Vec [64]. A word embedding represents words as n-dimensional vectors, allowing real-valued computation on those words. Although there are other methods to encode words, word embeddings have demonstrated success in recent literature [65, 66]. We test embedding dimensions of 16 and 32. We also limit the max length of the input sequence to 256 words.

Reuters is a data set of 8,982 training newswires and 2,246 test newswires. Like IMDB, the text is encoded as a sequence of words. The newswires are given a label from 46 possible topics. We do not enumerate the topics for brevity. For training, we test 8, 10, 12, and 15 epochs. Like IMDB, the Reuters data set is formatted as a sequence of words. We use word embeddings and test an embedding dimension of 32. We also limit the max length of the input sequence to 256 words.

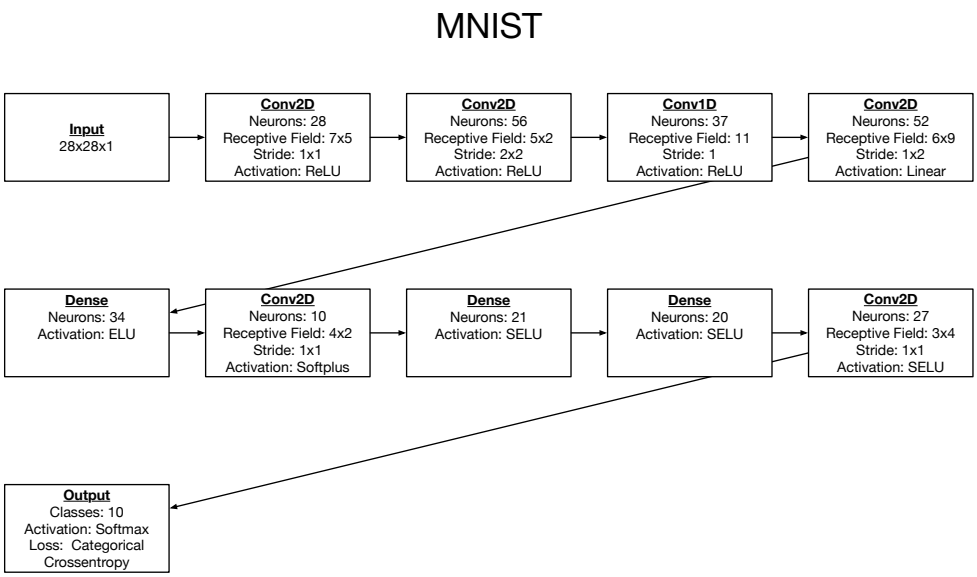
Chapter 6: Results

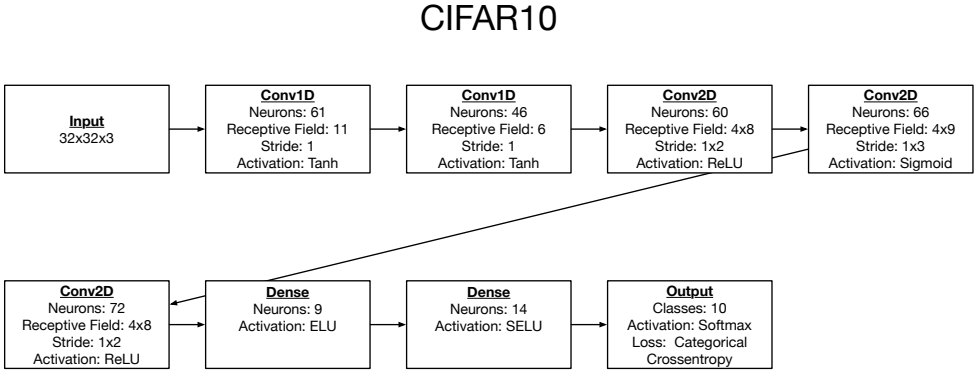
6.1 Overall Results

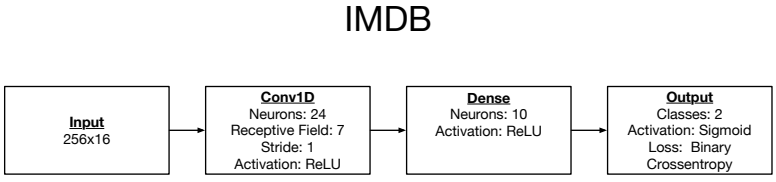
Table 6.1 displays the average validation accuracies for each data set's evolved networks. We compute this by using 10 epochs of training and the original validation sets that come with the data sets. At 10 training epochs, our models are likely undertrained compared to state-of-the-art results, but it allows us to overcome the computational requirements of validating 980 evolved neural networks. We include state-of-the-art results on the table, but note that we do not perform any data transformation, augmentation, or modification that is critical to obtaining state-of-the-art performance. Deng [62] discusses how, for the MNIST data set, there is a severe increase in error rate without the use of data augmentation techniques such as elastic distortion [67]. Enhancing the data is common for all of our data sets, but searching across augmentation techniques is outside the scope of our work. Additionally, we download the Reuters data set directly from Keras, which performs a unique pruning of the data. The state-of-the-art performance does not directly compare to our average accuracy; we display the accuracy for the closest variations of the Reuters data set that we are able to locate.

Regardless, we display the highest accuracy architectures for MNIST, CIFAR10, IMDB, and Reuters in Fig. 6.1. Each box represents a neural network layer, containing all the layer specific parameters. We include the input and output layers of each network for clarity.

In the following sections, we analyze the evolved networks in detail. It is vital that, regardless of the evolved neural network architecture, we always attach an appropriate output layer to every neural network. Because it is not possible to perform classification without the appropriate output layer, it is not counted in any of the following analysis. For MNIST, we use a dense output layer with 10 classes, a softmax activation function,







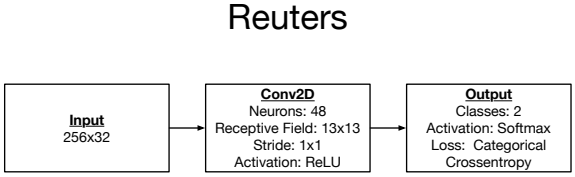


Figure 6.1: The highest accuracy neural network architectures from each data set.

Table 6.1: Average validation performance of each data set's evolved neural networks. We include state-of-the-art results on the table, but note that we do not perform any data transformation, augmentation, or modification that is critical to obtaining state-of-the-art performance. Additionally, we only train the models for 10 epochs in order to limit the computation time required to validate 980 neural networks.

Data	**Average Accuracy**	**Highest Accuracy**	**State-of-the-Art**
MNIST	98.98	99.36	99.79 [68]
CIFAR10	65.62	71.93	96.53 [69]
IMDB	87.42	89.38	95.40 [70]
Reuters	73.09	76.67	82.61 - 93.76 [71, 72]

categorical crossentropy loss, and the ADADELTA optimizer [73]. CIFAR10 uses the same output layer as MNIST. For IMDB, we use a dense layer with 1 node, a sigmoid activation function, binary crossentropy loss, and the ADADELTA optimizer. For Reuters, we use a dense layer with 46 classes, a softmax activation function, categorical crossentropy loss, and the ADADELTA optimizer.

6.2 Layer Types

First, we examine layer occurrence based on our four data sets. Fig. 6.2 shows the total layer distributions for MNIST, CIFAR10, IMDB, and Reuters. We compute these by counting the layer types (dense, conv1D, conv2D) across all layers of every evolved neural network. Fig. 6.3 digs into even greater detail and displays the layer distributions on a network-depth basis.

For the image classification problems, we see very similar total layer distributions in Fig. 6.2a and Fig. 6.2b. The convolutional 2D layer is the most common layer in both problems, the dense and convolutional 1D layers have similar frequencies in MNIST, and the convolutional 1D layer is slightly more common in CIFAR10. Hand-designed neural networks for image classification commonly use many convolutional 2D layers followed by dense layers. This design paradigm stems from the belief that the convolutional 2D layers act as feature extractors for the image, and the dense layers then combine those features

to perform the classification. If we examine Fig. 6.3a and Fig. 6.3b, we see that for both MNIST and CIFAR10, convolutional 2D layers are extremely common in the early layers of a network, while the frequency of dense layers is much higher in the later layers of a network. Although this confirms the effectiveness of traditional hand-designed architectures for image classification, the distribution of convolutional 1D layers is unexpected and perplexing. The frequency of convolutional 2D layers clearly declines as networks get deeper, but the frequency of convolutional 1D layers hovers around 20%-30%. Convolutional 1D layers, in general, are rarely ever utilized or even discussed in hand-designed image classification networks. Despite that, a significant amount of evolved networks perform well with convolutional 1D layers. We hypothesize that convolutional 1D layers effectively continue to utilize spatial locality in the data, even after severe flattening of the data's second dimension. In Chapter 2, we explain how convolutional 2D layers can end up flattening the second dimension until convolutional 2D layers are rendered useless. Due to the convolutional 1D layer's agnosticism towards the second dimension of the data's shape, it is possible that convolutional 1D layers allow one to extend a network's depth and thus spend additional time performing spatially sensitive feature extraction on the data. Prior work does not discuss this design pattern, which may merit further evaluation.

