NSGA-VIT: AN EVOLUTIONARY APPROACH TO VISION TRANSFORMER ARCHITECTURE DESIGN

by

ANDREW BECKER

(Under the Direction of Fred Maier)

ABSTRACT

In recent years, the success of Transformers has been demonstrated in computer vision (CV) tasks, with the Vision Transformer (ViT), which competes with CNN networks on image classification tasks when using pre-trained models. Many of these deep learning models are designed by experts, which takes knowledge, time, and labor costs. Neural Architecture Search (NAS), seeks to automate the process of designing a neural network architecture. In this paper, I propose NSGA-ViT, a multi-objective evolutionary NAS for designing Transformer-based networks for computer vision tasks. NSGA-ViT utilizes a multi-objective genetic algorithm (NSGA-II) to design a ViT network with two objectives: maximizing performance, and minimizing network size. NSGA-ViT searches a search space of self-attention and convolution operations to discover a transformer architecture which outperforms ViT on CIFAR-10 and while containing half the parameters.

INDEX WORDS: Neural Architecture Search, Evolutionary Computing, Computer Vision, Transformers, Deep Learning

NSGA-VIT: AN EVOLUTIONARY APPROACH TO VISION TRANSFORMER ARCHITECTURE DESIGN

By

ANDREW BECKER

B.S., The University of Georgia, 2022

B.A., The University of Georgia, 2022

A Thesis Submitted to the Graduate Faculty of The University of Georgia In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2023

NSGA-VIT: AN EVOLUTIONARY APPROACH TO VISION TRANSFORMER ARCHITECTURE DESIGN

by

ANDREW BECKER

Major Professor:	Fred Maier
Committee:	Khaled Rasheed Shannon Quinn

ACKNOWLEDGMENTS

I would like to thank my committee for seeing me through the process of completing this work. I would also like to thank my family and friends for sticking by my side and showing me support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION AND SUMMARY

1.1 Introduction

In recent years, the Transformer Architecture has become the de-facto solution for Natural Language Processing (NLP) tasks, outperforming both traditional machine learning methods and deep learning methods such as Convolution Neural Networks (CNN) and Recurrent Neural Networks (RNN)[1]. As a result, large language models built from transformers, such as Chat-GPT, built from the Generative Pre-trained Transformer (GPT-3) [2], have accrued widespread popularity for a myriad of generative text applications. Recently, the success of Transformers has been demonstrated in computer vision (CV) tasks, with the development of the Vision Transformer (ViT), which competes with state-of-the-art (SOTA) CNN networks on image classification tasks when using transfer learning to fine-tune pre-trained models [3].

Additionally, many of these state-of-the-art deep learning models are designed and built by teams of machine learning experts, which takes extensive expert knowledge, time, and labor costs. In recent years, however, research into Automated Machine Learning (AutoML) seeks to streamline the process of designing deep-learning algorithms by using machine learning to make the design decisions. Further, Neural Architecture Search (NAS), pioneered by Zoph et. al [4], seeks to automate the process of designing a neural network architecture by using a search strategy to design the architecture. NAS-built CNN architectures have been shown to compete with hand-designed architectures in a range of computer vision tasks [5, 6, 7]. In addition to optimizing the performance of networks, work in hardware-aware NAS has been performed, optimizing architectures for the trade-off between performance and computational complexity [8, 9], allowing

NAS-built architectures to be implemented in a range of compute environments depending on the resources available.

NAS is beginning to be applied to transformer architectures [10, 11], with a number of studies optimizing various hyper-parameters of the network, such as depth, number of attention heads, embedding size, etc. Additionally, some Transformer NAS frameworks utilize a hybrid search space of convolutions and multi-head attention layers in order to combine the benefits of transformers with CNNs for a myriad of NLP and vision tasks[11, 12].

In this paper, I propose a multi-objective evolutionary framework for designing Transformer-based networks for computer vision tasks, deemed NSGA-ViT. NSGA-ViT utilizes a multi-objective genetic algorithm (NSGA-II) to evolve a population of architectures according to two primary objectives: maximizing performance, and minimizing network size. NSGA-ViT searches a search space of self-attention and convolution operations to produce a hybrid convolution-attention transformer architecture which outperforms ViT on small datasets while maintaining a similar number of parameters.

The rest of this paper is organized as follows. Chapter 1 discusses motivations, contributions, and a summary of experiments, Chapter 2 provides a review of related literature, Chapter 3 outlines experimental procedures, Chapter 4 presents experimental results and discussion for NSGA-ViT, and Chapter 5 discusses limitations and future work.

1.2 Motivations

While there have been many hand-designed transformer architectures for computer vision [3, 13, 14] Neural Architecture Search for Vision Transformers has yet to be thoroughly explored in the existing literature [13]. Furthermore, Transformer architectures, while scaling up well [15], tend to perform poorly when scaled-down to small data sets due to the high computational cost of the multi-head attention layer. This means training transformers generally requires unreasonably large data sets which may not be publicly available, especially when using a supervised learning paradigm [16]. Additionally, the computational resources for pre-training models are not available to the public, with pre-training ViT costing hundreds of TPU days on JFT-300M and ImageNet-21k [3]. While pre-trained transformer models are available to the public [17, 18], pre-training custom-built architectures is largely unavailable due to computational costs.

The goal of this study is to use a hardware-aware NAS procedure to discover a resource-aware Transformer architecture which attains competitive train-from-scratch accuracy and small and mid-size data sets, and allows for successful training on small and mid-size, publicly available data sets to allow for greater accessibility to transformer usage and research.

1.3 Summary

The NSGA-ViT procedure is a two-step procedure, where in the first step (Search), evolutionary search is run to automate the design of the main computational cell in the ViT architecture. After search is performed, the second step, *Architecture Evaluation* is performed. In Architecture Evaluation, discovered architectures are compared under CIFAR-10 training and evaluation to determine the best solution. [13].

NSGA-ViT is a Neural Architecture Search (NAS) algorithm that utilizes a multi-objective genetic algorithm (NSGA-II) [19] to search a search space of convolution and attention operations to design a Transformer architecture for image classification tasks. The search space for NSGA-Vit is a variant of the Nas-Net search space, defined by Zoph 2018 [5]. In this search space, the search algorithm designs *cells* of operations, which are stacked to form the final architecture, mimicking the way many hand-designed deep learning models are built [5, 20, 21, 22, 1].

1.3.1 Evolutionary Search with Competing Objectives

For multi-objective optimization, I am choosing to optimize two criteria in each network. The first criteria is to maximize validation accuracy on CIFAR-10, and the second is to minimize the number of parameters in the network. During the search, I use the number of floating-point operations (FLOPS) as an estimator for the size of the network, following the same procedure as [8]. Because there are multiple objectives, I utilize a multi-objective genetic algorithm, NSGA-II, with binary tournament selection, two-point crossover, and polynomial mutation. NSGA-II utilizes a special non-dominated sorting algorithm to maintain a diverse population of solutions that are optimized along multiple objectives. Because individuals are ranked and sorted by their performance on each objective, there is no need to weight the objectives against each other, removing bias towards one objective from the evaluation.

In each generation of the search, a population of solutions is generated and evaluated via training and validation on CIFAR-10 [23]. The best performing individuals from the population have an opportunity to engage in *crossover* to share good genes and produce promising offspring. Each generation, only the best performing architectures on both objectives advance to the next generation (selected from the pool of offspring and parents), a framework known as an *elitist* search, since the best architectures remain in the population. As the search progresses, the population is optimized as a whole, and at the end of the search, the best performing individual, representing the *global optimum* is produced. Since NSGA-ViT jointly optimizes solutions under competing objectives, the final population of architectures represents the *pareto-front*, the global optima of trade-offs between the conflicting objectives.

There are two main experimental settings in which I run the NSGA-II algorithm to search for candidate architectures. These two search procedures are (1) a randomly initialized initial population, and (2) a 'warm-start' where the initial population consists of ViT architectures.

1.3.2 Cold-Start (Random Initialization)

In the Cold-Start setting, the first population is seeded using random initialization. The result is a wide diversity of initial architectures, which can be good for forming a diverse Pareto front at the end of the search. Because of the random initialization, this search procedure produces many wide cells which contain many hidden states to be combined at the end of the cell by the final combiner function. To allow for the network to capture the full complexity of the hidden states, the final combiner function is a concatenation of the hidden state tensors. In addition, the input embedding size is multiplied by the number of final hidden states to account for the size of the new concatenated tensor. This parameter is consistent with most CNN NAS methods [7, 4, 8, 6].

1.3.3 Warm-Start

To provide a more focused search around modifying and improving the ViT architecture, I test the effects of ”warm-starting” the search by initializing the first generation of architectures with the encoding for the ViT architecture. With this initialization, the diversity of individuals is reduced greatly compared to the random initialization, however, it leverages expert knowledge by seeding with a well-performing architecture. Specifically, I initialize with ViT because this is the architecture in which I seek to improve upon.

1.3.4 Architecture Evaluation

To evaluate the architectures gathered from NSGA-ViT, the cell with the highest validation accuracy from each search was gathered and trained on CIFAR-10 for 200 epochs, with 512 initial channels, and a network with 6 layers. The architecture with the highest validation accuracy is reported as the final architecture. To compare this architecture against ViT, I report a variety of architecture sizes to compare performance over different numbers of parameters. Following training of CIFAR-10, I scale-up the data-set and train the model on ImageNet-1k and compare the results against transformers in the literature. Finally, I utilize the models trained on ImageNet and fine-tune them on CIFAR-10 to evaluate the ability to utilize transfer learning.

1.3.5 Results

NSG-ViT produces an architecture that outperforms ViT on CIFAR-10, even in its smallest configuration. NSGA-ViT utilizes a combination of convolution, separable convolution, attention, and feed-forward layers to achieve 92.95% accuracy on CIFAR-10 when trained from scratch, an 11.5% improvement . Additionally, NSGA-ViT in its largest configuration is a similar size to the 8-head ViT implementation, containing roughly 3M more parameters than ViT.

NSGA-ViT also outperforms 8-head ViT utilizing the same training parameters on ImageNet-

1k by roughly 7%. However, NSGA-ViT under-performs ViT base from Dosovitskiy et. al [3] by roughly 6% while containing half the parameters.

Table 1.1: **Parameter size and CIFAR-10 Validation Accuracy for NSGA-ViT and ViT.** ViT implementations all use an 8-head attention layer.

Arch	Params (M)	Test Accuracy (CIFAR-10)
ViT 3 @ 256	5.140229	76.58
ViT 6 @ 512	41.008138	81.34
NSGA-ViT 3 @ 256	5.567242	88.37
NSGA-ViT 6 @ 512	44.224522	**92.95**

1.4 Contributions

This paper contributes to and extends the current body of literature in several ways, indicated by the headings below.

1.4.1 Applying NAS to ViT in Novel Way

The first contribution is taking the cutting edge field of Neural Architecture Search and applying it to the new and promising Transformer architecture. While various transformers have been designed using neural architecture search, a hybrid search space of attention layers and convolution layers has yet to be done for computer vision tasks. Overall, work in NAS for transformers is still in its infancy, with less than 4 years of work done in this field as of 2023 [13], leaving a significant space for work such as the present study. Additionally, much of the work in Transformer NAS is focused on optimization of language models, while NSGA-ViT expands this research into computer vision. This project expands transformer architecture search into computer vision in a way that has yet to be published.

Table 1.2: **Comparison of ImageNet Testing Accuracy**

Model	Params (M)	ImageNet Accuracy
ViT-B/16 [3]	86.6	79.8
ViT-8 6 @ 512	43.5	66.7
NSGA-ViT 6 @ 512	45.2	73.5

1.4.2 Multi-objective Optimization of Transformers

Furthermore, nearly all the literature on transformer architecture is concerned only with one objective: improving accuracy/performance. However, many transformer architectures are too large to be stored on a single GPU, let alone limited-compute environments such as smartphones or laptops. I expand this realm of research into the joint optimization of increasing testing performance while minimizing compute costs, and the result is a range of architectures along the size-performance trade-off, which can be used in many applications, including limited resource environments. NSGA-ViT utilizes a multi-objective genetic algorithm to do joint optimization without the need for weighting the importance of certain objectives, resulting in reduced user bias when performing optimization.

1.4.3 Optimizing ViT for small data-sets

Generally, Transformers are pretrained on very large datasets, containing billions of images, and then fine-tuned on smaller data-sets for application in a process called transfer learning. However, fine-tuning models tends to require a high computational costs when the model is pretrained on a large dataset. Thereby, by reducing the size of the pre-training data-sets, the resources required to fine-tune the model can be reduced, creating more efficient transfer learning [24]. However, standard Transformer networks require this large data when pre-training to produce good results. This paper looks into directly optimizing Transformers for smaller data-sets to increase the efficiency of transfer learning and utilize more readily-available data for training.

