12-2023

Generalized Differentiable Neural Architecture Search with Performance and Stability Improvements

Emily J. Herron
University of Tennessee, Knoxville, eherron5@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Emily J. Herron entitled "Generalized Differentiable Neural Architecture Search with Performance and Stability Improvements." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Data Science and Engineering.

Steven R. Young, Major Professor

We have read this dissertation and recommend its acceptance:

Steven R. Young, Catherine Schuman, Amir Sadovnik, Derek Rose

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Generalized Differentiable Neural Architecture Search with Performance and Stability Improvements

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Emily Herron

December 2023

Abstract

This work introduces improvements to the stability and generalizability of Cyclic DARTS (CDARTS). CDARTS is a Differentiable Architecture Search (DARTS)-based approach to neural architecture search (NAS) that uses a cyclic feedback mechanism to train search and evaluation networks concurrently, thereby optimizing the search process by enforcing that the networks produce similar outputs. However, the dissimilarity between the loss functions used by the evaluation networks during the search and retraining phases results in a search-phase evaluation network, a sub-optimal proxy for the final evaluation network utilized during retraining. ICDARTS, a revised algorithm that reformulates the search phase loss functions to ensure the criteria for training the networks is consistent across both phases, is presented along with a modified process for discretizing the search network's zero operations that allows the retention of these operations in the final evaluation networks. We pair the results of these changes with ablation studies of ICDARTS' algorithm and network template. Multiple methods were then explored for expanding the search space of ICDARTS, including extending its operation set and implementing methods for discretizing its continuous search cells, further improving its discovered networks' performance. In order to balance the flexibility of expanded search spaces with minimal compute costs, both a novel algorithm for incorporating efficient dynamic search spaces into ICDARTS and a multi-objective version of ICDARTS that incorporates an expected latency penalty term into its loss function are introduced. All enhancements to the original search algorithm are verified on two challenging scientific datasets. This work concludes by proposing and examining the preliminary results of a preliminary hierarchical version of ICDARTS that optimizes cell structures and network templates.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The methods presented in this work seek to advance the state-of-the-art of NAS by introducing a series of improvements to an existing gradient-based NAS framework. These include algorithmic changes that improve consistency between the networks of its search and retraining phases, leading to improved stability and more consistent results. Alongside these changes, methods for extending its search space are explored, resulting in further performance improvements. Novel methods for incorporating a dynamic operation space and a multi-objective loss function into the search phase are also introduced to accommodate expansions to the search without incurring excessive compute expense. Finally, a hierarchical version of these methods that allows network templates to be optimized in addition to cell structures is proposed. The best improvements are verified by deploying the search process to two scientific image tasks.

1.1 Neural Architecture Search

Neural architecture search (NAS) algorithms automatically design network architectures given minimal human input and limited computing resources. NAS algorithms typically frame the automation process as a search problem over a set of decisions representing different network architecture components. Most include a search space that defines a set of feasible solutions, a search strategy defined by an optimizer that solves for the best architecture for a given search space, and an evaluation strategy that estimates the performance of

discovered architectures on unseen data. Typically, a controller selects a set of candidate neural architectures from a search space of operation sets. These architectures are trained and ranked according to some evaluation metric, such as validation accuracy. The rankings of the candidate architectures provide feedback to the search strategy, which is updated and produces a new set of candidate architectures until a terminating condition is met. One of the earliest categories of NAS search strategies relied on an evolutionary approach to produce optimal candidate architectures by evolving populations of networks over several generations. More recently, NAS implementations have employed a search strategy that frames the search process as a reinforcement learning task, with the action typically representing the generation of candidate architectures and the reward being the candidate's performance on a test dataset.