The text classification problems have vastly different total layer distributions in Fig. 6.2c and Fig. 6.2d compared to both each other and the image classification problems. The text classification problems both make very little use of dense layers, but their relative frequencies of convolutional 1D and convolutional 2D layers are flipped. The evolved networks for IMDB fall in line with expected hand-designed architectures. It is common to treat a sequence of words as fundamentally one dimensional data, despite the fact that word embeddings introduce a second dimension to the data's shape. Although Reuters also frequently makes use convolutional 1D layers, there are many evolved networks that prefer convolutional 2D layers. We hypothesize that the 46 output classes in the Reuters data set have caused the neural networks to distribute information along the second dimension of the embedding in a different way than IMDB distributes the information. This could

potentially make convolutional 2D layers more effective at differentiating between classes by increasing utilization of spatial locality in the second dimension of the word embeddings.

6.3 Network Depth

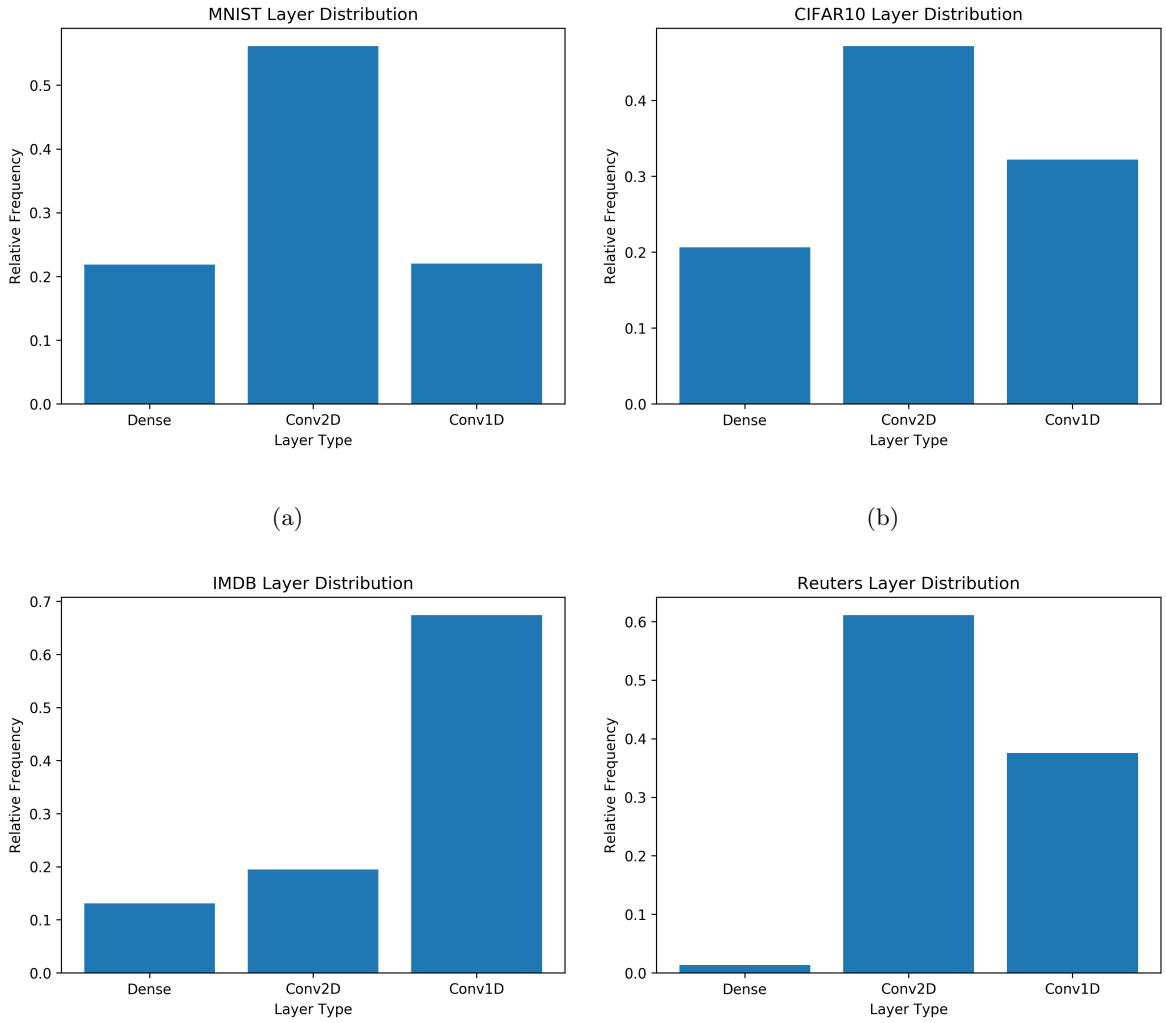
Another common point of inquiry in architecture design is network depth. Based on discussions in Chapter 3, there has been success with both shallow, wide networks and very deep networks. Fig. 6.4 shows the distributions of network depths across our different data sets.

We find that for both image classification problems, the evolution favors networks that are 4-8 layers deep, as shown in Fig. 6.4a and Fig. 6.4b. This reflects hand-designed architectures that frequently use 2-4 convolutional 2D layers followed by 2-4 dense layers in order to perform image classification. For the text classification problems, the genetic algorithm favors very shallow networks. Fig. 6.4c shows that most of the evolved networks for IMDB are 1-3 layers deep, while Fig. 6.4d shows that Reuters in particular prefers only 1 layer (recall that there is a dense output layer for every single network in order to properly generate output predictions).

The evolved networks never reach extremely high layer depths. Because many extremely deep networks require careful curation of layer sizes, activation function choices, and computational time, the genetic algorithm is unlikely to locate these solutions via randomization and limited computational time.

6.4 Neurons

Having seen the distributions of network depths and the types of layers that populate those depths, we now examine the the sizes of each layer (i.e. the number of neurons). Fig. 6.5 shows the distributions of neurons per layer across all networks in each data set. Remarkably, we find the regardless of layer type, the distribution of the number of neurons is very similar. The graphs resemble a right skewed (or positively skewed) distribution. The trend is very obvious for the image classification problems in Fig. 6.5a and Fig. 6.5b, and



(a) (b)

(c) (d)

Figure 6.2: The total layer distributions for each data set. The image classification problems (MNIST and CIFAR10) have similar layer distributions, while the text classification problems (IMDB and Reuters) differ significantly.