CHAPTER 2
RELATED WORK

2.1 Transformer Models and Self- Attention

2.1.1 Transformer Model Overview

The Transformer architecture was introduced by Vaswani et. al [1] as a novel solution to sequence-to-sequence problems like machine translation, building language models, and other NLP related problems. The main contribution of the Transformer architecture is its use of the *self-attention* mechanism (or *Scaled Dot-Product Attention*)(Figure 2.1). The self-attention mechanism takes the input data and transforms it via linear projection layers into the *key, value,* and *query* (K, V, Q) and uses them to compare input values against their time-stamp or positional encoding. This mechanism allows the self-attention layer to make representations of global features of the data, while convolution layers, used in CNNs, only create representations of local features. Additionally, self attention mechanism may be applied multiple times in parallel, with the outputs being concatenated together to improve performance. This type of attention layer is called a *multi-head self-attention* layer, and this is the attention layer that the Transformer architecture utilizes.

The equation for the Scaled-Dot-Product attention mechanism can be defined as:

$$Attention(Q, K, V) = Softmax(QK^T/\sqrt{d})V \tag{2.1}$$

Where Q, K, and V are the query, key, and value, T is the time-stamp (position) and *d* is the dimension of the input. Additionally, masking may be applied, hiding values from the attention layer, which sometimes improves performance [1].

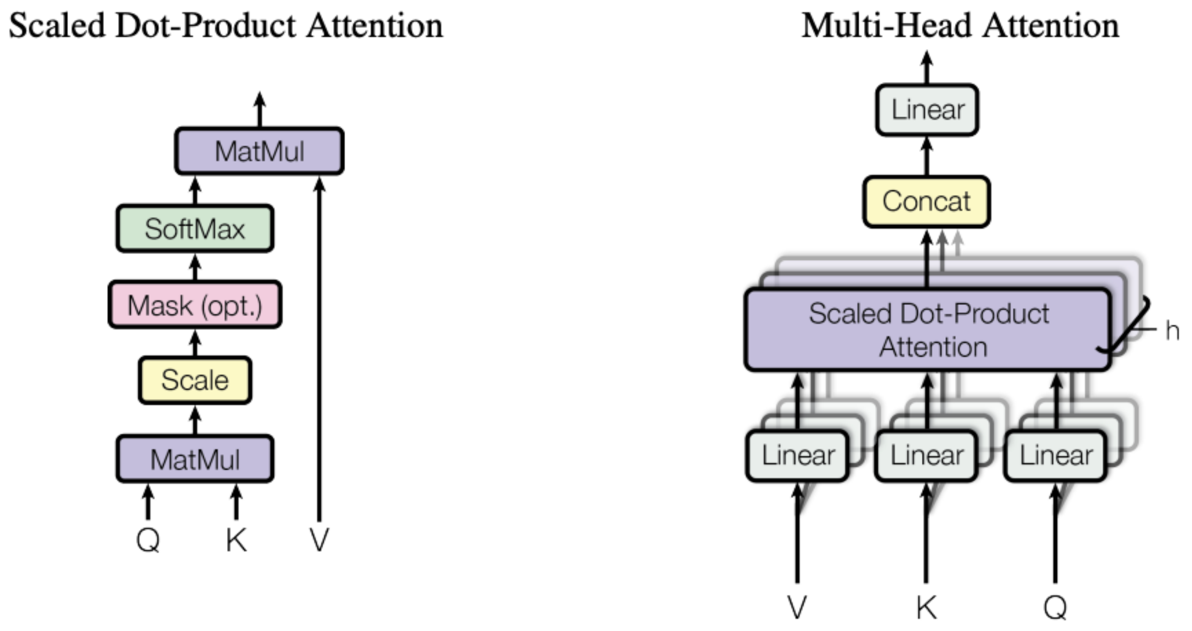


Figure 2.1: **Illustration of Attention Mechanisms from Vaswani et. al** [1] LEFT: Scaled Dot-Product Attention. RIGHT: Multi-head attention

The Transformer is made of two blocks, the *encoder* layer, and the *decoder* layer (Figure 2.2). Each block of the network is formed by an Attention layer followed by a fully connected feed forward network (multi-layer perceptron [MLP]) with residual connections between the operations of the block. This way, the MLP layer can see both the output of the attention layer and the unaltered input of the attention layer. For supervised training, inputs are fed into the encoder, and then the outputs of the encoder are fed into the decoder along with the outputs (labels) of the training data. The encoder stores the representations of the inputs, and the decoder converts the output probabilities into the output sequences. Inside the encoder and decoder, the *transformer block* is stacked multiple times to form the architecture. For the base Transformer, the encoder is made of 6 transformer blocks, and the decoder is made of 8 stacked transformer blocks Figure 2.2.

2.1.2 Vision Transformer

In 2021, Dosovitskiy et. al [3] developed a transformer architecture for computer vision tasks, called the Vision Transformer (ViT). In contrast to CNN models, ViT does not use convolutions

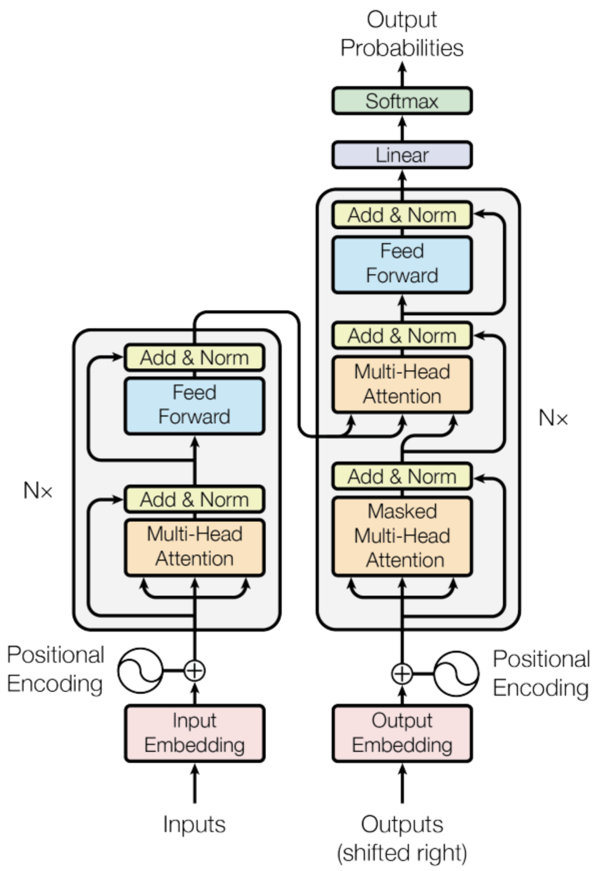


Figure 2.2: **Transformer architecture from Vaswani 2017** [1]. Inputs are fed into the encoder layer, which uses multi-head self-attention followed by a feed forward layer. Labels (outputs) are fed into the decoder layer, as well as the output from the encoder. Finally, the decoder output is fed into a linear layer with a softmax classifier to produce the output probabilities.

to extract local image features, and therefore reduces the inductive bias in the architecture as compared to CNN models. To reduce inductive bias, ViT takes advantage of self-attention and the transformer encoder to create representations for global image features, even at low levels in the architecture. Since the self-attention layer of the Transformer only takes in 1-dimensional data, but images are 2-dimensional, ViT adds preprocessing layers to make the image data compatible with the self-attention layer (Figure 2.3). First, the images are turned into non-overlapping ”patches” of image data by running though a NxN convolution stem with a stride of N (patchify stem), where N is the patch size. Then, the patches are flattened to produce a 1-dimensional representation of the image, and a position embedding, and learnable class token (common for NLP tasks) are added to the patch to closely replicate the structure of language data. After the preprocessing stem, the data is fed into a Transformer Encoder, with *layer normalization* occurring before attention layer and MLP layer. Finally, the output of the transformer encoder is fed into an MLP head classifier layer to make the class predictions.

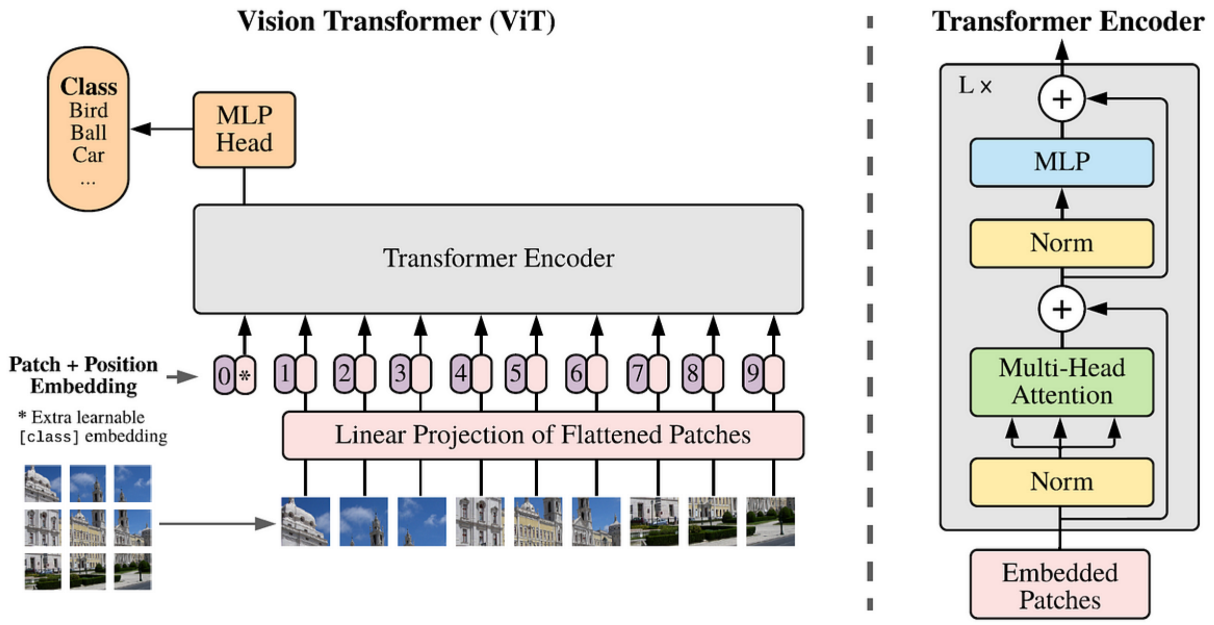


Figure 2.3: **Vision transformer architecture from Dosovitskiy et. al** [3]. Images are split into patches by an initial convolution layer before being flattened and adding position and class embedding. The resulting data is fed into a stack of transformer encoders, followed by an MLP classifier layer.

One drawback of ViT and Transformer architectures is the large data requirement for pretraining the network. It is recorded that ViT under-performs CNN architectures when trained and evaluated on ImageNet, which contains over 1 million training images, with 1000 unique class-labels [3]. However, ViT begins to compete with and outperform CNN architectures after pre-training on large datasets, ImageNet-21k [25] with 21,000 unique class labels, and the proprietary JFT-300M data-set [26], indicating a reliance on big-data for training transformer models for downstream tasks. Work is being done to develop more efficient transformers which can acheive high accuracy on ImageNet-1k and CIFAR-10 by training from scratch. Cao et. al [27] evaluates different vision transformer models under various train-from-scratch environments on small datasets (2040 images), adopting a method of pre-training on small data to create less data hungry fine-tuning.

Yuan et. al [28] adapts the patchify stem to create overlapping tokens in the images, achieving 83.3% accuracy when trained from scratch on ImageNet, a significant improvement to the base ViT [3]. However, work on creating less data-hungry ViT models remains in its infancy, allowing a niche for the present work to contribute.

2.2 Genetic Algorithm

2.2.1 GA overview

The genetic algorithm is a form of population-based optimization algorithms that improves a population of individual solutions simultaneously. The inspiration for this algorithm comes from Darwinian survival of the fittest, where only the best suited to survive organisms survive to reproduce. In a genetic algorithm, individuals are complete solutions to the given optimization problem. Each individual is evaluated, and assigned a ”fitness” score based on the individual’s performance on the evaluation. Once all the individuals are given a fitness score, there begins a selection process to determine which solutions will pass their ”genes” onto the next generation. Parents may be selected in a variety of selection schemes, such as *”roulette wheel,”* where individuals are chosen based on the proportion of the total population fitness that they carry, or *”tournament,”* where solutions are picked out for small tournaments with the highest ranking individuals advancing to parenthood. After parents are selected, they produce the next generation, or *offspring* through the process of *recombination* or *crossover*. Through crossover, genes are recombined between parents so that the portions of the fitness that produce competitive fitness scores are propagated throughout the population. After recombination, *mutation* is performed on the individuals to introduce new genes into the population. Mutation usually occurs at a very small rate, with a gene from a selected individuals switched out for another gene in the search space. In the genetic algorithm, mutation allows for *exploration* of the search space by introducing new genes into the population, and crossover allows for *exploitation* of good genes to improve the fitness of the population. At the end of a set number of generations or fitness evaluations, the best performing individual from the search is returned as the final solution. The psuedocode of the general-purpose genetic algorithm is presented in Algorithm 1.