Although the best implementations of both of these search strategies produce state-of-the-art results, a significant drawback of these early RL and EA-based NAS algorithms is their high computational expense, often exceeding the resources and capabilities of many researchers, and limiting the progress of research on this subject, which is in part due to these approaches generating and evaluating large numbers of candidate architectures. Several follow-up NAS implementations have sought to address this issue, by introducing innovations like modular search spaces and parameter-sharing methods. Modular search spaces, such as the one presented in NASNet Zoph et al. (2017), call for optimizing smaller modules or cells that can be copied and stacked according to some template to form the final architecture rather than searching for entire network architectures simultaneously. NAS methods that use parameter sharing allow weights to be shared across candidate networks by recycling weights across transformations to an existing architecture or by representing the search space with a large directed acyclic graph structure with subgraphs selected by the controller representing candidate architectures.

1.2 Differentiable Neural Architecture Search

Another class of NAS methods uses a gradient-based search strategy to efficiently search for optimal network architectures given a continuous and differentiable search space. One of the

most prominent gradient-based NAS approaches, DARTS, borrows both the modular search space of NASNet and the one-shot model graph parameter sharing approach of ENAS. However, instead of using a controller to select discrete candidate architectures from the search graph, DARTS relaxes the model graph into a continuous search space by placing a mixture of candidate operations at each edge and optimizing a set of weights associated with each operation using stochastic gradient descent. After the candidate operation weights have been optimized, the final architecture is generated by 'discretizing' the continuous search graph by discarding all but the highest probability operations at each edge, and the entire architecture is retrained from scratch. A disadvantage of the gradient-based search strategy presented by DARTS was the limited correlation between the shallow and overparameterized network optimized in the search phase and the final network architecture generated in the retraining phase. Several follow-up publications have sought to address this issue and build upon the DARTS algorithm. These include P-DARTS, which improves the correlation between the search and final networks by progressively increasing the depth of the search network while removing lower probability operations from the search graph. Another framework, CDARTS, builds upon P-DARTS by jointly optimizing search and final architectures using a cyclic approach that allows the networks to supply optimized architectures and performance-based feedback to each other during the search process.

However, the abovementioned efficiency improvements limit the correlations between the deep final networks and the shallow and overparameterized proxy networks, along with the diversity of candidate architectures. Furthermore, most NAS approaches in literature have been designed with traditional photographic image classification tasks in mind, with fewer implementations existing for types of tasks, including scientific datasets.

1.3 Contributions

The contributions of this work include:

- The introduction of ICDARTS, a modified version of the CDARTS algorithm in which the evaluation network is consistent across both the search and retraining phases, improving the algorithm's stability and consistency in results across multiple runs.

- The expansion of ICDARTS' operation space to include operations with varying complexities.
- A novel tournament-style algorithm for incorporating dynamic operation spaces into ICDARTS enables the algorithm to traverse large operation spaces efficiently.
- An alternative protocol for discretizing search cells that expands and adds diversity to the space of discoverable architectures.
- The introduction of a latency loss penalty term that enables ICDARTS to balance high performance with low compute costs.
- Experimental results demonstrating that ICDARTS extends well to real-world scientific datasets.

1.4 Publications

- Herron, E. J., Young, S. R., and Rose, D. C. (2023). ICDARTS: Improving the Stability and Performance of Cyclic DARTS, Submitted for publication to Journal of Machine Learning Research (JMLR).
- Herron, E. J., Young, S. R., and Potok, T. E. (2022). ICDARTS: Improving the Stability of Cyclic DARTS. In 2022, 21st IEEE International Conference on Machine Learning and Applications (ICMLA), IEEE.
- Duncan, J., Fallas, F., Gropp, C., Herron, E., Mahbub, M., Olaya, P., Ponce,E., Samuel, T. K., Schultz, D., Srinivasan, S., Tang, M., Zenkov, V., Zhou, Q., and Begoli, E. (2021). The sensitivity of word embeddings-based author detection models to semantic-preserving adversarial perturbations.
- Herron, E. J., Young, S. R., and Potok, T. E. (2020). Ensembles of networks produced from neural architecture search. In Jagode, H., Anzt, H., Juckeland, G., and Ltaief, H., editors, High Performance Computing, pages 223–234, Cham. Springer International Publishing.