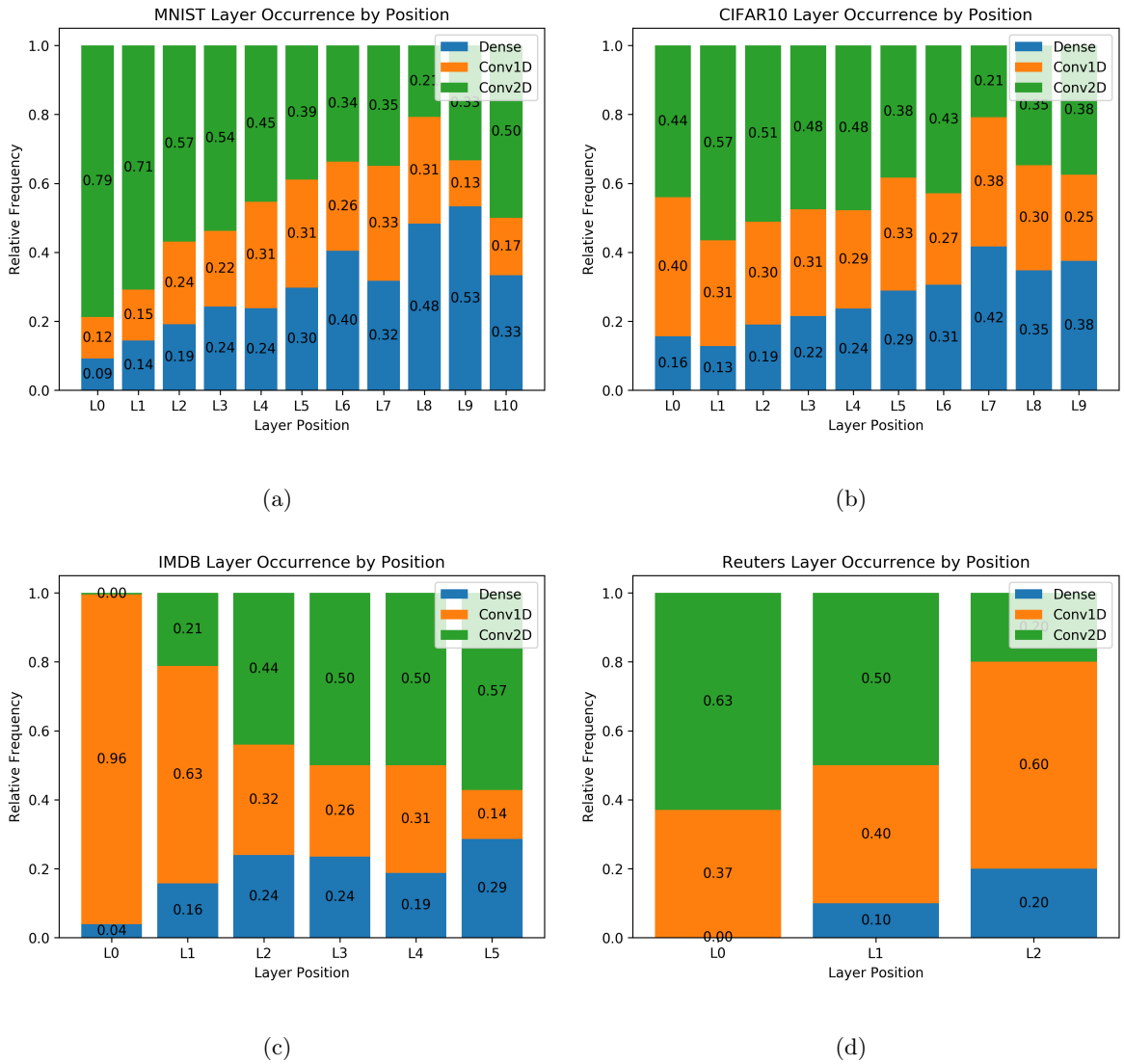
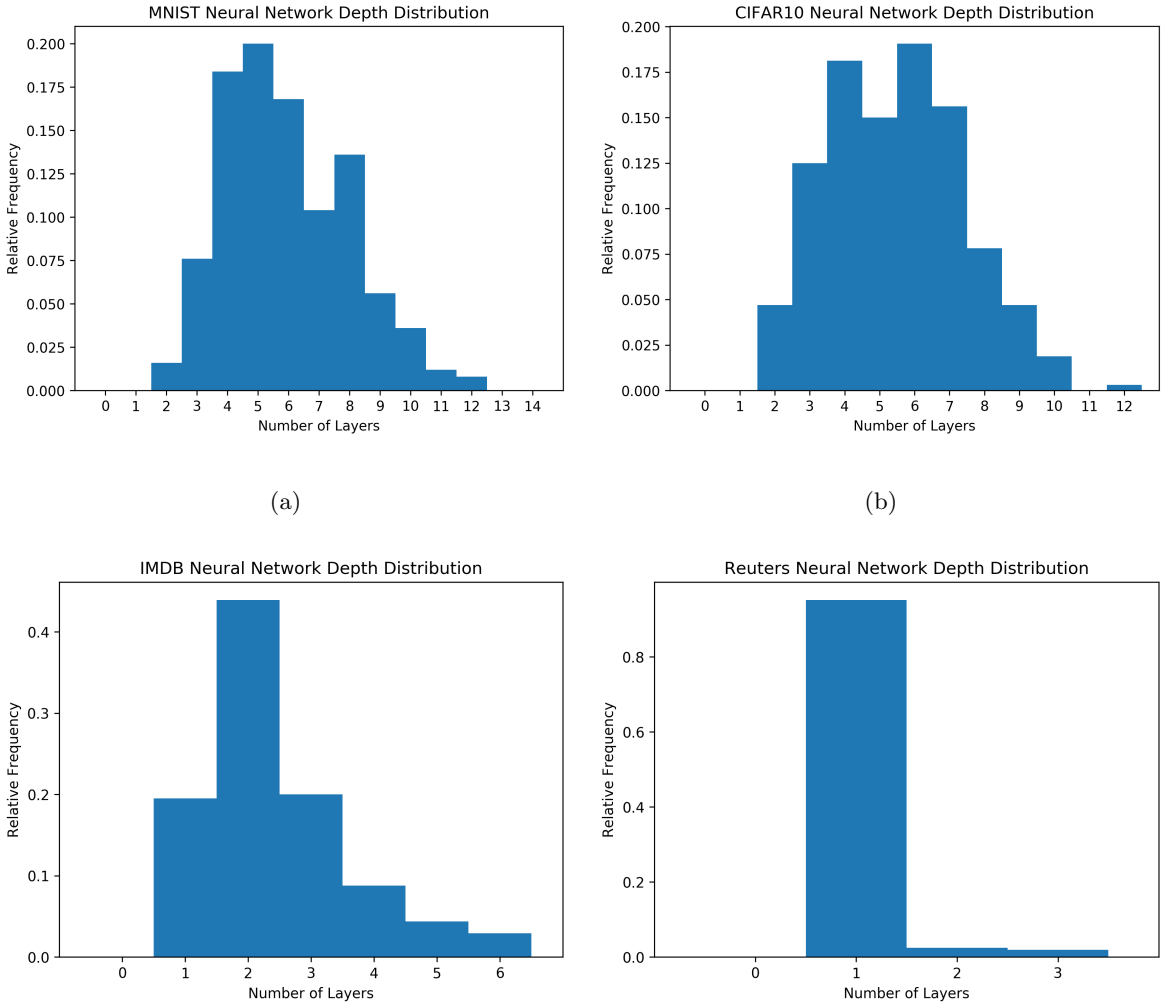


Figure 6.3: The distributions of layer types based on the layer's position in the neural network. The x-axis is the depth of the network and the y-axis is the relative frequencies of each layer at a given depth. Layer depths with 3 or fewer evolved networks are excluded.



(a) (b)

(c) (d)

Figure 6.4: The distributions of network depths across all networks in each data set.

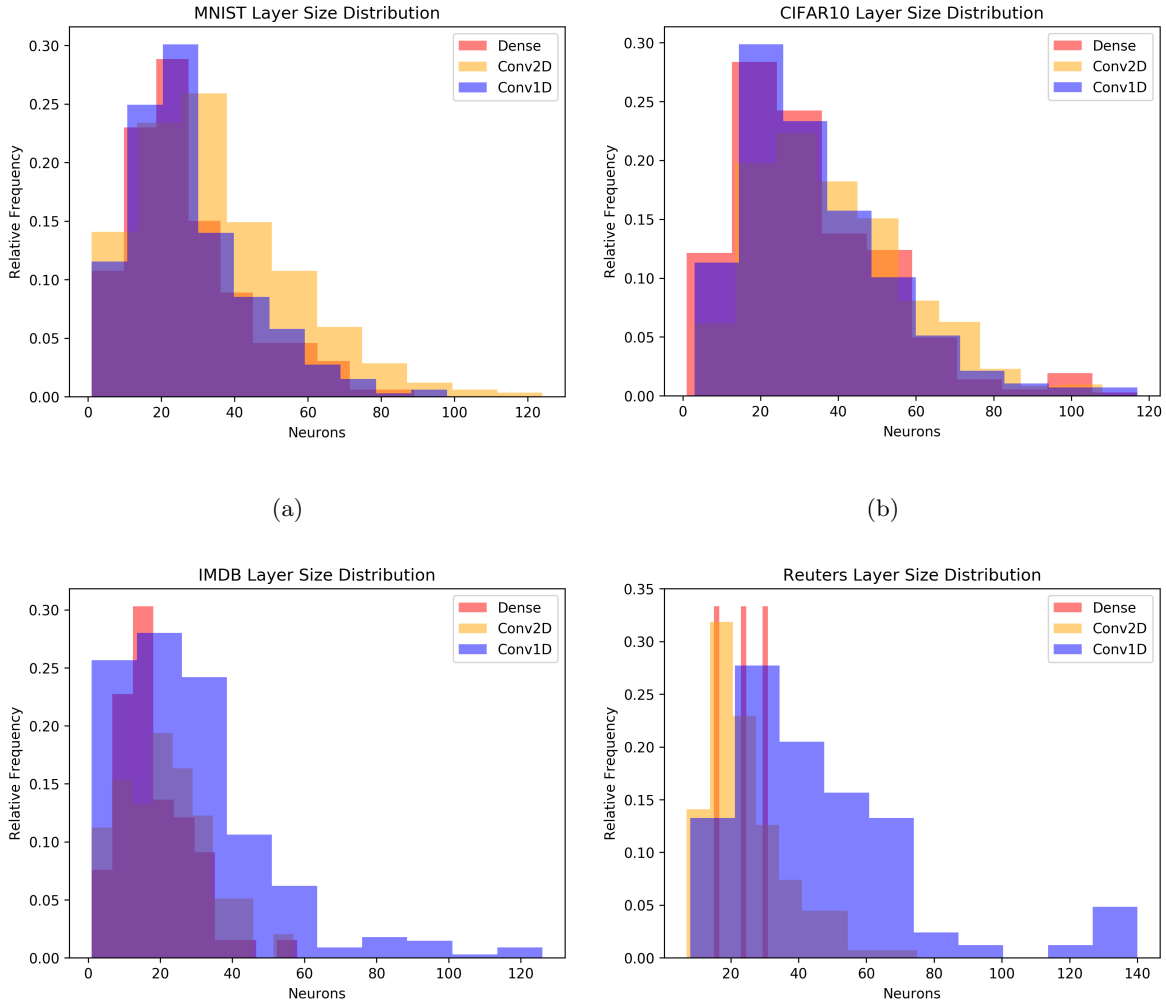
still generally true for the text classification problems in Fig. 6.5c and Fig. 6.5d. Although only 3 of the 205 networks for Reuters evolve a dense layer, the number of neurons for those dense layers still occur around the median value.