Algorithm 1 Genetic Alogorthm. After initializing a fixed number of generations and an initial population, each individual is evaluated to determine its fitness score. Then, each generation, the individuals from the population with high fitness scores are selected via *selection* operators to be added to the parent pool. Then, parents from the parent pool are chosen for *crossover*, sharing genes to create *offspring*. The offspring undergo *mutation*, and are then evaluated for their fitness scores. When the number of offspring matches the population size, the previous population is removed, and the offspring population takes their place (generational GA). This process occurs for the number of predefined generations. The highest scoring solution in the search is reported.

```
n_gens                                                          ▷ Number of generations
population
while $|population| < P$ do                                     ▷ Initialize and evaluate population
    initialize individual
    evaluate individual
end while
while $generation < n\_gens$ do                                 ▷ Perform evolutionary operations
    while $|offspring| < |population|$ do                       ▷ Generational algorithm
        $sample \leftarrow \emptyset$                           ▷ parent candidates
        while $|samples| < S$ do
            $candidate \leftarrow$ random element from population
            add candidate to sample
        end while
        $parent1, parent2 \leftarrow$ highest scoring individuals in sample
        offspring $\leftarrow=$ Crossover(parent1, parent2)
        offspring $\leftarrow=$ Mutate(offspring)
        offspring.fitness $=$ Evaluate(offspring)
        add offspring to population
    end while
    remove dead from population                                 ▷ Lowest scoring
end while
return highest-scoring in search
```

2.2.2 Non-Dominated Sorting Genetic Algorithm

The Non-Dominated sorting genetic algorithm II (NSGA-II) is a multi-objective genetic algorithm originally proposed by Deb et. al [19] as a solution to optimization problems with multiple conflicting objectives. NSGA-II is an *elitist* algorithm, meaning the best solutions are always kept in the population allowing for each generation to represent an updated *pareto frontier*, representing the best trade-offs between the objectives. In addition to elitism, NSGA-II utilizes a fast non-dominated sorting approach, which has a time complexity of $O(MN^2)$, compared to NSGA-I, which utilizes a sorting algorithm with a time complexity of $O(MN^3)$ [19], where M is the number of objectives and N is the population size. This non-dominated sorting algorithm seeks to retain only each solution that is not dominated by another solution. In order for a solution to be dominated, it must be defeated on one or more objectives, while defeating the other individual on no objectives. This way, each solution that is retained represents the best trade-off between the objectives compared to its neighbors.

In order to determine the best individuals in each population, each solution p is assigned two values, a domination count, n_p, which is the number of solutions that dominate p, and $S(p)$, the set of solutions which p dominates. All the solutions on the domination front have the domination count, $n_p = 0$. Then, for each solution on the first domination count, each member of $S(p)$ is visited and its domination count is reduced by 1. If any of these members have a domination count, $n_p = 0$, it is placed in a second list, Q, which is the second non-dominated front. Then, the procedure is done again on list Q until the third non-dominated front is found. In this procedure, each solution is visited a maximum of $N - 1$ times before its domination count becomes 0, and it is never visited again. This results in a time complexity of $O(MN^2)$. This fast non-dominated sorting approach is an essential piece of NSGA-II, allowing fast evaluations of each non-dominated front in the population.

Since NSGA-II is a multi-objective elitist algorithm, in theory, the final population is the best estimation of the pareto front. However, to produce a useful set of solutions, it is ideal that algorithms are adequately spaced along the pareto front. To accomplish this, NSGA-II utilizes a unique *crowding* function, which preserves the diversity of solutions along the non-dominated front. The crowding function estimates the average density of solutions around each solution in the non-dominated front to determine the solutions to keep in the population. Each solution has two metrics used for determining which individuals to keep: 1) the domination rank (which domination front does the solution belong to), and 2) the crowding distance. This way, for two solutions, the one in the better ranking domination front is kept, and if they are in the same domination front, the one that is less crowded (largest crowding distance) is kept. This results in an even dispersal of solutions across the pareto front, without the need for user tuning of any parameters which introduce bias.

NSGA-II has become the most widely used multi-objective GA due to its fast time complexity and efficient domination calculations. Because of these features, NSGA-II is a promising algorithm for multi-objective optimization, such as the work done in this paper.

2.3 Neural Architecture Search

2.3.1 NAS overview

Neural Architecture Search (NAS) is an automated optimization procedure to design neural network architectures. Traditionally, the best performing neural network architectures are designed by hand, however, this process requires expert knowledge, often a team of machine learning experts, along with time, labor, and associated labor costs. Furthermore, a neural network is considered a *black-box* or *unexplainable* form of AI, meaning the representations inside the architecture are not interpret-able by human practitioners, leaving design decisions up to a guess-and-check framework. Neural Architecture Search, developed by (Zoph 2016)[4] seeks to eliminate much of expert-knowledge and manual labor required for designing high-performing neural architectures. The process of NAS involves using a machine-learning based search procedure on a selected search space which defines network operations and connectivity. NAS can be done on two levels of the architecture, allowing for design of the overall connectivity of the whole architecture (macro search space), or searching for small blocks of operations, called *cells*, which are stacked to form the final architecture (micro or cell-based search space), mimicking the way many hand-built neural architectures are designed [29]. Furthermore, a variety of machine-learning algorithms may be used to sample and evaluate architectures within the search space.

Neural Architecture search can be considered a *bi-level* optimization problem, where two layers of the problem are optimized simultaneously. In the outer level of optimization, the search procedure is used to optimize the individual architecture. During the evaluation phase, the inner

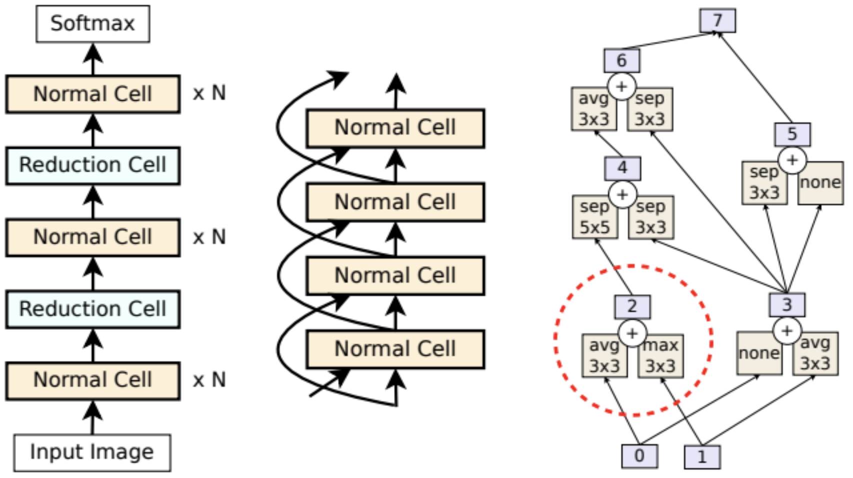


Figure 2.4: **Illustration of Macro-level architecture from Zoph and Real [5, 6]**. **LEFT**: the overall structure of the architecture trained on CIFAR-10. Normal cells are alternated with reduction cells (Not used in present study) at two depths in the network. Cells are stacked linearly. **MIDDLE**: the structure of the normal blocks in the model. Normal cells contain residual connections, allowing each cell to receive input from two cells before it. **RIGHT**: illustration of a possible cell from NAS-Net search space. The red circle highlights one operational block, made up of two operations from hidden states 0 and 1, respectively, combined by an addition operator. This cell contains 5 blocks with a final combiner.

level of optimization is evoked, where the network is trained and network weights are optimized to provide a fitness evaluation of each individual. The inner level of optimization is done by some variation of Stochastic Gradient Descent (SGD), such as Adam or traditional SGD depending on the types of architectures being evaluated. While this accrues large computation costs, it is the simplest and most straightforward way to evaluate the success of an individual solution.

One form is Reinforcement Learning [4, 5], which uses a Recurrent Neural Network (RNN) as the controller to sample architectures from the search space, and the controller is rewarded by feedback from the *environment*, which is the training-validation environment. Through rewards for good architectures, over many iterations, the controller improves the architectures being sampled, until termination criteria are met. This search uses a hand-designed *cell-based* search space deemed NAS-Net search space, in which operations are applied in pairs and combined to form hidden states inside the network. Each cell is made of a fixed number of these operation pairs, called *blocks*. For each block, the controller samples two *hidden states* for input, an *operation* to

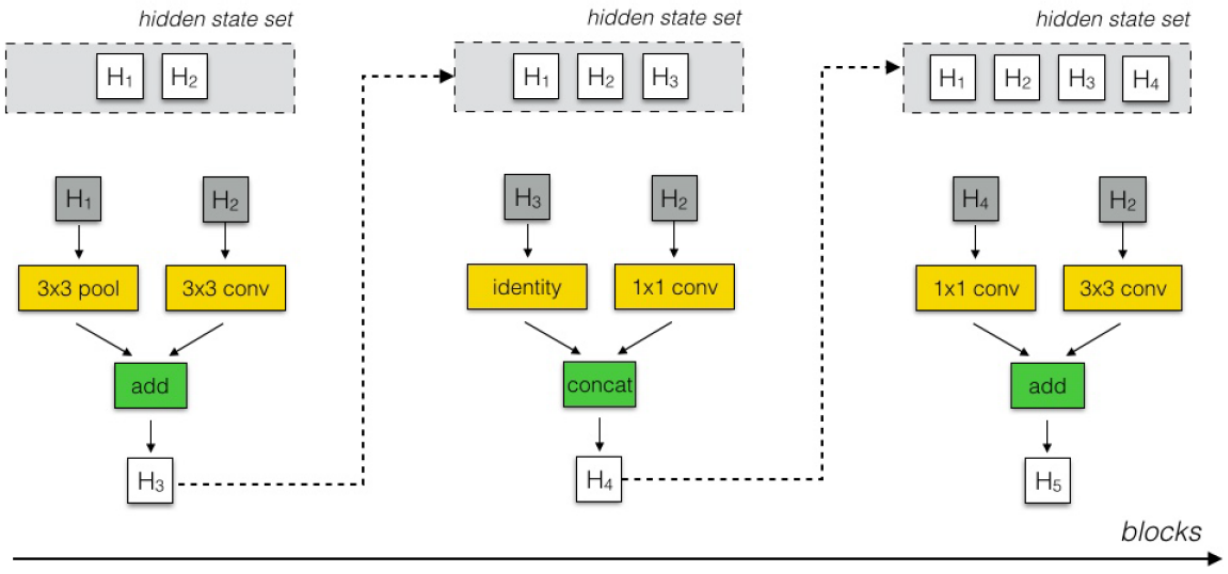


Figure 2.5: **NAS-Net Search Space from Zoph et. al** [5]. Cells are formed via a series of operational blocks. Blocks are formed by selecting two input hidden states, an operation to be performed on each state, and a combination operation to produce the new hidden state.

be performed on each hidden state, and a *combiner function* to combine the states to produce a new hidden state. Any blocks with no directed output are concatenated together to form the output of the cell. During hidden state selection, the controller may sample any hidden state before it, including the output of the previous cell ($h[i]$) and the output of the cell prior to the previous cell ($h[i-1]$).

Real et. al [6] demonstrated that evolution provides a more efficient search procedure for NAS, saving GPU costs, while producing an architecture competitive with hand-designed architectures, and architectures found through reinforcement learning. Furthermore, Real et. al [6] demonstrates that evolution is effective at searching the NAS-Net search space, as they utilize the same search space as Zoph et. al, 2018 [5]. In this framework, outer-level optimization is done using a genetic algorithm (see **Genetic Algorithm**) where each individual is a cell encoding, and networks are trained one-by-one to produce a fitness score. Lu et. al [8] takes this a step further by using a multi-objective genetic algorithm (NSGA-II) to optimize competing objectives, maximizing network performance while minimizing network size to produce a population of architectures that can be used across different compute sizes.

The implementation of the architecture search for NSGA-ViT is heavily inspired by Lu et. al [9], who report their discovered architecture (NSGA-Net) in a macro-level architecture search, in which they use NSGA-II to define the connectivity of an entire network. However, in their implementation, they include reports of their methodology used on the NAS-Net search space [5]. This paper utilizes their implementation of NSGA-Net on the NAS-Net search space as the basis for the methodology, i.e. utilizing NSGA-II on the NAS-Net search space, utilizing crossover and polynomial mutation, and following the same architecture encoding framework.

2.3.2 NAS for Transformers

Although all the previous examples are searching for CNN architectures, So 2019 [11] utilizes the NAS-Net search space to search for transformer cells that include both multi-head attention layers and *Nx1* convolution operations to extract local features from 1-dimensional linguistic data. The researchers use a single-objective genetic algorithm to sample the cell-based search space for both the encoder layer and the decoder layer. Unlike [6], the number of stacked cells in the final architecture is included in the search space, optimizing the size of the network for the validation phase. The evaluation stage of the search is performed using WMT'14 data set. The resulting architecture, known as the Evolved Transformer, uses a combination of 1-dimensional convolutions and multi-head attention layers to acheive SOTA results on a selection of language tasks. The success of this method is shown in large language models, as the evolved transformer was used as the backbone for Gooogle's Meena chatbot [30].