1.5 Outline

Chapter 2 contains a literature review of the field of NAS. It begins by discussing the motivations for the field and the earliest NAS methods, introduces gradient-based NAS, and discusses ongoing challenges in the field. Chapter 3 introduces improvements to the algorithm and operation space of CDARTS that enhance its stability and performance. The necessity of these changes is further verified with ablation studies. Chapter 4 details further improvements, specifically expansions to the algorithm's search space. This chapter introduces a novel approach for efficiently traversing large search spaces, a new method for discretizing search networks that increases the diversity of discoverable architectures, and a multi-objective version of ICDARTS that optimizes performance and latency. Chapter 5 discusses the results of applying the best overall improvements to ICDARTS to two challenging scientific image datasets. Chapter 6 proposes a preliminary hierarchical version of ICDARTS that optimizes network templates in addition to cell structures. Finally, Chapter 7 summarizes these contributions and lays out possible directions for future improvements and study.

Chapter 2

Background and Literature Review

This chapter discusses the motivations behind neural architecture search (NAS) algorithms and provides an overview of the field's history and current state-of-the-art. It begins with a discussion of early NAS algorithms and their limitations. It then discusses methods for improving the efficiency and performance of early NAS algorithms, including modular search spaces and gradient-based NAS. The remainder of the chapter discusses the most recent innovations in differentiable NAS and the methodology to ensure that these methods efficiently discover high-quality and scalable architectures.

2.1 Motivation

Deep learning methods have succeeded in a multitude of tasks, including image recognition (Tan and Le, 2021), object detection (Redmon and Farhadi, 2017; He et al., 2017; Wang et al., 2020b), semantic segmentation (Nekrasov et al., 2019), speech recognition Pan et al. (2020), and machine translation Gehring et al. (2017). The strong generalization capabilities of deep learning models are primarily attributable to their ability to engineer an abstract hierarchy of features given an unstructured data set by learning automatic feature extractors. This paradigm shift has eliminated the laborious need for researchers to manually engineer features for unstructured datasets and allowed them to pivot their focus to the design of deep learning architectures. However, engineering these increasingly complex models is labor-intensive and error-prone. When encountering a new task, machine learning researchers