One of the most interesting results from the neuron distributions is that the number of neurons for dense layers and convolutional layers is very similar. Due to striding and the limited size of convolutional receptive fields, a convolutional neuron has much fewer parameters than a dense neuron. The neuron distributions indicate that it is not necessary to increase the number of convolutional neurons in order to compensate for the reduced number of parameters. Although the actual number of neurons likely depends on the complexity of the data and the classification problem, setting the number of convolutional neurons to a similar value as the dense neurons is a valuable design pattern.

6.5 Activation Functions

The choice of activation functions is a common issue in neural network design. Fig. 6.6 displays the distribution of activation function from every layer of evolved networks for each data set. Fig. 6.7 delves into more detail and displays the relative frequencies of the activation functions at different network depths. Across all the data sets, ReLU is a clear favorite, which is in line with many hand-designed architectures and prior work in analyzing activation functions. The figures do, however, show a distinction between the image and text classification problems. The image classification problems in Fig. 6.6a and Fig. 6.6b make occasional use of the other activation functions except sigmoid and softplus, which are rarely selected in the evolved networks. The text classification problems in Fig. 6.6c and Fig. 6.6d prefer ReLU by a very large margin.

An interesting trend that appears for the image classification problems when looking at the distributions of activation functions across network depths, shown in Fig. 6.7a and Fig. 6.7b, is that there is a decline in ReLU activations as the networks get deeper. If we combine that with our previous observation that early layers are predominantly convolutional 2D and later layers are mostly dense, the distribution of activation functions across network depths



(a) (b)

(c) (d)

Figure 6.5: The distributions of neurons per layer across all networks in each data set.

may suggest that the ReLU activation is the strongest choice for convolutional 2D layers, but not necessarily so for dense layers. Mixing convolutional layers with ReLU activations and dense layers with other activation functions is a potential design pattern that has not been discussed in prior work and may merit further investigation.

6.6 Receptive Fields

Although convolutional layers are well known for their performance on data with spatial locality, there is still a question of how much locality should be captured in a convolutional neuron. Fig. 6.8 shows the distributions of convolutional 1D receptive field sizes. The general shape of the distributions is similar across all of the data sets, but the centers of the distributions are not. The image classification problems in Fig. 6.8a and Fig. 6.8b are centered around smaller receptive field sizes, while the text classification problems in Fig. 6.8c and Fig. 6.8d are centered around larger receptive field sizes. There are several potential explanations for this phenomenon. It is possible that the one dimensional locality of the image classification problems is less important than the one dimensional locality of the text classification problems. Alternatively, the fact that the image classification problems evolve deeper networks may allow multiple consecutive layers of convolutions to broaden its convolutional locality while limiting the number of parameters in the network.

Fig. 6.9 shows the distributions of convolutional 2D receptive field sizes. The differences between distributions for the first and second dimensions of the receptive field are a peculiar finding. For every data set except CIFAR10, the second dimension of the receptive field is significantly smaller than the first dimension. This phenomenon is still much stronger in both text classification problems than it is in MNIST, however. The reason for this phenomenon in MNIST is not entirely clear. One potential explanation that can be garnered from examining the data is that there are many images that contain many columns with no information. Is it possible that this reduces the efficacy of a larger receptive field in the second dimension. The smaller size of the receptive field's second dimension in the text classification problems is reasonable, though perhaps unexpected. Although the word

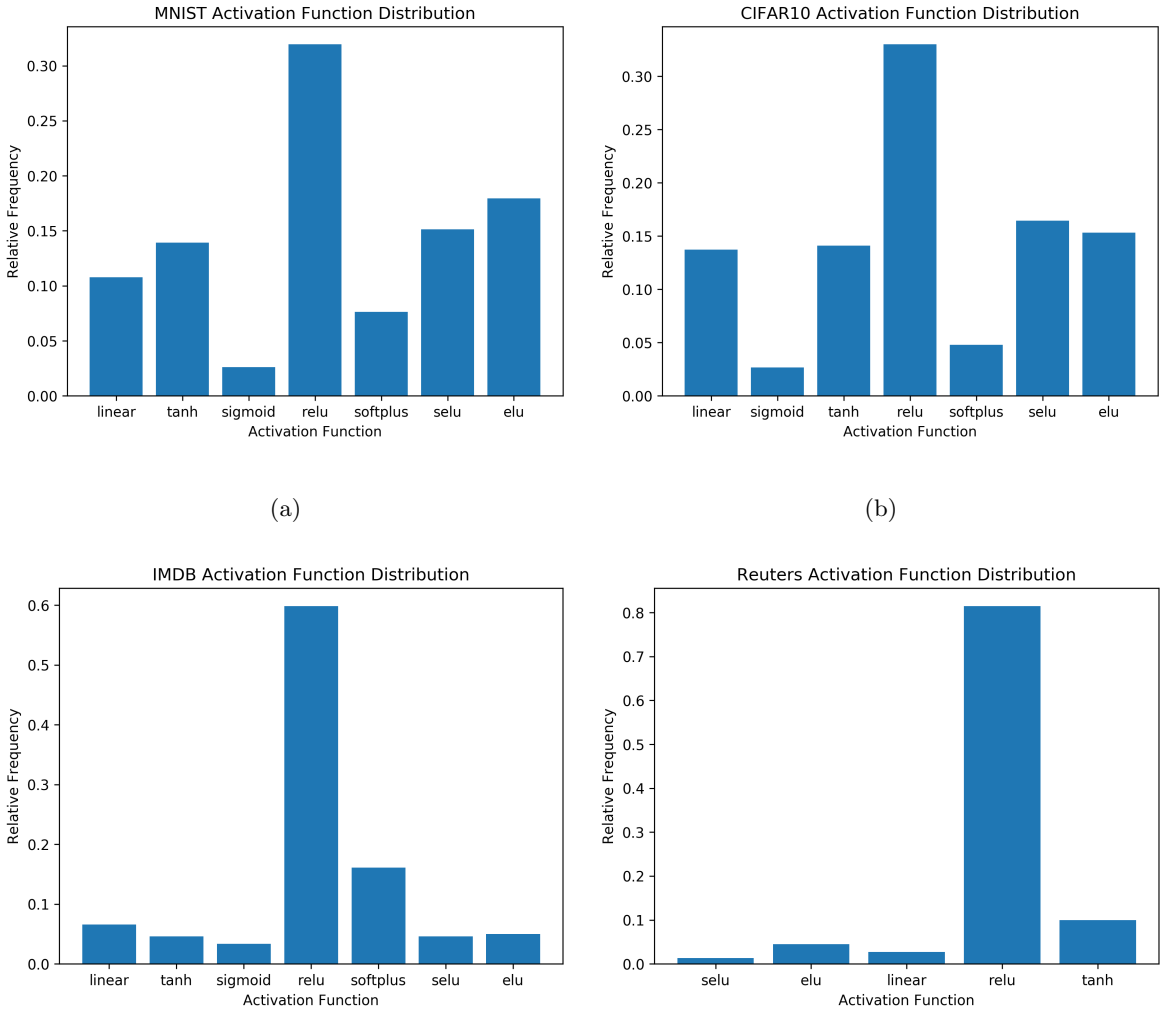


Figure 6.6: The total activation function distributions for each data set.