The search space and implementation of the Transformer encoder for this paper is inspired heavily from So et. al, who utilize the NAS-Net search space to design cells for the transformer encoder. Like So et. al, we utilize 4-head, 8-head, and 16-head MHSA layers, as well as a subset of convolution operations and nonlinearities from the search space. See Appendix A for a list of all the possible operations in our search space. Additionally, like So et. al, we explore both a ”cold-start” initialization and a ”warm-start” initialization for the evolutionary search. See Chapter 3 for more details about the search and the different initialization methods.

2.4 Data-Sets

In many NAS frameworks for computer vision, architecture training is done on common benchmark datasets for computer vision, such as CIFAR-10, CIFAR-100 [23], and ImageNet [25] [4, 5, 6, 7, 8, 31, 13].

2.4.1 CIFAR-10

CIFAR-10 is a small image dataset that consists of 60,000 $32x32$ color images, spread evenly across 10 mutually exclusive classes [23]. The training set consists of 50,000 images, and the test set consists of 10,000 images, all evenly spread across the classes. The dataset was developed as a benchmark for low-resolution computer vision tasks. An expanded version, CIFAR-100, contains 60,000 images across 100 unique class labels.

2.4.2 ImageNet

ImageNet was created by Deng et. al as a large-scale image data-set based upon the WordNet ontology [32]. ImageNet seeks to populate a majority of the 80,000 wordnet synsets with 50-100 high resolution images. The result is a hierarchically structured data-set with 3.2 million cleaned and labeled images spread over 5247 categories. A commonly used subset of ImageNet, ImageNet-1k, contains 1,281,167 training images, 50,000 validation images, and 100,000 test images, spread across 1,000 classes. This subset was introduced for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [33] in 2012, a widely popular and influential computer vision competition, resulting in the introduction of a number of state-of-the-art computer vision models, such as AlexNet [20], GoogLeNet [21], Resnet [22], and SENet [34].

Much of the work in NAS for computer utilizes CIFAR-10 as a proxy data-set to predict model performance on ImageNet during the search procedure. This framework speeds up the search process to allow for realistic compute-times. This method was first validated by Zoph et. al [4], and followed in all the previously discusses NAS methods for CNNs.

CHAPTER 3
METHODOLOGY

3.1 Overview of NSGA-VIT procedure

NSGA-ViT is an iterative search algorithm that optimizes a group of potential solutions, called the *population*. Each candidate solution in the population is called an *individual*. In each generation, a subset of the population is chosen to become a *parent* for the next generation. By utilizing *crossover*, the algorithm recombines the parents to form *offspring*, which compete with the parent population for survival into the next generation. Each generation, the population is updated with the best performing individuals from the parents and offspring to create the next population. In this way, the entire population of solutions is optimized at once, providing a range of potential candidates that optimize the trade-off between competing objectives.

The NSGA-ViT procedure is a two-step procedure, where in the first step (Search), evolutionary search is run to automate the design of the main computational cell in the final architecture. After search is performed, the second step, *Architecture Validation* is performed. In Architecture Validation, the size, weights, learning-rate, dropout, and other hyperparameters of the network are optimized by hand or simple grid search to maximize the accuracy of the final architecture[13].

3.2 Data Sets

The structure of the Neural Architecture Search (NAS) can be considered a bi-level optimization problem, in which the outer level of optimization involves using a search procedure to find the optimum neural architecture, and the inner level of optimization is training the weights of each candidate architecture. Because of the bi-level nature of the optimization problem, the inner level

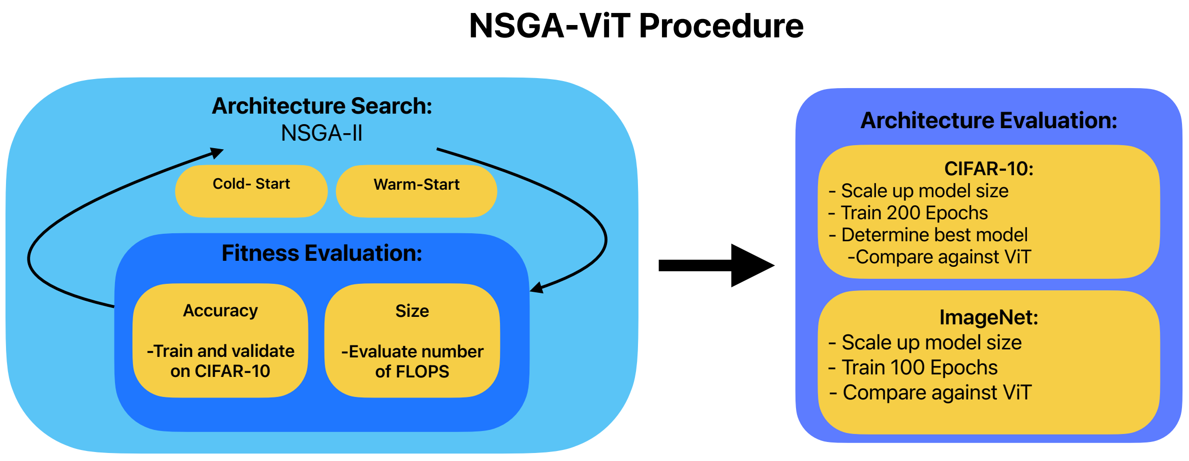


Figure 3.1: **NSGA-ViT Procedure:** NSGA-ViT is a two step procedure, cosisting of architecture search and architecture evaluation. Architecture Search is bi-level optimization problem where architecture configurations are optimized in the outer layer, and connection weights are optimized via training in the inner layer. Architecture evaluation consists of training and evaluating discovered architectures against each other to determine the best solution, and then comparing against Transformers in the literature.

of optimization must occur at every step of the search procedure, leading to long run times. In order to reduce the run times of each search, I have taken steps to reduce the complexity of the inner optimization problem to ensure faster training.

While ViT networks typically require very large amounts data to perform at the state-of-the-art level, small datasets like CIFAR-10 are often used as a proxy data-set for predicting ImageNet performance during architecture search [5, 7, 6]. Additionally, time-to-train is quicker for smaller data-sets, as they contain fewer examples to train the network, and a smaller resolution. Therefore, during the search procedure, all networks are trained on CIFAR-10, and resulting architectures are optimized on CIFAR-10 to produce a train-from-scratch accuracy score. After optimizing the final architecture on CIFAR-10, we move on to evaluation of train-from-scratch accuracy on the larger ImageNet (1k) data set.

3.3 Designing the Search Space

3.3.1 NSGA-ViT Search Space

The search space for NSGA-ViT is a variant of the Nas-Net search space, defined in Zoph 2017 [5] (see section 2.3). In this search space, the architecture is designed by stacking *cells* of operations and hidden states, to form the final architecture structure. The search space is at the *cell - level*, where convolution and attention operations are applied in pairs, and the results are added together. Each pair of operations and their associated add functions is considered a *block*, and the output of each block is a new hidden state. Each block is provided an index in the encoding of the search space to determine the order of operations (see **Encoding**). At the end of the cell, any hidden states with no output are combined via the final combiner function (add or concatenate) to produce the output of the cell.

3.3.2 Encoding Architectures in the Search Space

Genetic algorithms often do not operate directly on the individual solutions *(phenotypes)*. Instead, each individual is represented by a *genotype*, or a simple encoding of the individual that provides instructions for building the phenotype. Encoding the individuals in this way allows for simple operations to be performed in order to change each individual throughout the generations [35]. Individuals in NSGA-ViT are encoded using a list of integers to represent the cell in the NAS-Net search space. Integers are listed in groups of 2, where every group represents a hidden-state operation and index. Every group of 4 integers represents two hidden states which are added together. In this encoding, a cell with two blocks (four hidden states) would be represented as:

$$[[[x_o^1, x_i^1], [x_o^2, x_i^2]], [[x_o^3, x_i^3], [x_o^4, x_i^4]]] \tag{3.1}$$

where every x_o represents an operation, and every x_i represents the index of the block that serves as the input for the operation. In this way, the operation numbers define the operations and the indices define the structure of the cell. The cell is then repeated to form the complete architecture of the network.

3.4 Evolutionary Search Procedure

3.4.1 Genetic Algorithm and Objectives

For the outer-level of optimization, I use a multi-objective genetic algorithm (NSGA-II) to optimize two criteria in each network. The first criteria is to maximize validation accuracy on CIFAR-10, and the second is to minimize the number of parameters in the network.

During the search, I use the number of floating-point operations (FLOPS) as an estimator for the size of the network, following the same procedure as Lu 2019 [8]. Because there are multiple competing objectives, the search produces a final population that estimates the *pareto front*, representing the best trade-offs between each objective.

I utilize a multi-objective genetic algorithm, NSGA-II, with binary tournament selection, two-point crossover, and polynomial mutation. NSGA-II utilizes a special non-dominated sorting algorithm to maintain a diverse population of solutions that are optimized along multiple objectives. Because individuals are ranked and sorted by their performance on each objective, there is no need to weight the objectives against each other, removing bias towards one objective from the evaluation.

Additionally, the chosen evolutionary operations have various benefits for the proposed search procedure. Utilizing crossover, not common in evolutionary NAS methods[6, 11, 36], allows for architectures in the population to share genes that contribute to performance on the fitness evaluation. The non-dominated sorting algorithm implemented by NSGA-II preserves solutions closest to the pareto-front and removes solutions that are defeated on both objectives by another individual. This mechanism allows for a smooth multi-objective search without the need for weighting objectives and introducing bias into the fitness evaluation. Mutation is important because it introduces new genes into the gene-pool, allowing for exploration of the search space and preserving genetic variation in the population. Utilizing tournament selection allows the algorithm to rank architectures on a comparative-basis, eliminating the need for high raw scores on objectives. This is useful when training small architectures, as the raw accuracy of each network may be low due to pruning of the network size.

The evolution phase of the optimization procedure uses a population size of 40, and evolves for 50 generations, resulting in the evaluation of 2000 architectures per search. At the end of each search, a Pareto front of solutions which optimize accuracy at different computational complexities is produced. To evaluate the best architectures against networks from the literature, I take the solution with the highest validation accuracy at the end of the search as the final architecture for architecture validation.

3.4.2 Random Initialization vs Warm-Starting the Search

There are two main experimental settings in which I run NSGA-ViT to search for candidate architectures. These two search procedures are (1) a randomly initialized initial population (cold-start), and (2) a 'warm-start' where the initial population consists of ViT architectures. Evolutionary search can be particularly sensitive to initialization, especially when resource or time constraints, or over-exploitation can limit the portion of the search which is evaluated. With this in mind, I wanted to compare the effect of initializing randomly with a more 'focused' search, which centers around the architecture I wish to improve on. Within each experimental setting, a few preset parameters are changed to keep each search more cohesive with the current body of research, and optimize the performance of the common architectures found in each framework.

3.4.3 Cold-Start (Random Initialization)

In the Cold-Start setting, the first population is seeded using random initialization. The result is a wide diversity of initial architectures, which can be good for forming a diverse Pareto front at the end of the search. Initial experimentation indicates that because of the random initialization, this search procedure produces many wide cells which contain many hidden states to be combined at the end of the cell by the final combiner function. To allow for the network to capture the full complexity of the hidden states, the final combiner function is a concatenation of the hidden state tensors. In addition, the input embedding size is multiplied by the number of final hidden states to account for the size of the new concatenated tensor. This parameter is consistent with CNN NAS frameworks [7, 4, 8, 6]. The solution space in this search tends to be dominated by Multi-Scale Transformers, where the embedding size of the transformer changes due to the width of the cells, often scaling up to 3-4 times the initial embedding size.

3.4.4 Warm-Start

To provide a more focused search around modifying and improve the ViT architecture, I test the effects of ”warm-starting” the search by initializing the first generation of architectures with the encoding for the ViT architecture. With this initialization, the diversity of individuals is reduced greatly compared to the random initialization, however, it leverages expert knowledge by seeding with a well-performing architecture. Specifically, I initialize with ViT because this is the architecture in which I seek to improve upon, and as mentioned previously, I configured the skeleton of the network to include the patchify stem, flattening step, position embedding, and class tokens of the ViT architecture. This technique is inspired by So et. al [11], who record a improvement of model performance by initializing their search with a population of Transformer architectures compared to random initialization. To remain consistent with the search parameters set in [11], any hidden states without output via addition to form the output tensor of the cell. In initial experiments, a large majority of the architectures observed in this setting contain only one final hidden state, therefore, the final combiner function is not used. However, to remain consistent with So et. al [11] and the standard Transformer architectures [1, 3], the final combiner function in this setting is addition.

Additionally, the traditional architecture for ViT contains encoder cells stacked with no skip connects between each cell. Therefore, each cell only receives input from the cell before it. However, the encoding scheme in this study allows a hidden state from two cells before before the current cell, equating to a skip-connect on the macro level. I have included skip-connected hidden state in the search space, and the search procedure may find benefits in including the skip connect as an available hidden state.

3.5 Architecture Evaluation

3.5.1 Comparing Discovered Architectures

Once Candidate architectures have been gathered from the search, the best architectures from each search are compared across the same learning conditions. To determine the best performing architecture, I train each architecture discovered in the search on CIFAR-10, and take the architecture with the best validation accuracy. During architecture evaluation on CIFAR-10, Networks are trained for 200 epochs, using *Adam* optimizer, with a learning rate of 0.0001, no dropout or weight decay, and *Cosine Annealing* as the learning rate scheduler. The best performing architecture is then taken for comparison against ViT networks.