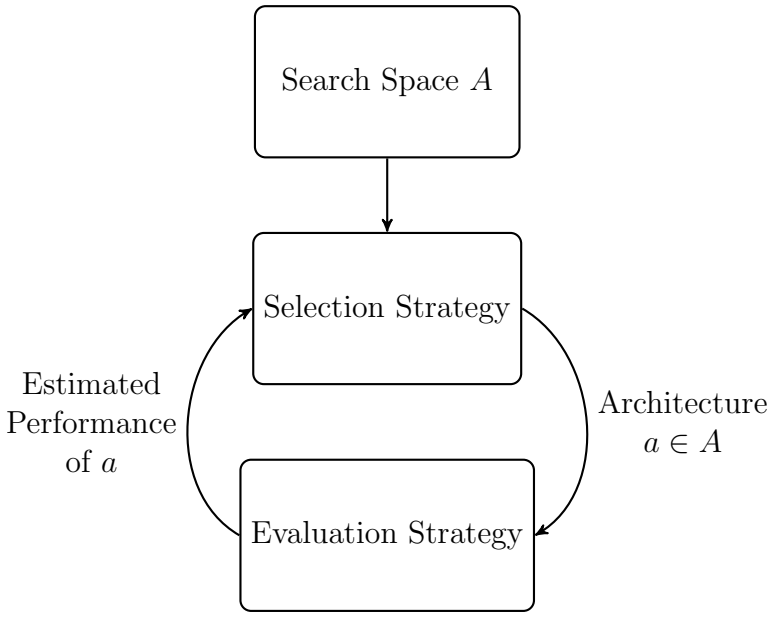
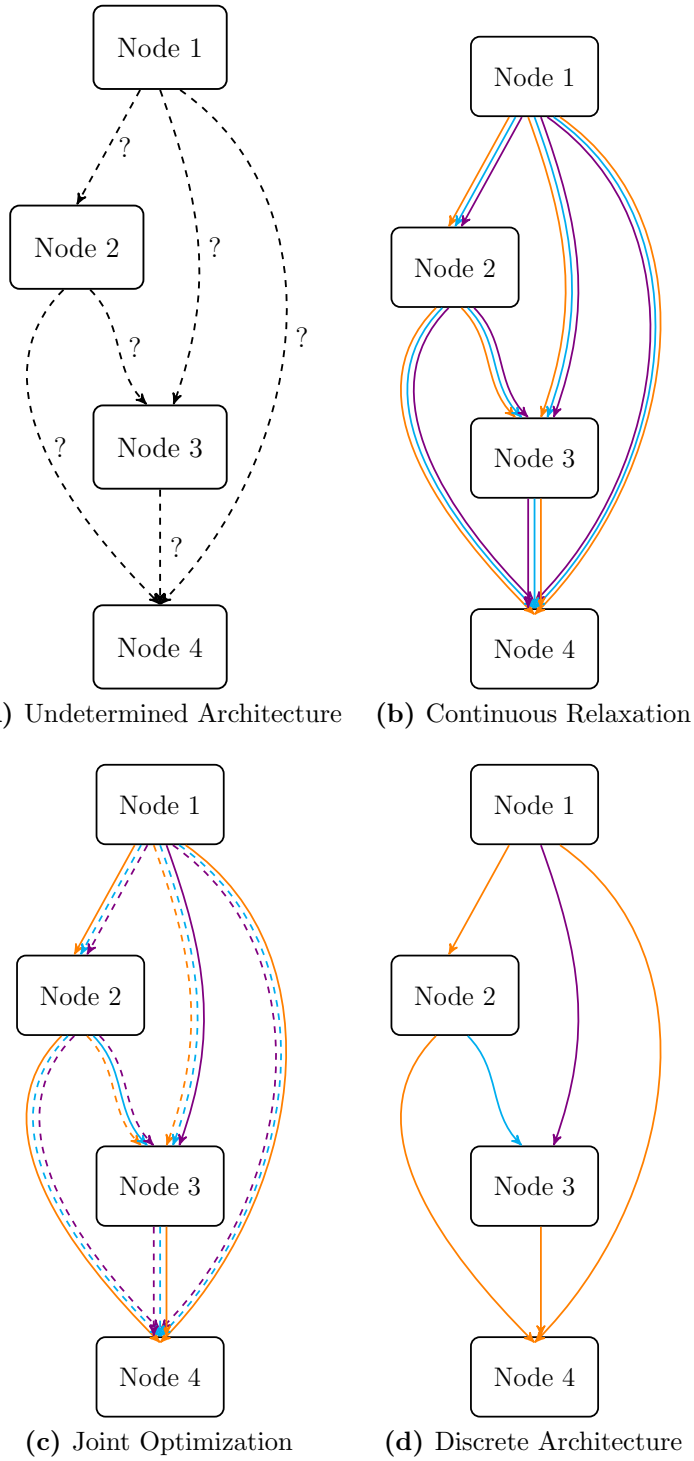


Figure 2.1: The general NAS pipeline. A controller selects candidate architectures, a, from the search space A based on the selection strategy. The evaluation strategy then ranks and evaluates the candidate architectures. The results of the evaluation strategy are supplied as feedback to the selection strategy, and the cycle repeats until a terminating condition is met.