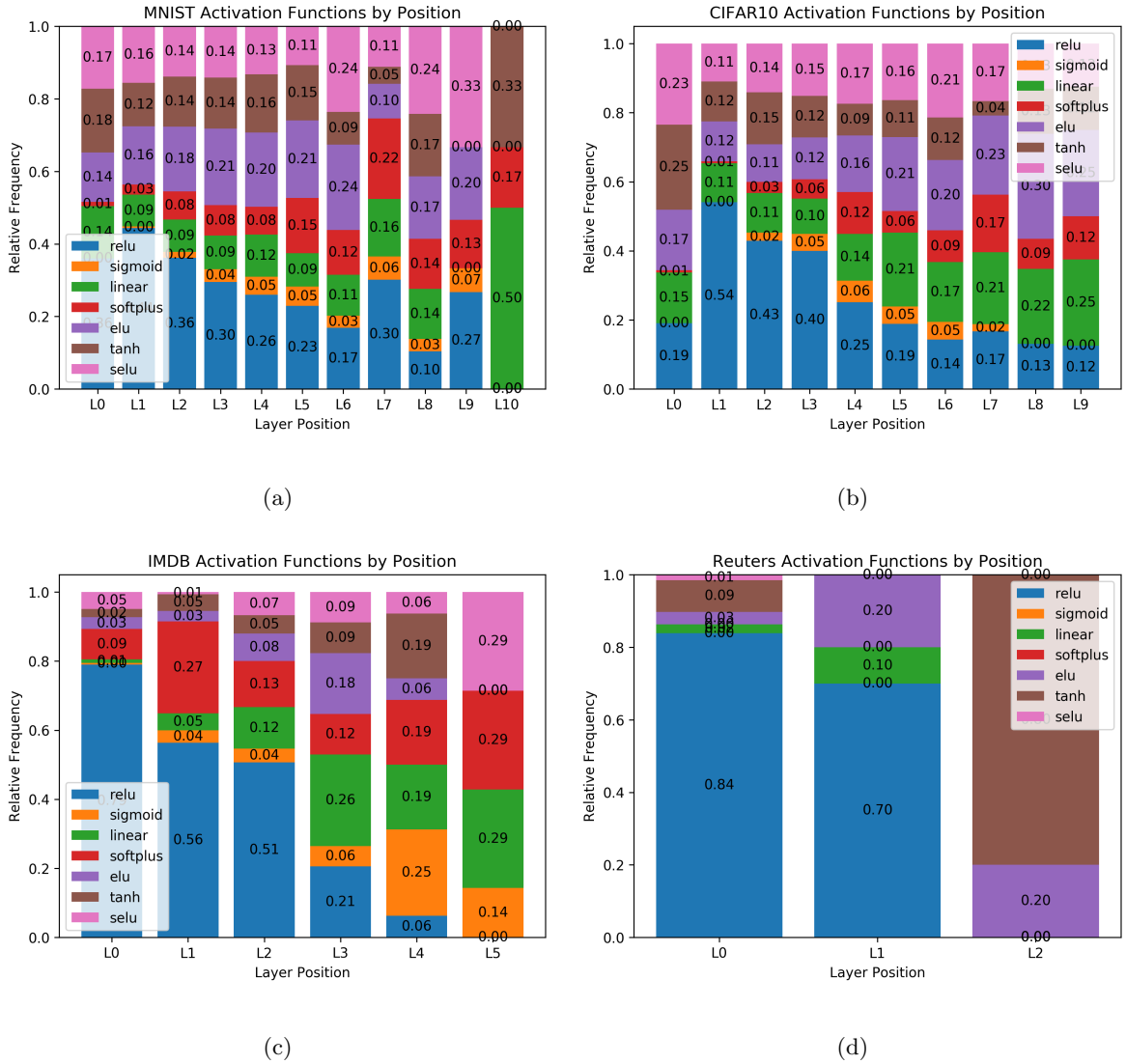


Figure 6.7: The distributions of activation functions based on a layer's position in the neural network. The x-axis is the depth of the network and the y-axis is the relative frequency of each activation function at a given depth. Layer depths with 3 or fewer evolved networks are excluded.

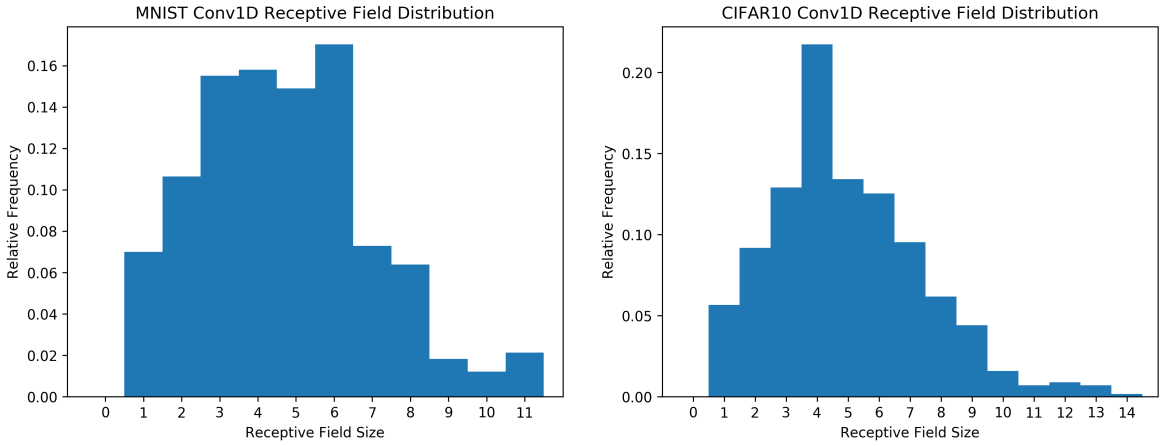
embeddings add a second dimension to the data, the original data – the sequence of words – is one dimensional. Additionally, the size of the second dimension is much smaller than the first dimension even with the word embedding. The input sequence may be up to 256 words, while the embedding dimension is 16 or 32. Because the use of convolutional 2D layers in text classification problems is fairly uncommon, this design pattern is rarely, if ever, discussed in prior work.

6.7 Stride Lengths

With the receptive field sizes in mind, we examine the stride lengths of the convolutions next. Fig. 6.10 displays the stride lengths of the convolutional 1D neurons, and Fig. 6.11 displays the stride lengths of the convolutional 2D neurons. The distributions of stride lengths are extremely similar for every convolution across all data sets. The convolutional 1D strides and both dimensions of the convolutional 2D strides heavily favor a stride length of 1 or 2. The stride length appears to be a case where the absence of data says more than the data itself. For every data set and every type of convolution, there are no long stride lengths relative to the size of the receptive fields. This indicates that setting the stride length to 1, regardless of the type of convolution, is an effective choice. This is a design pattern that has not been otherwise explored in prior work.

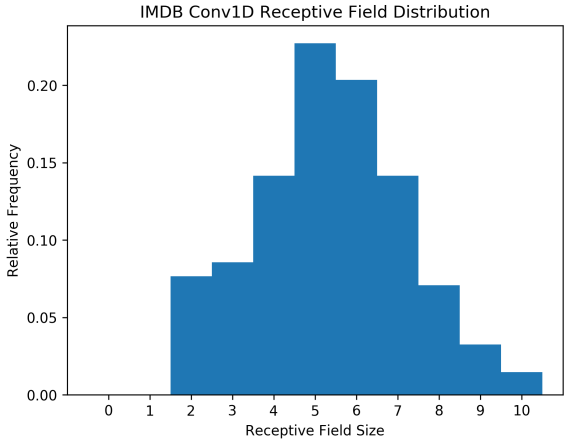
6.8 Batch Size

The decision of a network's batch size is often based on computational limits, but the batch size impacts a network's learning trajectory as well. Smaller batches utilize less memory and update the network's weights more frequently, but each update is a less accurate estimate of the gradient. Larger batch sizes do not suffer from as many fluctuations in the gradient estimates. Fig. 6.12 shows the distribution of batch sizes for each data set's evolved networks. The batch size distributions indicate that the genetic algorithm has a preference for certain batch sizes because our network initialization procedure selects an initial batch

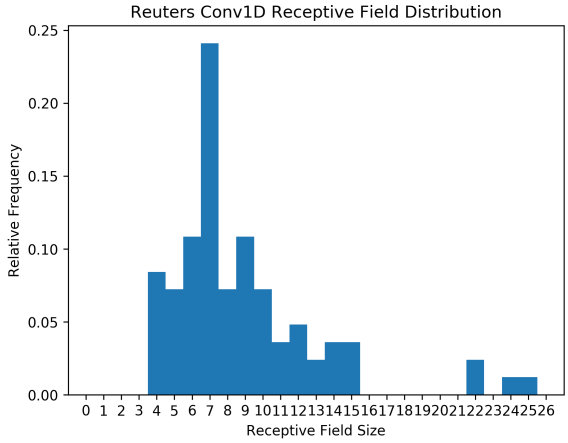


(a)

(b)



(c)



(d)

Figure 6.8: The distributions of convolutional 1D receptive field sizes for each data set.