3.5.2 Evaluation on CIFAR-10 and ImageNet

Following Network Evaluation on CIFAR-10, I train NSGA-ViT on ImageNet-1k for 300 epochs, following training parameters reported by Dosovitskiy et. al [3]. I report the testing accuracy on ImageNet compared to other networks in the literature with a similar number of parameters. I utilize a network architecture with 12 layers and an embedding size of 512 for NSGA-ViT training on ImageNet.

CHAPTER 4
RESULTS

4.1 Search Results

During each search iteration, roughly 2000 architectures are evaluated. Since NSGA-ViT is a multi-objective procedure, the final population of architectures contains an estimation of the pareto front, representing the best trade-offs between validation accuracy and the smallest parameter size. After the search, the architecture with the highest validation accuracy is taken as the best architecture to be reported. The search takes 12 GPU days on a single AX5000 GPU.

4.1.1 Cold-Start vs Warm-Start

The Cold-start and warm-start settings are able to achieve comparable results after 50 generations. Figure 4.1 shows a comparison of NSGA-ViT architectures discovered during the search procedures. Architectures labeled *NSGA-ViT-A* were discovered during the cold-start search, and architectures labeled *NSGA-ViT-B* were discovered during the cold-start search. NSGA-ViT was discovered during warm-start. Architectures discovered during the warm-start setting tend to have a larger parameter size than architectures in the warm start setting with the same number of layers and input channels due to the increased output dimension of wide cells.

Figure 4.2 shows the accuracy statistics for each generation in the search in the cold-start and warm-start search settings. The table presents the best error, mean error, median error, and worst error recorded in each generation. As seen on the table, in both the cold-start and warm-start settings, the mean error and the best error consistently improves over the course of the search. In the cold start search the worst error spikes around 10-15 epochs due to the instability of initializing

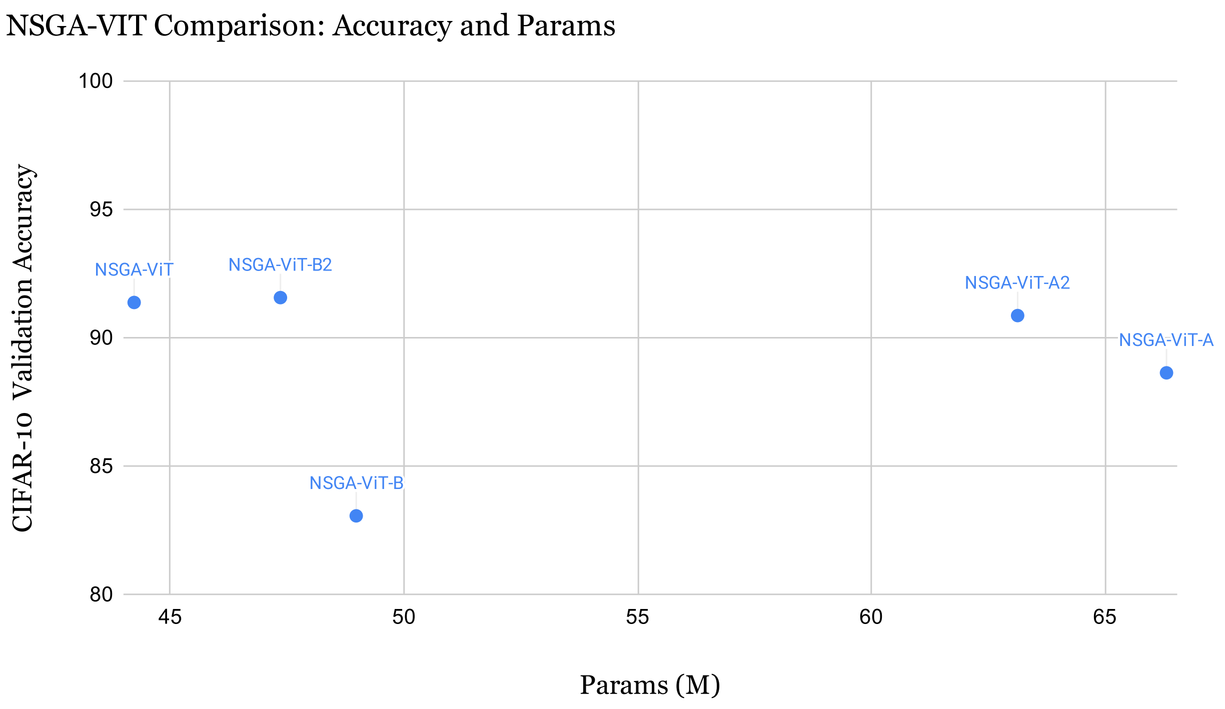
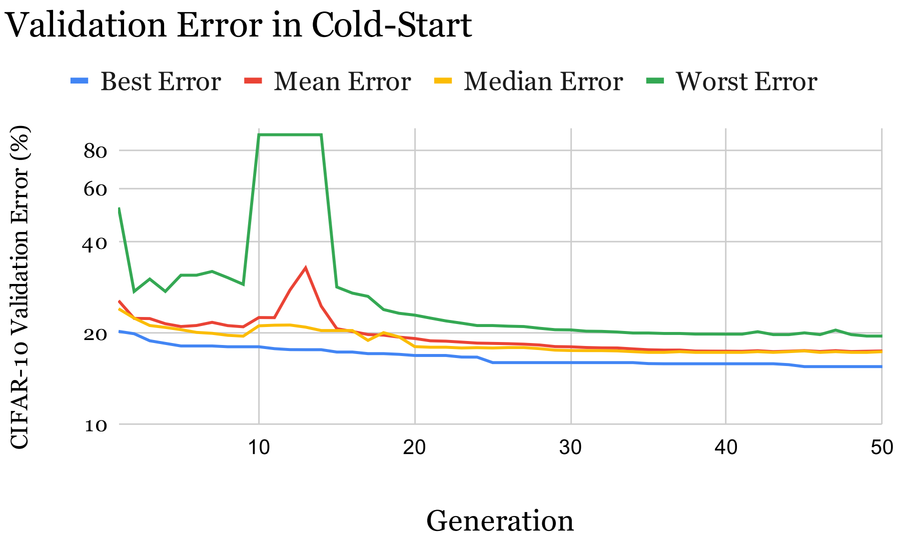


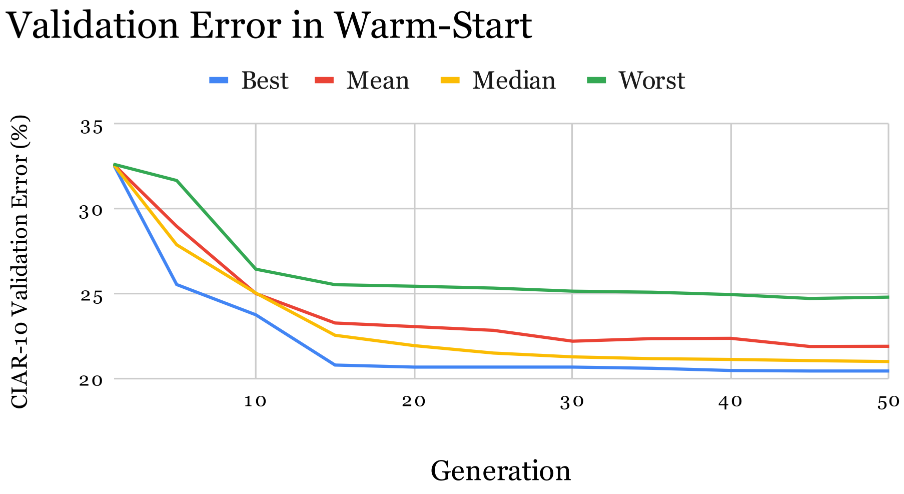
Figure 4.1: **Comparison of CIFAR-10 Accuracy and parameter sizes for NSGA-ViT models.** Architectures labeled *NSGA-ViT-A* were discovered during the cold-start search. Architectures labeled *NSGA-ViT-B* were discovered during the cold-start search. NSGA-ViT was discovered during warm-start. Each architecture was trained for 200 epochs on CIFAR-10 with an architecture of 6 layers and an input dimension of 512.

randomly, but levels out for the remainder of the search.

Figure 4.3 shows the complexity statistics for generation in the search in the cold-start and warm-start search settings. The table presents the best complexity, mean complexity, median complexity, and worst complexity recorded in each generation. All complexities are measured in floating-point operations (FLOPS), which was used as the estimator for parameter size in the search. As seen on the table, in the warm-start setting, the mean complexity and the best complexity consistently improves over the course of the search. In the warm-start the best complexity remains the same over the course of the search, while the worst complexity and mean complexity actually increase slightly during the search, indicating that the initial ViT model is the smallest model discovered during the search.

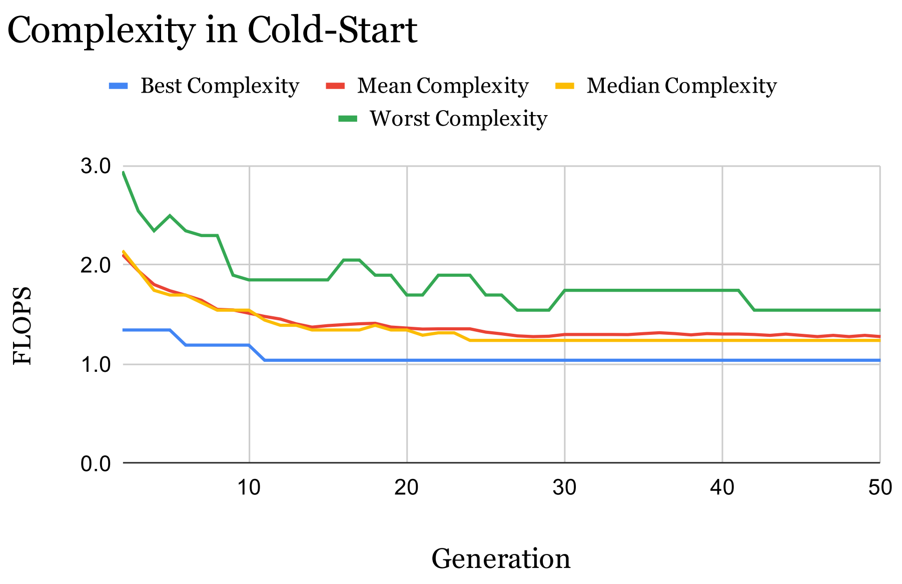


(a) CIFAR-10 validation error for Cold-Start setting.

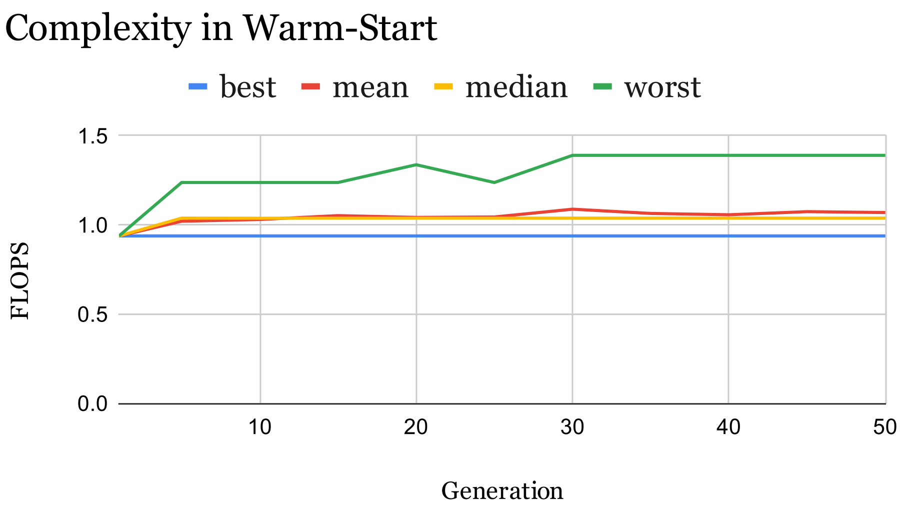


(b) CIFAR-10 validation error for Warm-Start setting.

Figure 4.2: **Graph of CIFAR-10 validation error statistics for each generation in the search.** Statistics reported are best error, mean error, median error, and worst error for each generation in the cold-start and warm-start setting.