Table 2.1: Timeline of Key NAS Methods Leading Up to CDARTS with Contributions

1989	**Innervator** Miller et al. (1989) - Evolutionary NAS
2017	**NAS-RL** Zoph and Le (2016) - Reinforcement Learning Based NAS
2018	**ENAS** Pham et al. (2018) - Parameter Sharing
2018	**NASNet** Zoph et al. (2017) - Cell Search Spaces
2018	**DARTS** Liu et al. (2018b) - Gradient-based Optimization
2019	**P-DARTS** Chen et al. (2019) - Progressive Correlation
2020	**CDARTS** Yu et al. (2020) - Cyclic Optimization



(a) Undetermined Architecture **(b)** Continuous Relaxation

(c) Joint Optimization **(d)** Discrete Architecture

Figure 2.2: The continuous relaxation and discretization of the DARTS cell-level search space. (a) shows the general structure of the cell's directed acyclic graph, (b) the continuous relaxation of the search graph so that a mixture of candidate operations represents each edge, (c) the search cell following joint optimization of the probabilities of the mixed candidate operations and network weights, and (d) the final discrete architecture.

often deploy a state-of-the-art deep learning architecture designed for and evaluated on a limited number of popular benchmark datasets. Since image datasets can vary widely in size, resolution, and subject matter, a single state-of-the-art network architecture that generalizes well to one task may perform poorly on a different dataset.

Furthermore, researchers' knowledge of architectures suitable for specific tasks is often limited to knowledge of models in existing literature. This fixed thinking based on limited knowledge may blind experts to categories of architectures that have yet to be discovered. Research into the explainability as to why particular deep learning architectures and features are effective is ongoing. It has yet to deliver tools to propose architectures for a given dataset Wistuba et al. (2019). As a result, the Neural Architecture Search (NAS) field has emerged to address these concerns by automating the search process.

NAS algorithms automatically engineer network architectures while utilizing minimal computing resources and human input. They have been successfully applied in image classification, object detection, semantic segmentation, adversarial learning, speech recognition, and multi-objective optimization. However, much of the literature in the field concentrates on image classification tasks. NAS algorithms typically frame the automation process as a search problem over a set of decisions comprising a neural architecture's different components. NAS frameworks typically include a *search space*, a *search strategy*, and an *evaluation strategy*.

The *search space* defines the set of feasible solutions to a NAS method and is typically represented by a set of operations and constraints on a network. The *search strategy* is the protocol for searching an optimal architecture given the search space. The *evaluation strategy* (also known as the performance estimation strategy) estimates the performance of a discovered architecture on unseen data. In earlier NAS approaches, a controller selects a set of candidate neural architectures from a search space of operation sets. These architectures are then trained and ranked based on an evaluation metric, such as accuracy on a validation dataset. The rankings of the candidate architectures are supplied as feedback to the search strategy. The search strategy is cyclically updated and produces a new set of candidate architectures at each iteration until a terminating condition is reached. Figure 2.1 depicts this general process.

2.2 Early Neural Architecture Search

The earliest NAS methods date back over 30 years and relied on neuro-evolutionary approaches Miller et al. (1989). Most early evolution-based NAS approaches optimized the architectures and weights of candidate networks Angeline et al. (1994); Stanley and Miikkulainen (2002); Stanley et al. (2009). However, more recent solutions use gradient-based optimization to learn the network weights, reserving the evolutionary methods for optimizing the neural architectures Real et al. (2017); Suganuma et al. (2017); Real et al. (2019); Xie and Yuille (2017a); Elsken et al. (2019), as this strategy has proven most effective for scaling to contemporary architectures with millions of weights for supervised learning tasks. Evolutionary NAS algorithms evolve populations of architectures. At each iteration, a subset of architectures are sampled based on fitness (i.e., accuracy on the validation dataset) to serve as parents to the population's next generation. Offspring are generated by applying mutations to the architectures of parents, typically presenting as a minor change such as the removal or addition of a layer, a change to the layer's hyperparameters, or adding a skip connection. Evolution-based NAS approaches vary in their methods for selecting parents and generating offspring. Methods for selecting parents include using a tournament-based selection approach Real et al. (2019, 2018), sampling from a multi-objective Pareto front Elsken et al. (2019), and removing the worst or oldest individuals Real et al. (2019); Elsken et al. (2019). Methods for generating offspring range from random initialization to passing learned weights from parents to children through network morphisms Elsken et al. (2018) to allowing inheritance of parameters not affected by mutationsReal et al. (2018); Elsken et al. (2019).