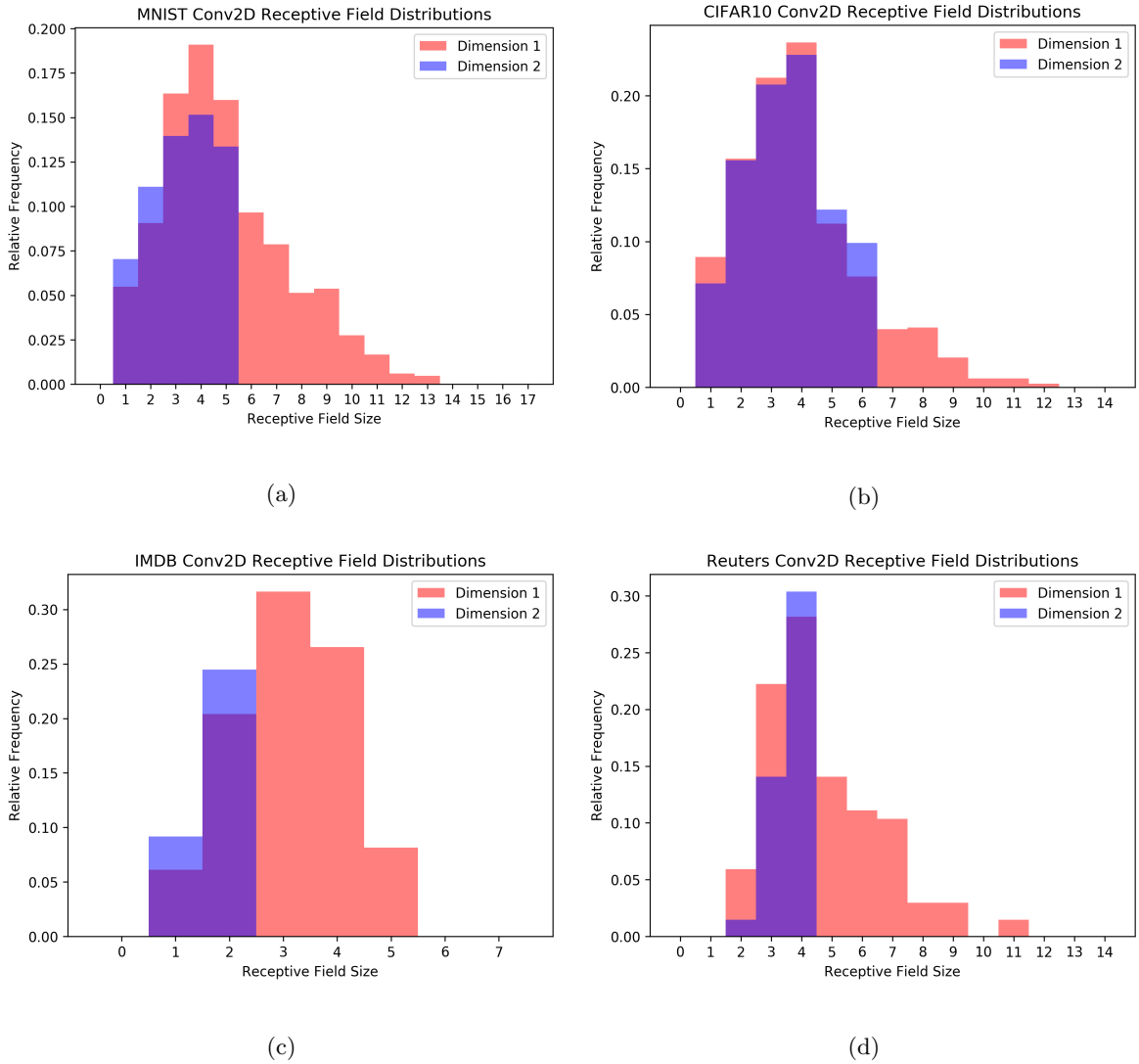
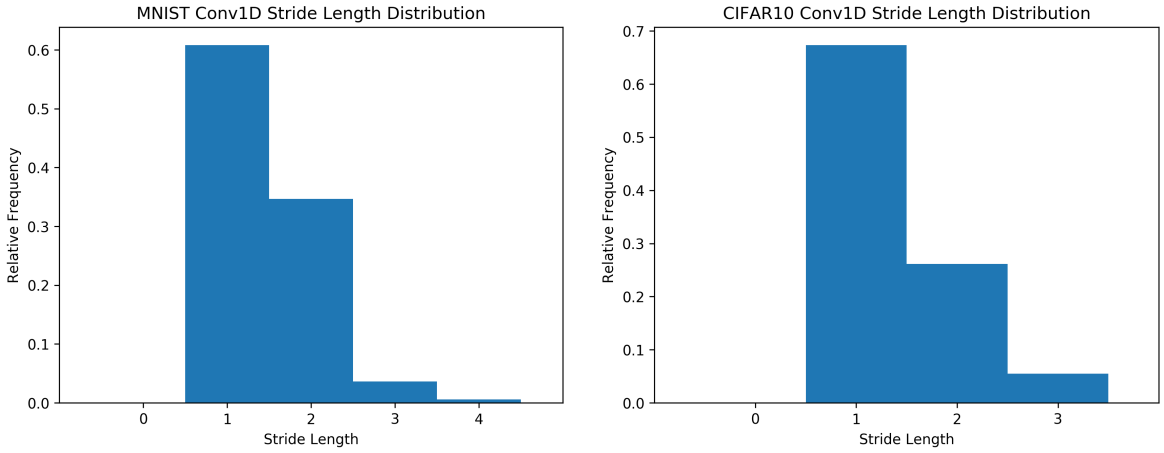
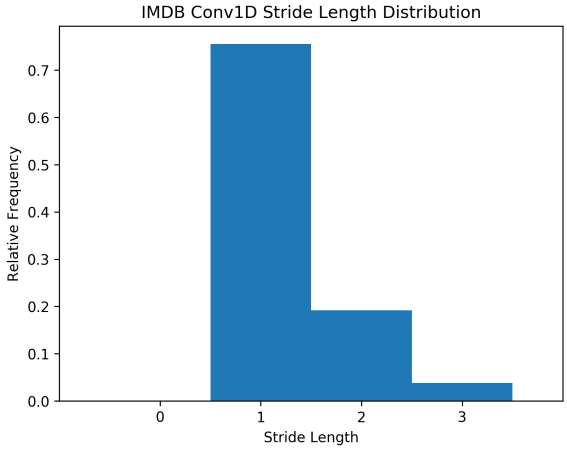


Figure 6.9: The distributions of convolutional 2D receptive field sizes for each data set. We display the two dimensions of the receptive fields separately.

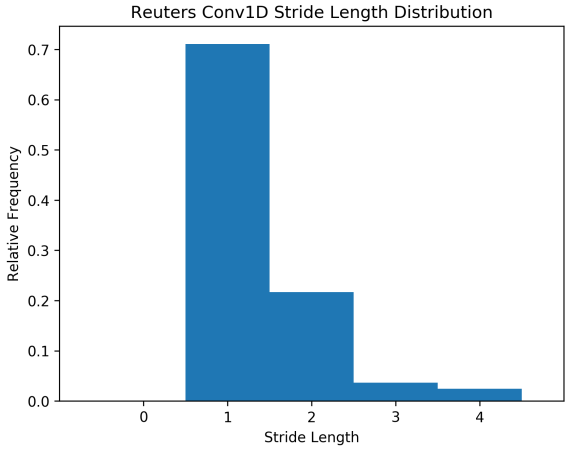


(a)

(b)

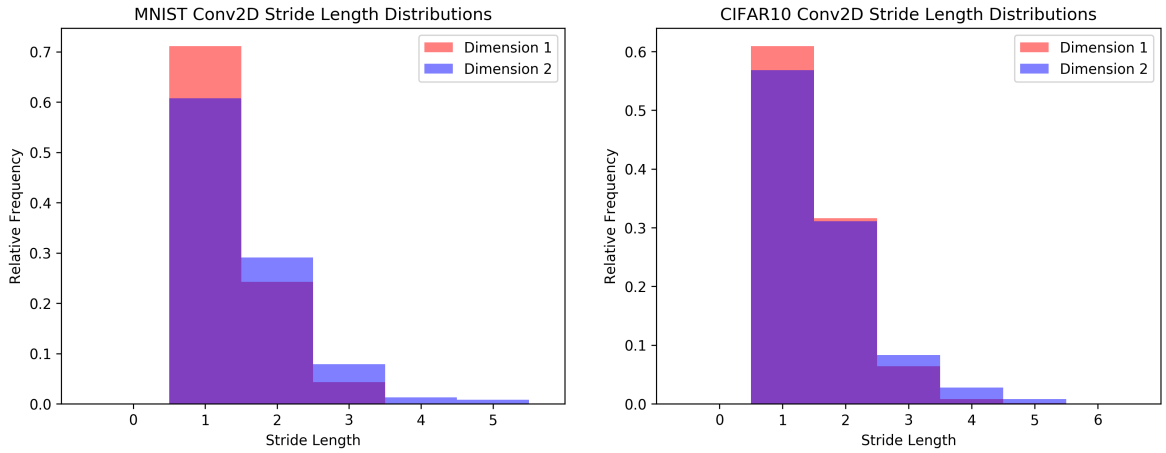


(c)



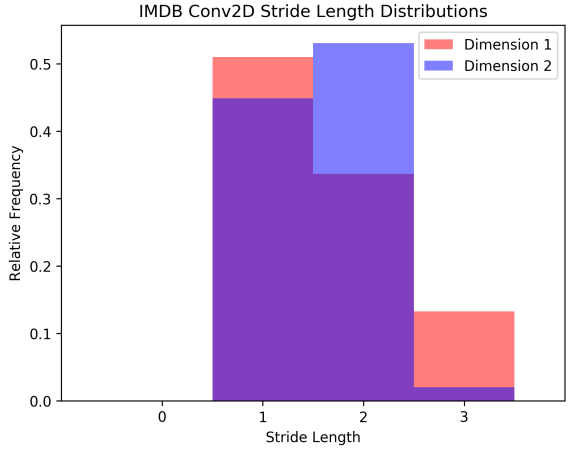
(d)

Figure 6.10: The distributions of convolutional 1D stride lengths for each data set.