(a) Complexity metrics for the cold-start search

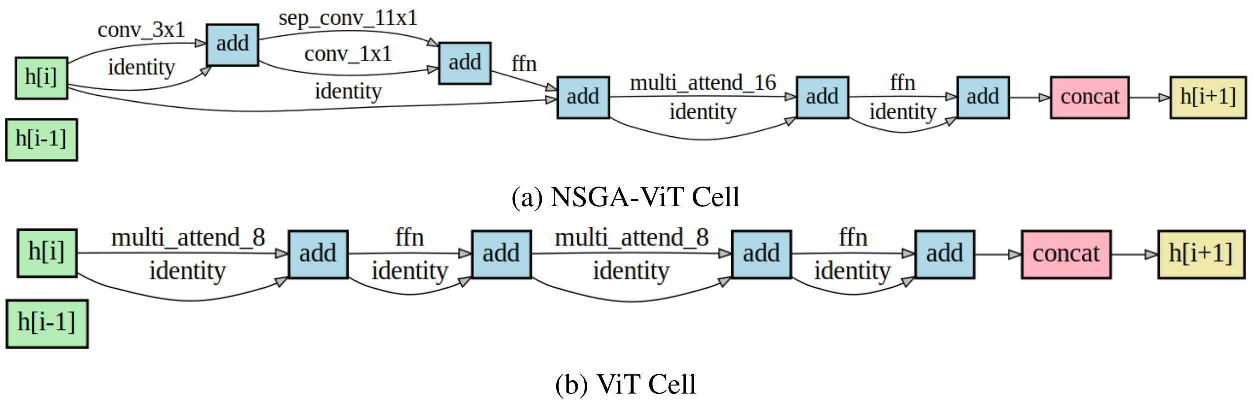


(b) Complexity metrics for the warm-start search

Figure 4.3: **Graph of computational complexity for each generation in each search setting.** Complexity is represented in FLOPS. Architectures in the cold-start setting are initialized with 6 cells, and architectures in the warm-start setting are initialized with 3 cells.

4.2 NSGA-ViT Architecture

After validating results against the other architectures discovered in searches, the most robust architecture is presented here as NSGA-ViT. The NSGA-ViT cell architecture is presented visually in Figure 4.4. The NSGA-ViT architecture retains the multi-head attention and FFN layer at the end of the network from the ViT architecture. However, NSGA-ViT replaces the first half of the ViT cell with a series of convolution operations, using $3x1$, $1x1$, and $11x1$ separable convolutions. The architecture also retains the unbroken chain of identity operations from input to output that is observed in the ViT architecture.



(a) NSGA-ViT Cell

(b) ViT Cell

Figure 4.4: **NSGA-ViT Architecture**. TOP: NSGA-ViT architecture found in the search. BOTTOM: ViT architecture with 8-head MHSA layer. Each cell contains 2 transformer layers, resulting in 2 MHSA and FFN layers per cell.

4.3 CIFAR-10 Results

This section presents the results of NSGA-ViT architecture evaluation on CIFAR-10. CIFAR-10 training is done for 200 epochs using the Adam optimizer with decoupled weight decay unless otherwise specified. All networks use the patchify stem with a patch size of 4 unless otherwise specified. See Appendix C for details on the training parameters.

Table 4.1: **Parameter size and CIFAR-10 Validation Accuracy for NSGA-ViT and ViT.** ViT implementations all use an 8-head attention layer. Additionally, ViT implementations have 2 transformer blocks per cell. Network sizes are reported as (cells @ initial channels). All networks are trained for 200 epochs unless otherwise specified. ViT with with convolution stem replaces patchify stem with 4 3x3 convolutions and one 1x1 convolution per Xaio et. al[14]

Arch	Params	Test Accuracy (CIFAR-10)	f1-score
ViT 3 @ 256	5.140229	76.58	-
ViT 6 @ 512	41.008138	81.34	-
ViT (convolution stem)[14] 6 @ 512	42.797130	82.53	-
NSGA-ViT 3 @ 256	5.567242	88.37	-
NSGA-ViT 6 @ 512	44.224522	91.35	-
NSGA-ViT 6 @ 512 (600 epochs)	44.224522	92.35	-
NSGA-ViT 6 @ 512 + reg	44.224522	**92.95**	0.92

4.4 Comparison Against ViT on CIFAR-10

Table 4.1 shows the train-from-scratch CIFAR-10 test accuracy of NSGA-ViT compared to a traditional ViT model with 8 heads. NSGA-ViT outperforms ViT at both model sizes on CIFAR-10 by roughly 10% The base ViT size (12 layers, 6 cells in my implementation) with 8 heads achieves a mild 81.34% validation accuracy on CIFAR-10 after 200 epochs, and contains 41M parameters, while NSGA-ViT with similar parameter size (44M parameters) performs with 91.35% validation accuracy after 200 epochs. In addition, the NSGA-ViT small network outperforms ViT base by roughly 7% with only 5.6M parameters.

NSGA-ViT achieves 92.95% train-from-scratch validation accuracy on CIFAR-10 when trained for 200 epochs using Adam with a weight decay of 0.3, learning rate of 0.0001 with cosine annealing learning scheduler. Contributing to this performance was a high regularization, including a dropout probability of 0.2, image cutout, and gradient clipping at global norm 1. Interestingly, adding heavy regularization improves performance more than increasing training time from 200 epochs to 600 epochs, as shown in Table 4.1. NSGA-ViT also achieves an f1-score of .92, indicating good precision and recall.

Table 4.2: **Comparison of ImageNet Testing Accuracy**. All Transformer architectures and their train-from-scratch results on ImageNet-1k are reported from the literature, except ours (NSGA-ViT). Transformers are generally labeled **S** for *small* configuration, and **B** for *base* configuration, and labeled with the number of layers, where applicable.

Model	Params (M)	ImageNet Accuracy
CvT-21 [37]	32.0	82.5
T2T-ViTt-19 [28]	39.0	81.4
ResNet101 [22]	44.7	77.4
Swin-S [38]	50.0	83.0
NesT-B [39]	68.0	83.8
DeiT-B/16 [40]	86.6	81.8
ViT-B/16 [3]	86.6	79.8
ViT-8 6 @ 512	43.5	66.7
NSGA-ViT 6 @ 512	45.2	73.5

4.4.1 Confusion Matrix on CIFAR-10

Figure 4.5 Shows the Confusion Matrix for ViT on CIFAR-10 testing. Results are fairly even across classes with the largest confusion between *cats* and *dogs*. The class that was correctly classified the most was *car* with 97% accuracy, and the classes *plane, car, frog, horse, ship* and *truck* have similarly high accuracy.

4.5 ImageNet Results

Table 4.2 shows a comparison of size, FLOPS, and train-from-scratch accuracy on ImageNet-1k between NSGA-ViT and other Transformers from the literature. Although different configurations are available, architectures with similar parameter sizes to NSGA-ViT are chosen, as well as the base configuration of various transformers, which tend to be larger. As demonstrated, NSGA-ViT outperforms the ViT configuration of similar size (our ViT implementation, 8-heads, 6 cells, 512 embedding dimension) utilizing the same training parameters. However, both 8-head ViT and NSGA-ViT under-perform other ViT architectures in the literature in a train-from-scratch setting. Many of these transformers utilize hand-designed alterations to the ViT architecture, i.e. adapting the stem to include shifting windows [38] or recursively aggregating neighboring tokens into one token [28]. Additionally most of these methods utilized more advanced training techniques such as

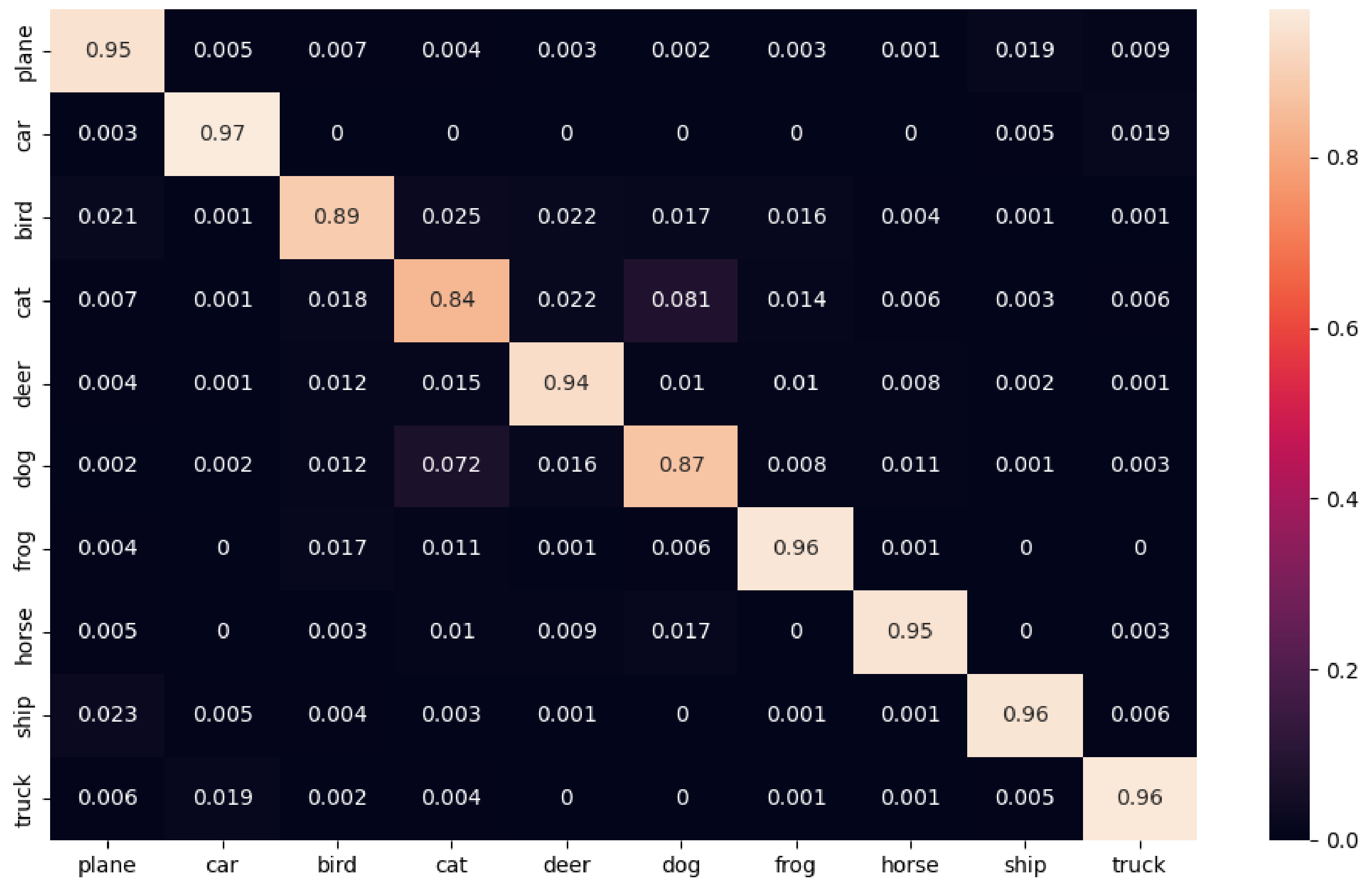


Figure 4.5: **Confusion Matrix for NSGA-ViT on CIFAR-10**. NSGA-Vit outperforms ViT on CIFAR-10 when trained from scratch, achieving relatively balanced performance across classes. The largest confusion between classes can be seen between cats and dogs.

utilizing a CNN teacher for learning token distillation [40] or utilizing advanced regularization and data-processing techniques, such as Mixup, augmented regularization, or altering the way postition and class tokens are generated in the network [13, 37, 28, 38, 39, 40]

4.6 Discussion

This section contains a discussion the results gathered from NSGA-ViT.

4.6.1 Evaluating Search Results

Figure 4.1 Shows a comparison of size and performance for all the architectures found in the search. As Shown, each search produced models that perform similarly on CIFAR-10. One notable difference is a performance of roughly 83% accuracy for NSGA-ViT-B, however, this search terminated early after roughly 600 fitness evaluations. Another notable feature is that searches in the "warm-start" setting produced final models with roughly 50% more parameters than architectures in the warm start. As seen in Appendix A, each search produced a substantially different architecture, indicating that a variety of architectures perform similarly in the evaluation stage. This hints at the potential for a wider variety of successful transformer-based models, either NAS discovered or handmade, in the future.

Cold Start vs. Warm-Start: Additionally, Figure 4.3 and Figure 4.2 show the graphs of accuracy and complexity metrics across generations in the search. In the cold start search, we can see that the best, worst, mean, and median metrics for both accuracy and size improve throughout the duration of the search, with an outlier architecture occurring between 10 and 15 epochs. However, when we examine the performance of the best architecture in each generation, we can see only a small improvement of 2-4% from beginning to end in the search. Although the best accuracy and size see only a small improvement throughout the search, The architectures with the best accuracy generally are the largest architectures, so a significant improvement from 3.0 MFLOPS to 1.5 MFLOPS from beginning to end is substantial if we consider this correlation with high accuracy. Interestingly, due to the high variety of architecture configurations, *crossover* operates less as an exploitative operator and more as a large ”jump” mutation. This is because the exact good genes cannot be determined without predictive analytics or ablation studies, and deep learning models tend to be unstable, meaning a small change to the architecture can result in a large difference in performance. Because NSGA-ViT is elitist, and as such tends to be highly exploitative, risking becoming trapped in local optima. However, because crossover acts as large jumps in the configuration of an architecture, the algorithm may be able to effectively escape a local optimum.