Another prominent class of NAS methods frames the search problem as a Reinforcement Learning (RL) task. Most of these methods designate the search space as the action space and the generation of a candidate architecture as the RL agent's action. The agent's reward is the candidate's performance on a test dataset Zoph and Le (2017); Elsken et al. (2019). NAS-RL Zoph and Le (2016) was a pioneering RL-based method for NAS that brought the field mainstream attention in the machine learning community by achieving state-of-the-art results on the CIFAR-10 and Penn Treebank benchmark. This algorithm used a

recurrent neural network policy to generate a string signifying a candidate architecture. The initial implementation trained the policy with the REINFORCE policy gradient algorithm, although later editions used proximal policy optimization. A follow-up to NAS-RL, MetaQNN Baker et al. (2016) similarly applies RL to NAS by framing the selection task as a Markov decision process and applying Q-learning to train the policy. Case studies comparing random search, RL, and evolution-based NAS methods found that RL and evolution-based methods performed equally in terms of final accuracy Real et al. (2019).

Other early NAS strategies include surrogate model-based optimization (SMBO) Hutter et al. (2011), in which a surrogate model evaluates and selects promising candidate cells and calculates response values for these cells, which are used to update the model, Bayesian Optimization methods Shahriari et al. (2016), and Monte Carlo Tree Search methods, which represent and exploit the search space as a tree structure Negrinho and Gordon (2017); Elsken et al. (2019). A significant downside to each of these categories of NAS is their high computational expense, as each consumes hundreds of GPU hours, exceeding the resources and capabilities of many researchers and limiting the research progress on this subject.

Modular search spaces were one solution to the high computing costs of early NAS methods. Early NAS methods optimized candidate architectures within global search spaces. Global search spaces allow NAS methods to discover entire neural architecture graphs simultaneously and with a high degree of freedom. The earliest global search spaces were chain structured, and discovered architectures were represented as an ordered sequence of layers such that each layer had only the previous layer as its parent and took only the output of the previous layer as input Wistuba et al. (2019); Ren et al. (2020). However, since Xie and Yuille (2017b), the preferred structure of a global search space has shifted to sequentially connected segments ordered according to a template. A follow-up publication to NAS-RL Zoph et al. (2017) introduced the NASNet search space, a modular search space designed based on the observation that many state-of-the-art networks consist of repeatedly stacked units. Rather than simultaneously generating the entire network architecture graph, NAS-RL's controller generates modules or cells that are copied and stacked as part of the complete architecture. This architecture is then trained and evaluated to supply feedback to the controller. The complete network architecture's structure and the placement of its cells

are arranged according to a pre-defined template, in which cell topologies remain consistent. Hyperparameters, such as expected input dimensions and the number of channels associated with convolution operations, associated with each cell may vary based on their depth in the network.

The NASNet search space includes both normal and reduction cells. The primary purpose of normal cells is to extract prominent features, while the reduction cells reduce the dimensions of the inputs. Each cell consists of k nodes, defined by two inputs and one operation. The possible inputs for each cell include the outputs of the last two cells (allowing for skip connections between cells) and the outputs of a previous node in the cell. The outputs of each cell are the concatenation or summation of all nodes that are not input into any other node. The final architecture consists of multiple normal cells repeating motifs followed by a single reduction cell. Several subsequent works have suggested eliminating the need for reduction cells. For instance, Block-QNN replaced the reduction cells with a simple pooling operation Zhong et al. (2018), Dpp-net replaced the reduction cells with average pooling Dong et al. (2018), and Hierarchical-ENAS replaced reduction cells with a 3×3 convolution with a stride of 2.

Cell-based search spaces offer improved efficiency and flexibility by reducing the search to a few cell motifs rather than an entire architecture. The cells can then be applied to other tasks by stacking different numbers of cells to have the appropriate network capacity. The search space is reduced by searching for a few cell motifs rather than an entire architecture. The authors estimate a seven times speedup compared to the global search space of their previous work. Since this search space significantly reduces the computational complexity of earlier NAS approaches, it has been widely adopted in subsequent NAS literature Real et al. (2019); Liu et al. (2018b); Zoph et al. (2017).