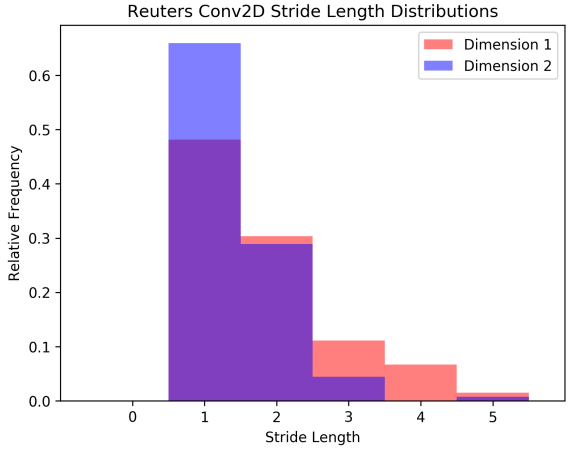


(a)

(b)



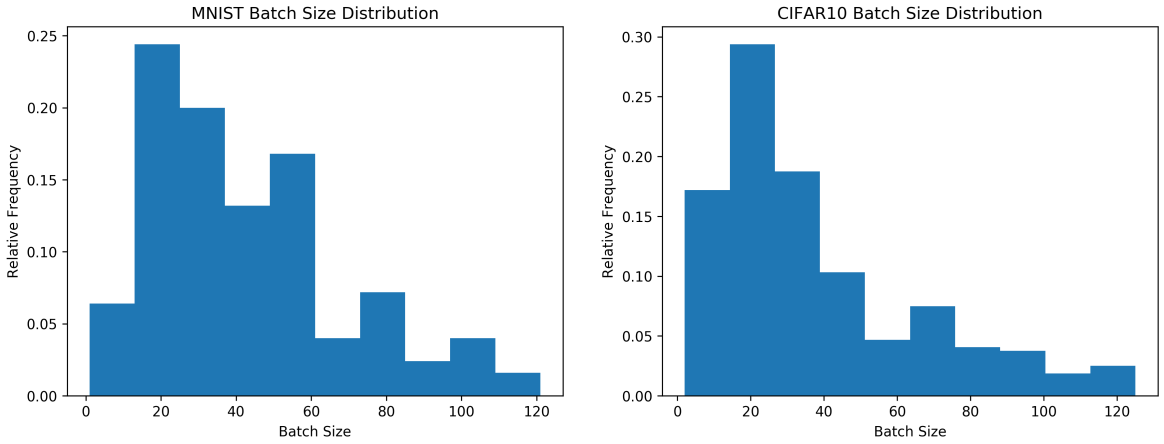
(c)



(d)

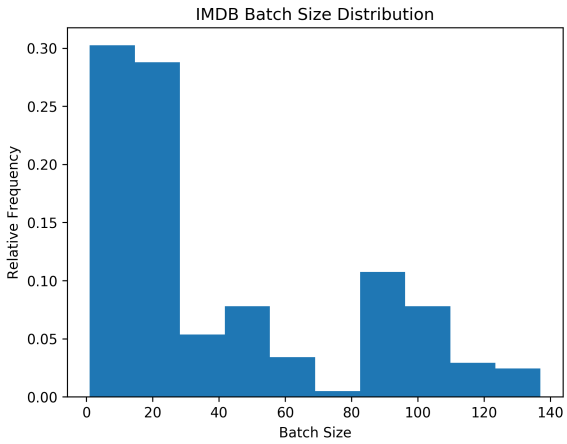
Figure 6.11: The distributions of convolutional 2D stride lengths for each data set. We display the two dimensions of the stride length separately.

size from 1 to 128 with a uniform random chance. It is not entirely clear, however, what criteria motivates the genetic algorithm's preference. A possible cause is the quantity of available data. MNIST has 60,000 training samples, CIFAR10 has 50,000 training samples, IMDB has 25,000 training samples, and Reuters has 8,982 training samples. The median batch sizes for MNIST, CIFAR10, IMDB, and Reuters are 36, 30, 23, and 11, respectively. If we divide the number of samples by the batch size, we obtain the number of weight updates per training epoch. The number of updates per epoch using the median batch sizes for MNIST, CIFAR10, IMDB, and Reuters are 1,667, 1,667, 1,087, and 817, respectively. Interestingly, the image and text classification problems each tend to evolve networks with similar numbers of updates per epoch. Keskar et al. [74] investigate the effect of batch size on validation performance for MNIST, TIMIT [75], and CIFAR. They find that very large batch sizes do cause significant drops in performance, but their experiments do not estimate the number updates per epoch. Our results suggest a relationship between available training data, batch size, and problem type, which may merit further investigation.

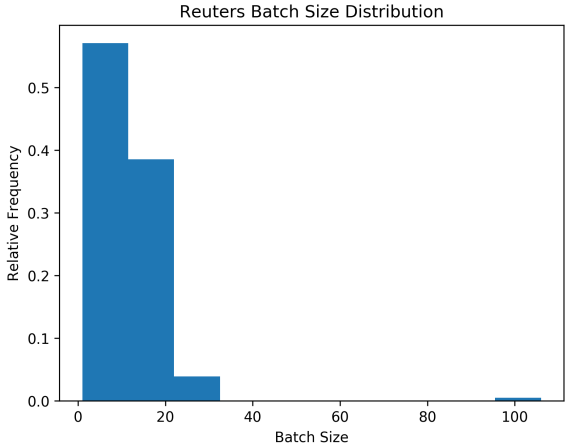


(a)

(b)



(c)



(d)

Figure 6.12: The batch size distributions for each data set.

Chapter 7: Conclusion

In this work, we genetically evolve neural networks across a variety of data sets in order to gain insights into effective neural network architectures. We induce design patterns from 980 evolved networks across 4 data sets. We reiterate a few of our design patterns here. Our layer type analysis shows that the output required by the classification problem may be a strong indicator of layer type because the text classification data sets have similar inputs, significantly different ouputs, and two distinct layer type distributions. The network depth analysis shows that the input data has a large impact on the network's depth because the image classification problems favor 4-8 layers, while the text classification problems favor 1-2 layers. The neuron distributions demonstrate that having a similar number of neurons in convolutional and dense neurons is an effective design, despite the fact that convolutional neurons have fewer parameters. The activation function charts indicate that ReLU is better in most cases, while sigmoid and softplus should be actively avoided. The receptive field analysis suggests that the second dimension of a convolutional 2D layer does not always have to be square with the first dimension, and that it is actually beneficial to modify a single dimension of the receptive field's size with respect to the input data's shape and characteristics. The stride length investigation demonstrates that a stride length of 1 is an effective choice regardless of the type of convolutional layer, and that very long stride lengths are rarely a good choice. The batch size distributions suggests that batch size may scale with the quantity of available training data and the type of classification problem.

By repeating the network evolution on the same problems hundreds of times, we are able to uncover the consistency of design patterns, which is a problem that has been rarely attempted in prior work. Although maximizing the performance of a model on a single problem may involve very specific adaptations, consistent design patterns allow one to formulate effective architectures without vast computational resources or significant experience

in hand-designing models. The empirical nature of consistent design patterns also enhances the rigor of neural network architectural design in general.

7.1 Challenges and Limitations

Inevitably, we encounter several challenges and limitations throughout the work. The first limitation of computation time is clear. Collecting more information about evolved networks via repeated runs of the genetic algorithm, new types of data sets, and various genetic algorithm parameters will always be beneficial to the breadth and consistency of uncovered design pattern patterns. Selecting where to allocate computational time will likely remain a challenge in design pattern discovery.