In the warm-start search, the initial population was made up of ViT architectures. This means that mutation plays a large role in producing new architectures early in the search. We can see from the graph of validation error (Figure 4.2), that quick improvements are made early in the search, indicating the effectiveness of the selection operator in picking effective solutions to advance generations. By halfway through the search the best and worst accuracies level out, remaining relatively constant through the end of the search. During this time however, the rest of the population continues to improve, indicated by the improvement in mean and median Since models produced in the warm-start are less genetically diverse than in the cold-start setting, crossover works in an exploitative manor, especially early on, recombining many identical elements while introducing small variations to the parent architectures. This feature may contribute to the significant improvements seen early on in the search. Examining the graph of complexity in the warm-start search, we can see that the complexity of architectures in each generation actually increases slightly. This may be because the ViT cell that initializes the search is a 5-block cell, containing an extra skip connection compared to the standard 4-block ViT to keep the number of blocks consistent across the searches. This extra block of skip-connections allows for the architecture to be complexified by replacing the connections with operations, a design decision I feel is validated, as including an extra skip connect allows for a more sophisticated layer of operations to potentially replace the complex MHSA layer. Even so, the worst size in each generation only increases by about 0.5 MFLOPS from beginning to end of the search, indicating that NSGA-ViT successfully evaluated architecture along both objectives.

4.6.2 Optimizing for Small Data-Sets

The NSGA-ViT process successfully optimizes a ViT network tailored for small datasets. By utilizing CIFAR-10 performance as an evaluation metric for the search, models that perform well with small data outperform models that may have performed better on larger datasets. While this leaves the ability to scale the model up to question and experimentation, models are produced with a low-risk for over-fitting small datasets, a significant gap in the literature to be filled [24, 27, 13]. Specifically, the models perform well on the dataset used for evaluation during the search, outperforming ViT by nearly 10% in our largest configuration. This indicates the potential for various future experiments searching on datasets with higher resolution images, like subsets of ImageNet to gather models that perform well with higher resolution images in small samples.

Looking at the Confusion Matrix for ViT on CIFAR-10 (Figure 4.5), we can see that NSGA-ViT has fairly even class distribution across classes, indicating a well balanced model. However, there is a relatively large confusion between two classes, *dog*, and *cat*, animals with similar features. This indicates that NSGA-ViT may not be representing distinguishing features thoroughly enough and relying more on global feature representation to make decisions (i.e. small size, 4 legs, etc.). Regardless, The significant improvement over ViT, which contains no local feature representation, indicates that some features are being represented in NSGA-ViT which are not featured in ViT.

4.6.3 ImageNet Discussion

Though training for NSGA-ViT follows the training procedure outlined by Dosovitskiy et al. [3], utilizing a learning rate of 0.0001 with the Adam optimizer and cosine annealing scheduler, as well as a warmup period of roughly 10 epochs, NSGA-ViT vastly underperforms other transformers on ImageNet training. Additionally, NSGA-ViT underfits the dataset, indicating that adding regularization (as done for CIFAR-10) is not a promising solution for improving ImageNet performance.

However, examining the configuration of NSGA-ViT may reveal essential elements that hinder succsess on ImageNet. The first is that the search procedure was done on CIFAR-10, training networks on a small data-set of low-resolution images. The result is an architecture optimized for low-resolution images, not guaranteeing success on higher-resolution images.

The second feature of NSGA-ViT that may be hindering its success on ImageNet is related to the size of the MHSA layer and input dimension. NSGA-ViT was configured for $32x32$ images on ImageNet. Thus, with an MHSA with 16 heads and an input dimension of 512, each head in the attention layer, resulting in 32 pixels (i.e. one whole image) being fed to each head in the attention layer. However, the base ViT configuration contains 12 heads in the MHSA layer and an input dimension of 768, resulting in 64 pixels being sent to each head of the attention layer. ImageNet images are processed with a resolution of $224x224$ and ViT uses a patch size of 16, resulting in 4 patches being sent to each attention head. However, in NSGA-ViT, only 2 patches are sent to each attention layer, which may not be enough to determine global features from the images. This can be solved by increasing the input dimension of NSGA-ViT to 1024, resulting in the same number of pixels being sent to each attention head, but this will increase the parameter size of the model. Unfortunately, utilizing one GPU for training on ImageNet limits my ability to train under this parameter setting, and further experimentation may be reserved for future work. See chapter 5 for further discussion of future work on ImageNet training.

CHAPTER 5
CONCLUSION

This paper proposes NSGA-ViT, an evolutionary NAS algorithm for optimizing Vision Transformer Architectures. NSGA-ViT utilizes a multi-objective genetic algorithm to optimize Transformer architectures for performance and computational complexity.

The resulting NSGA-ViT architecture utilizes a combination of 1-dimensional convolutions, multi-head attention mechanisms, and feed-forward layers to produce a Vision Transformer architecture that acheieves 92.35% testing accuracy on CIFAR-10 when trained from scratch. Additionally, NSGA-ViT contains only 47M parameters, roughly half the size of ViT Base, defined by Dosovitskiy et. al [3], which contains roughly 86M parameters. Additionally, NSGA-ViT outperforms the 8-head ViT implementation, containing roughly 44M parameters, by 12% on CIFAR-10, and NSGA-Vit Small outperforms ViT by 7% while containing only 5M parameters.

On ImageNet training, NSGA-ViT under-performs ViT and other transformers, most likely due to the reduced number of pixels being sent to each MHSA head, compared to ViT.

5.1 Limitations

5.1.1 Computational Resources

This study contains a number of limitations due to computational resource constraints. One limitation is the paper only evaluates performance on image classification tasks, and therefore, results cannot be generalized to other computer vision tasks, such as object detection or specialized image recognition tasks. Another limitation of the study is the lack of computational resources available during experimentation. All experiments were done on a single NVIDIA AX5000 GPU. In contrast, many NAS and transformer training paradigms utilize a number of GPUs with data parallelism to speed up training time [5, 3, 6, 13]. With only one GPU, training for 300 epochs takes 30 days, so searching architectures on ImageNet for better performance predictions was infeasible, and performing hyperparameter tuning on ImageNet for architecture evaluation was restricted to a minimum.

Additionally, increasing the input embedding to improve imagenet training is limited by the GPU size. A discussion of future work to tune hyperparameters for ImageNet training is presented in Future Work. However, NSGA-ViT performs well on the CIFAR-10 data-set when trained from scratch, allowing low-resource fine-tuning for other small data-sets [27].

5.1.2 Not Generalized to Other Search Spaces

Furthermore, the search procedure was limited to a specific hand-designed search space, and results cannot be generalized to search space with different operations. Adding more operations to the candidate operations increases the generalizability of the search space, but increases the search space size by an order of magnitude. Since a search of 2000 architectures only covers a fraction of the search space, increasing the size of the search space results in a less robust exploration of the space.

5.2 Future work

5.2.1 Expanding the Search Space

Results for the current study are not generalizable to different search spaces. Because of this, future work may be done applying a similar framework on different search spaces. The NSGA-ViT search space uses attention, convolution, separable convolution, and nonlinearities in the search space. So 2019 [11] includes *lightweight convolutions*, originally proposed by Wu 2019 [41]. As the name suggests, lightweight convolutions are lightweight (low-memory) convolution blocks that were developed as an alternative to attention layers. Including these in the search space may contribute to smaller models, helping the search to produce models that score well on the size objective.

Further, many parameters of the NSGA-ViT transformer were designed by hand, such as the number of layers, input dimension, hidden layer size in feed-forward networks, and patch size. Some of these parameters were set based on expert knowledge, such as a small patch size on CIFAR-10 networks, since many studies indicate that smaller patch sizes improve performance on ViT networks [3, 27, 37]. However, some of these parameters, such as size, input dimension, and hidden layer size may be included in the search space, resulting in an algorithm which requires less expert knowledge to design the final algorithm. There are various stem configurations available in the literature, such as Wu's including replacing the patchify stem with a small convolution stem [37], or utilizing a stem using overlapping patches, demonstrated by Liu et al. [38]. It may be possible to include an encoding for several different stem configurations in the architecture search, leading to a more robust automation of the design process.

Additionally, the search may be configured to to handle variable length encoding, as done by So et. al [11], who leave the number of blocks as a variable in the encoding, as well as encoding for input and output dimension of every operation. Expanding the search space in this way may lead to difficulty utilizing certain types of *crossover* in a many evolutionary computing packages, however, as presented by He 2021 [29], crossover is generally considered non-essential in evolutionary neural architecture search. furthermore, the length of the encoding could be expanded to include an individual encoding for every cell of the network, allowing a certain level of control over the macro architecture.

All of the mentioned additions to the search space will increase the size of the space by orders of magnitude, and therefore will require further methods to be taken to improve the efficiency or speed of the search to ensure a robust optimization within the search space.

5.2.2 Increasing Search Speed

All searches in NSGA-ViT were completed on a single NVDIA AX5000 GPU with no data parallelism. Because of this, only one architecture could be trained at a time, meaning fitness evaluation is a bottleneck for the evolutionary search. In the future, there are various options available to increase the speed of the search, although some may result in less valid fitness evaluations.

The first and most simple method would be to eliminate the resource constraints. By utilizing data parallelism and multiple GPUs, training can be performed on multiple networks simultaneously, reducing the time required to evaluate an entire population. Furthermore, *Surrogate Assisted NAS* utilizes machine learning algorithms which estimate the fitness of each individual, eliminating the need for training of every model, which may speed up evaluation signifigantly. However, this method requires a data-set of architectures to train the surrogate algorithm, which may not be available, applicable (valid), or require significant initialization time if done from scratch.

5.2.3 Hyperparameter Tuning for ImageNet training

In this study, only one set of hyperparameters were used for training on ImageNet. I followed the ImageNet-1k training parameters set by Dosovitskiy et. al [3], e.g. Adam learning rate, learning rate of 0.0001, cosine annealing scheduler, linear warm-up, and high weight decay of 0.3. However, training under these parameters leads to NSGA-ViT under-fitting the data-set. Possible improvements to the training scheme to be done in future work could be altering the learning rate scheduler (cosine annealing, linear, hybrid) and maximum learning rate, as fine-tuning these parameters is essential for effective training. Further, I followed Dosovitskiy et. al in utilizing a high weight decay, anticipating overfitting, however, this parameter could be dialed back to evaluate its effect on the results. Further, image augmentation schemes which have become popular for ViT training, such as Mixup and CutMix are not supported by the version of pytorch used for this project, and future implementations may include these new data processing steps to assist with training [27]. As shown in **??** when training a similar sized ViT configuration with standard training parameters, it under-performs NSGA-ViT and state-of-the-art transformer models. This indicates that more advanced training procedures, utilized in most of these models [40, 39, 38, 28] is essential for achieving successful training in ImageNet-1k.

In addition, parameters that may be adjusted are the number of heads in the MHSA layers and the embedding dimension. Having the right ratio between image size, embedding size, and number of attention heads is crucial for ensuring effective global representations in the network [3, 15]. In this study, the network size was configured for CIFAR-10, with NSGA-ViT utilizing 16-head attention layer (Figure 4.4) and an embedding size of 512, resulting in each head getting 32 pixels fed into it. However, this configuration may not be ideal for an image with $224X224$ resolution. Dosovitskiy [3] uses a ratio of 764 embedding size for 8 heads in the attention layer, resulting in 64 pixels per layer. Adjusting the embedding size for future parameter tuning on ImageNet may be effective for increasing its performance, however, this increases the size of the network to roughly 167M parameters.

REFERENCES

[1] A. Vaswani *et al.*, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, I. Guyon *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017.

[2] “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[3] A. Dosovitskiy *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *International Conference on Learning Representations*, 2021.

[4] B. Zoph and Q. Le, “Neural architecture search with reinforcement learning,” in *International Conference on Learning Representations*, 2017.

[5] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” pp. 8697–8710, Jun. 2018.

[6] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, “Regularized evolution for image classifier architecture search,” 2019, p. 19.

[7] H. Liu, K. Simonyan, and Y. Yang, “DARTS: Differentiable architecture search,” in *International Conference on Learning Representations*, 2019.

[8] Z. Lu *et al.*, “Nsga-net: Neural architecture search using multi-objective genetic algorithm,” ser. GECCO ’19, Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 419–427, ISBN: 9781450361118.

[9] “Multiobjective evolutionary design of deep convolutional neural networks for image classification,” *IEEE Transactions on Evolutionary Computation*, vol. 25, pp. 277–291, 2 Apr. 2021.

[10] K. T. Chitty-Venkata, M. Emani, V. Vishwanath, and A. K. Somani, “Neural architecture search for transformers: A survey,” *IEEE Access*, vol. 10, pp. 108 374–108 412, 2022.

[11] D. So, Q. Le, and C. Liang, “The evolved transformer,” in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Sep. 2019, pp. 5877–5886.

[12] X. Shi, P. Zhou, W. Chen, and L. Xie, *Darts-conformer: Towards efficient gradient-based neural architecture search for end-to-end asr*, 2021. arXiv: 2104.02868 [cs.SD].

[13] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, “Transformers in vision: A survey,” *ACM Computing Surveys*, vol. 54, no. 10s, pp. 1–41, Jan. 2022.