Another limitation is the scope of our evolved network analysis. For the most part, we assume that each network characteristic in Chapter 6 is independent, but this is likely not the case for all combinations of characteristics. Just the permutations of 2 or 3 characteristics introduces an immense amount of possible design patterns. In order to perform this analysis, it may be necessary to use a method for automatically locating and filtering consistent, effective design patterns.

7.2 Future Work

Upon evaluating our evolved networks, we discover several potential avenues to enhance neural network architecture design. In the layer type analysis, we find a previously unexplained design pattern that frequently distributes convolutional 1D layers across the networks of image classification problems. In the activation functions section, we find another unique design pattern that involves mixing ReLU activated convolutions with dense layers that utilize other functions. Although our results in the stride lengths analysis show that a small stride length is effective for any type of convolution, there has not otherwise been any formal results discussing the effects of data, problem type, or convolution type on stride lengths. Lastly, the batch size distributions suggest a relationship between available training data,

batch size, and problem type that may merit further investigation.

Expanding the depth of the design pattern analysis is a valuable extension to this work. It is likely that certain network parameters are related, and changes in one parameter can impact the effectiveness of a different parameter. Developing a methodology to evaluate these interrelationships may help explain more elusive design patterns.

Another possible avenue for future work is the development of an architectural prediction model. By utilizing our database of evolved networks, it is possible to develop a model that accepts a data set's characteristics as input and predicts effective design choices for that data set. By using interpretable models for this task, it may be possible to further clarify the relationships between a data set and its effective design choices.

Bibliography

Bibliography

[1] R. Vargas, A. Mosavi, and R. Ruiz, "Deep learning: A review," *Advances in Intelligent Systems and Computing*, vol. 5, 08 2017.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[4] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," in *Readings in speech recognition*. Elsevier, 1990, pp. 393–404.

[5] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[6] J. Bergstra, F. Bastien, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, D. Warde-Farley, I. Goodfellow, A. Bergeron *et al.*, "Theano: Deep learning on gpus with python," in *NIPS 2011, BigLearning Workshop, Granada, Spain*, vol. 3. Citeseer, 2011, pp. 1–48.

[7] H. Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing, "Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server," in *Proceedings of the Eleventh European Conference on Computer Systems*. ACM, 2016, p. 4.

[8] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," *arXiv preprint arXiv:1703.00548*, 2017.

[9] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, p. 48, 2001.

[10] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao, "Relation classification via convolutional deep neural network," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, pp. 2335–2344.

[11] C. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, pp. 69–78.

[12] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific reports*, vol. 8, no. 1, p. 6085, 2018.

[13] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 4694–4702.

[14] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.

[15] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[16] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *International Workshop on Artificial Neural Networks*. Springer, 1995, pp. 195–201.

[17] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing," in *Advances in neural information processing systems*, 2001, pp. 472–478.

[18] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[19] E. W. Weisstein, "Hyperbolic tangent," 2002.

[20] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, 2017, pp. 971–980.

[21] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh annual conference of the international speech communication association*, 2010.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[24] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[25] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*. Springer, 2010, pp. 92–101.

[26] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Lulu, 2013, available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[27] C. Darwin, *On the origin of species, 1859*. Routledge, 2004.

[28] N. M. Razali, J. Geraghty *et al.*, "Genetic algorithm performance with different selection strategies in solving tsp," in *Proceedings of the world congress on engineering*, vol. 2, no. 1. International Association of Engineers Hong Kong, 2011, pp. 1–6.

[29] D. B. Fogel, "An information criterion for optimal neural network selection," *IEEE Transactions on Neural Networks*, vol. 2, no. 5, pp. 490–497, 1991.

[30] H. Akaike, "Information theory and an extension of the maximum likelihood principle," in *Selected papers of hirotugu akaike*. Springer, 1998, pp. 199–213.

[31] M. Gutierrez, J. Wang, and R. Grondin, "Estimating hidden unit number for two-layer perceptrons," in *IJCNN International Joint Conference on Neural Networks*. Publ by IEEE, 1989, pp. 677–681.

[32] S. K. J. Hwang, "An algebraic projection analysis for optimal hidden units size and learning rates in back-propagation learning," in *IEEE... International Conference on Neural Networks*, vol. 1. SOS Printing, 1988.

[33] C. L. Mallows, "Some comments on c p," *Technometrics*, vol. 15, no. 4, pp. 661–675, 1973.

[34] A. R. Barron, "Predicted squared error, a criterion for automatic model selection," *Self-organizing methods in modeling*, pp. 87–103, 1984.

[35] J. Moody and J. Utans, "Principled architecture selection for neural networks: Application to corporate bond rating prediction," in *Advances in neural information processing systems*, 1992, pp. 683–690.

[36] I. A. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," *Journal of microbiological methods*, vol. 43, no. 1, pp. 3–31, 2000.

[37] M. N. Jadid and D. R. Fairbairn, "Neural-network applications in predicting moment-curvature parameters from experimental data," *Engineering Applications of Artificial Intelligence*, vol. 9, no. 3, pp. 309–319, 1996.

[38] G. Lachtermacher and J. D. Fuller, "Back propagation in time-series forecasting," *Journal of forecasting*, vol. 14, no. 4, pp. 381–393, 1995.

[39] R. Hecht-Nielsen, "On the algebraic structure of feedforward network weight spaces," in *Advanced Neural Computers*. Elsevier, 1990, pp. 129–135.

[40] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[42] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[43] E. Dufourq and B. A. Bassett, "Eden: Evolutionary deep networks for efficient machine learning," in *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*. IEEE, 2017, pp. 110–115.

[44] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2902–2911.

[45] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 497–504.

[46] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International Conference on Machine Learning*, 2015, pp. 2342–2350.

[47] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[48] M. N. Gurcan, H.-P. Chan, B. Sahiner, L. Hadjiiski, N. Petrick, and M. A. Helvie, "Optimal neural network architecture selection: improvement in computerized detection of microcalcifications," *Academic Radiology*, vol. 9, no. 4, pp. 420–429, 2002.

[49] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[50] I. Loshchilov and F. Hutter, "Cma-es for hyperparameter optimization of deep neural networks," *arXiv preprint arXiv:1604.07269*, 2016.

[51] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[52] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *CoRR*, vol. abs/1710.05941, 2017. [Online]. Available: http://arxiv.org/abs/1710.05941

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[54] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[55] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, and A. Roth, "The reusable holdout: Preserving validity in adaptive data analysis," *Science*, vol. 349, no. 6248, pp. 636–638, 2015.

[56] J. P. Ioannidis, "Why most published research findings are false," *PLoS medicine*, vol. 2, no. 8, p. e124, 2005.

[57] J. P. Simmons, L. D. Nelson, and U. Simonsohn, "False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant," *Psychological science*, vol. 22, no. 11, pp. 1359–1366, 2011.

[58] A. Gelman and E. Loken, "The statistical crisis in science," *The best writing on mathematics*, vol. 2015, p. 305, 2016.

[59] A. Y. Ng *et al.*, "Preventing" overfitting" of cross-validation data," in *ICML*, vol. 97, 1997, pp. 245–253.

[60] R. B. Rao, G. Fung, and R. Rosales, "On the dangers of cross-validation. an experimental evaluation," in *Proceedings of the 2008 SIAM International Conference on Data Mining*. SIAM, 2008, pp. 588–596.

[61] B. L. Miller, D. E. Goldberg *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.

[62] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[63] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[64] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[65] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[66] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[67] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *null*. IEEE, 2003, p. 958.

[68] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International conference on machine learning*, 2013, pp. 1058–1066.

[69] B. Graham, "Fractional max-pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[70] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[71] M. Thoma. (2017) The reuters dataset. [Online]. Available: https://martin-thoma.com/nlp-reuters/

[72] E. S. Tellez, D. Moctezuma, S. Miranda-Jiménez, and M. Graff, "An automated text categorization framework based on hyperparameter optimization," *Knowledge-Based Systems*, vol. 149, pp. 110–123, 2018.

[73] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[74] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[75] J. S. Garofolo, "Timit acoustic phonetic continuous speech corpus," *Linguistic Data Consortium, 1993*, 1993.

[76] A. Fiszelew, P. Britos, A. Ochoa, H. Merlino, E. Fernández, and R. García-Martínez, "Finding optimal neural network architecture using genetic algorithms," *Advances in computer science and engineering research in computing science*, vol. 27, pp. 15–24, 2007.

[77] I. Kanellopoulos and G. G. Wilkinson, "Strategies and best practice for neural network image classification," *International Journal of Remote Sensing*, vol. 18, no. 4, pp. 711–725, 1997.

Curriculum Vitae

Ben Gelman received his Bachelor of Science in Computer Science from George Mason University in 2015, where he worked as a research assistant for several years. He performed his graduate studies while doing machine learning research at Two Six Labs. He plans to continue his research in deep learning, hoping to both advance the field and apply it to many realistic problems.