[14] T. Xiao, M. Singh, E. Mintun, T. Darrell, P. Dollar, and R. Girshick, “Early convolutions help transformers see better,” in *Advances in Neural Information Processing Systems*, vol. 34, Curran Associates, Inc., 2021, pp. 30 392–30 400.

[15] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, *Scaling vision transformers*, 2022. arXiv: 2106.04560 `[cs.CV]`.

[16] X. Chen, S. Xie, and K. He, *An empirical study of training self-supervised vision transformers*, 2021. arXiv: 2104.02057 `[cs.CV]`.

[17] T. Wolf *et al.*, *Huggingface’s transformers: State-of-the-art natural language processing*, 2020. arXiv: 1910.03771 `[cs.CL]`.

[18] A. Paszke *et al.*, *Pytorch: An imperative style, high-performance deep learning library*, 2019. arXiv: 1912.01703 `[cs.LG]`.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012.

[21] C. Szegedy *et al.*, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[22] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[23] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (canadian institute for advanced research),”

[24] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi, *Escaping the big data paradigm with compact transformers*, 2022. arXiv: 2104.05704 `[cs.CV]`.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[26] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, “Revisiting unreasonable effectiveness of data in deep learning era,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 843–852.

[27] Y.-H. Cao, H. Yu, and J. Wu, “Training vision transformers with only 2040 images,” in *Computer Vision – ECCV 2022*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T.

Hassner, Eds., Cham: Springer Nature Switzerland, 2022, pp. 220–237, ISBN: 978-3-031-19806-9.

[28] L. Yuan *et al.*, “Tokens-to-token vit: Training vision transformers from scratch on imagenet,” in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 538–547.

[29] X. Chen, S. Xie, and K. He, “An empirical study of training self-supervised vision transformers,” pp. 9620–9629, Oct. 2021.

[30] D. Adiwardana *et al.*, *Towards a human-like open-domain chatbot*, 2020. arXiv: 2001.09977 `[cs.CL]`.

[31] P. Ren *et al.*, “A comprehensive survey of neural architecture search: Challenges and solutions,” *ACM Computing Surveys*, vol. 54, 4 Jul. 2021.

[32] C. Fellbaum, *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[33] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[34] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7132–7141.

[35] J. H. Holland, “Genetic algorithms and adaptation,” in *Adaptive Control of Ill-Defined Systems*, O. G. Selfridge, E. L. Rissland, and M. A. Arbib, Eds. Boston, MA: Springer US, 1984, pp. 317–333, ISBN: 978-1-4684-8941-5.

[36] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, “A survey on evolutionary neural architecture search,” *IEEE Transactions on Neural Networks and Learning Systems*, Feb. 2023.

[37] H. Wu *et al.*, “Cvt: Introducing convolutions to vision transformers,” in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 22–31.

[38] Z. Liu *et al.*, “Swin transformer: Hierarchical vision transformer using shifted windows,” in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2021, pp. 9992–10 002.

[39] Z. Zhang, H. Zhang, L. Zhao, T. Chen, S. Ö. Arik, and T. Pfister, “Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 3, pp. 3417–3425, Jun. 2022.

[40] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, “Training data-efficient image transformers; distillation through attention,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 10 347–10 357.

[41] F. Wu, A. Fan, A. Baevski, Y. Dauphin, and M. Auli, “Pay less attention with lightweight and dynamic convolutions,” in *International Conference on Learning Representations*, 2019.

[42] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, 2023. arXiv: 1606.08415 [cs.LG].

Appendices

APPENDIX A
SEARCH DETAILS

A.1 Candidate Operations

The following is a list of the candidate operations utilized in the search space.

- 4-head self-attention
- 8-head self-attention
- 16-head self-attention
- Feed-Forward Network
- Skip connection
- GELU [42]
- 1x1 convolution
- 3x1 convolution
- 3x1 depth-wise separable convolution
- 5x1 depth-wise separable convolution
- 7x1 depth-wise separable convolution
- 9x1 depth-wise separable convolution
- 11x1 depth-wise separable convolution

A.2 Evolution

NSGA-II is used to optimize competing objectives, the validation accuracy on CIFAR-10, and minimizing the number of floating point operations (FLOPS) in the network. Evolution is run for 50 generations, with a population size of 40 (2000 fitness evaluations), and survival selection between both children and offspring. Since NSGA-II is an elitist search, the best architectures always remain in the population.

Selection. Parent selection is done though tournament selection with size = 2. parents are selected at random for the tournament, and the highest ranking individual always wins.

Crossover NSGA-ViT utilizes 2-point crossover with a probability of 0.9. [8] demonstrates the effectiveness of crossover in NAS using NSGA-II, while traditional methods typically only use mutation for child production. I include crossover to remain consistent with [8].

Mutation. Polynomial mutation with integer rounding is used at a probability of 0.02. This mutation has a higher likelihood to exchange an integer in the genome for a close value, and a lower likelihood to mutate to a value further away.

Survival survival selection is done in a $\mu + \lambda$ schema, meaning the selection pool consists of parents and children. All the individuals are sorted and ranked based on their performance on the objectives, and a crowding metric is used to preserve diversity across the pareto-front. The best performing individuals move onto the next generation in an elitist fashion.

A.3 CIFAR-10

To carry out architecture search, I hold out half the CIFAR-10 training data as a validation set. A small network of 3 cells with an initial embedding dimension of 256 is trained using Adam for 20 epochs for each architecture in the search. CIFAR-10 training during the search allows for a reduced training time, and Zoph et al. [5] indicated that performance on CIFAR-10 is a good estimator of performance on larger datasets, such as ImageNet.

A.4 Architecture Details

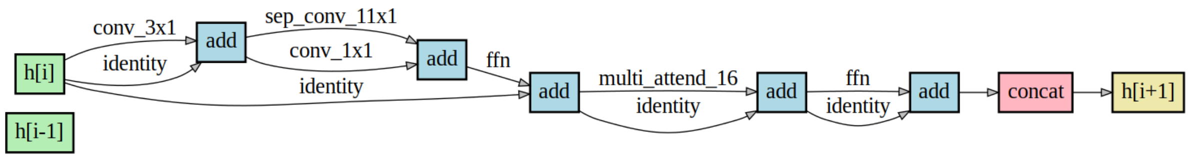
During search, all networks are implemented with 3 layers and an initial embedding dimension of 256. During cold-start search, the embedding dimension increases proportionally to the width of the cell, as final hidden states are concatenated together. each cell is initialized with 5 blocks. In the warm-start search, a 5 block ViT implementation is used for the initialization, where the first block consists of skip-connections from the input. This implementation allows for more possible architectures while retaining essential parts of the ViT architecture.

Per Vaswani and Dosovitsky, [1, 3] Feed Forward operations in the search pool are implemented using $1x1$ Convolution projections rather than linear layer embeddings. All FFNs in the search operations use two layers with GELU nonlinearity on the hidden layer. Embedding sizes for the input, hidden layer, and output, are $(N, 4N, N)$ where N is the embedding dimension.

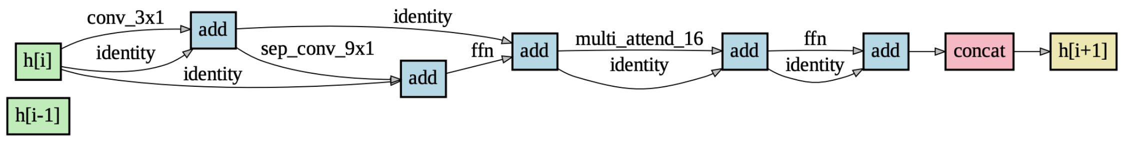
Additionally, the base ViT network from Dosovitskiy et. al [3] utilizes 12 heads and 12 layers with an embedding dimension of 768, resulting in a network with 86M parameters. Due to resource limitations, particularly, using 1 GPU, ViT networks are all implemented with 8-heads and either 256 channels or 512 channels depending on the network depth.

APPENDIX B
NSGA-VIT ARCHITECTURES

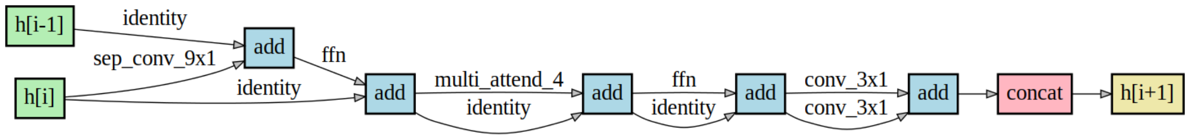
This section presents visualizations of all the NSGA-ViT architectures found in the experimental procedure. NSGA-ViT-A architectures were discovered in the cold-start setting, and NSGA-ViT-B cells were found in the warm-start setting.



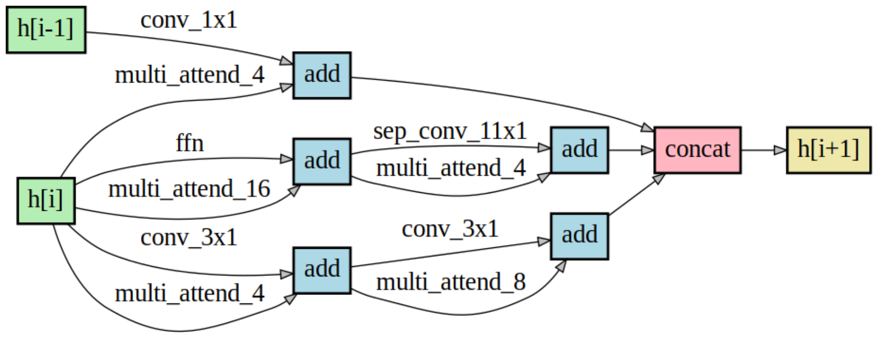
(a) NSGA-ViT



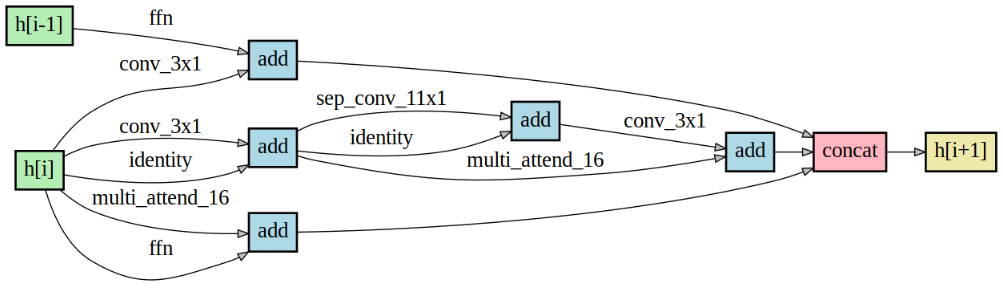
(b) NSGA-ViT-B



(c) NSGA-ViT-B2



(d) NSGA-ViT-A



(e) NSGA-ViT-A2

Figure B.1: **Architectures Discovered in NSGA-ViT**

APPENDIX C
ARCHITECTURE EVALUATION DETAILS

C.1 Data-sets and Processing

C.1.1 CIFAR-10

During CIFAR-10 training, images use a size of $32X32$ and are processed using random horizontal flip and random rotation, as well cutout where applicable. Batches of 128 images are used for training and validation.

C.1.2 ImageNet

ImageNet images are cropped using a random center crop of size = $224X224$ and processed using random horizontal flip, random rotation, and cutout, where applicable. Batches of 64 images are used for training and validation.

C.2 Training Parameters

This section presents the parameters used for architecture validation.

Config	epochs	optim	lr	scheduler	dropout	cutout	decay	data
3 @ 256	200	AdamW	0.0001	cosine	-	-	-	CIFAR-10
6 @ 512	200	AdamW	0.0001	cosine	-	-	-	CIFAR-10
6 @ 512	200	AdamW	0.0001	cosine	0.2	16	0.3	CIFAR-10
6 @ 512	600	AdamW	0.0001	cosine	0.2	-	0.3	CIFAR-10
6 @ 512	100	AdamW	0.0001	cosine	0.1	-	0.1	ImageNet
6 @ 512	300	AdamW	0.0001	cosine	0.1	-	0.1	ImageNet

Table C.1: Configurations and parameters for architecture training. Configurations are presented as *cells @ init channels*. *lr* stands for learning rate, and *decay* refers to weight decay in Adam

APPENDIX D
IMPLEMENTATION DETAILS

D.1 Software Implementation

Architecture Search and Validation are implemented in python using Pytorch, Torch Vision, and Pymoo package for implementing NSGA-II. The code for training and implementing models is inspired by and modified from (Liu 2019) [7] and code for performing NAS is inspired by and modified from Lu et.al [8]. All Transformer architectures are implemented by modified code from PyTorch source code [18], utilizing the VisionTransformer class as the blueprint for the network backbone, including the patchify or convolution stem, image flattening, positional embeddings, and class tokens. Code for this paper can be found at https://github.com/drewbecker02/nsga-vit/.

D.2 Experimental Equipment

Training and Search were performed on a single NVIDIA AX5000 GPU. Training during search is performed with 8 workers, and validation during search is performed with 4 workers.