Another solution for improving early NAS frameworks' computing efficiency is parameter sharing across multiple architectures Pham et al. (2018); Cai et al. (2017); Bender et al. (2018). One framework that leverages parameter sharing for RL-based NAS is EAS, which employs a meta-controller that traverses the search space using network transformation operations such as widening or inserting a layer or adding skip connections within an existing network. Weights are recycled across transformations by leveraging the class of

function-preserving transformation operations to explore the search space efficiently by taking advantage of knowledge from previously explored models Cai et al. (2017). Another NAS approach that utilizes parameter sharing is ENAS. In this method, the controller discovers networks by searching and evaluating subgraphs within a larger directed acyclic graph. The graph structure of the search space allows weights to be shared between prospective architectures. ENAS achieved a 1000x speed up on the original NAS-RL and NASNet search spaces Pham et al. (2018).

2.3 Gradient-based NAS

A more recent category of efficient architecture search strategy that has become prominent in NAS literature is using a continuous rather than discrete search strategy to enable gradient-based optimation. Early NAS methods optimize architectures within discrete spaces. Differentiable Neural Network Architecture Search (DAS) Shin* et al. (2018) was the first NAS implementation to explore a differentiable continuous search space in NAS by applying gradient-based optimization techniques to search hyperparameters for convolutional layers.

DARTS expands upon this concept by introducing a one-shot model that, rather than searching for discrete architectures, relaxes the search into a continuous space that can be optimized via gradient descent by placing a mixture of candidate operations at each edge of the model. Following the modular search space introduced by NASNet, cells are represented as directed acyclic graphs of N connected nodes. In order to limit GPU memory consumption, the algorithm is partitioned into two consecutive phases: search and retraining. In the search phase, DARTS identifies optimal architecture motifs via a shallow, fully connected network comprised of continuous connections between nodes, $x^{(i)}$, represented by a mixture of candidate operations $o^{(i,j)}$ at each edge $e^{(i,j)}$. Discrete nodes in the search phase can be expressed as $x^j = \sum_{i<j} o^{(i,j)}(x^{(i)})$. DARTS achieves a continuous search space by weighting each operation by the softmax of the corresponding α value of all possible operations at each edge:

$$\bar{o}^{(i,j)}(x_i) = \sum_{o \in O} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})} o(x_i) \tag{2.1}$$

The $\alpha^{(i,j)}$ values are vectors of size $|O|$ that parameterize the candidate operation strength for each edge in the graph. Each intermediate node of the cell is computed by taking the sum of the outputs of the preceding nodes: $x_j = \sum \hat{o}^{(i,j)(x_i)}$. The outputs of each intermediate node are then concatenated to produce the cell's output. The search network consists of repeating cell motifs searched by optimizing the connection weights of the mixture of candidate operations at each edge of its directed acyclic graph (DAG). The objective of this architecture search is to find the search cells' connection weights, α that minimize the validation loss, $L_{val}(w_S^*, \alpha)$ and the weights, w by minimizing the training loss $L_{train}(w, a^*)$. This objective may be framed as the bilevel optimization problem:

$$\begin{aligned} &\min_{\alpha} L_{val}(w_S^*, \alpha) \\ s.t.\ &w_S^*(\alpha) = \arg\min_{w_S} L_{train}(w_S, \alpha) \end{aligned} \tag{2.2}$$

After the topologies of the search cells have been optimized within the search phase, a separate retraining phase occurs in which a deep evaluation network is constructed and retrained to obtain its final performance. Cells are derived by discretizing the continuous operations with the search network cells to construct the final deep network. Discretization is accomplished by removing all operations except those on the directed edges corresponding to the top-k best (where $k = 2$) α values from each previous node with $o^{(i,j)} = argmax_{o \in O} \alpha_o^{(i,j)}$ Zoph et al. (2018); Real et al. (2019); Chen et al. (2019). The continuous relaxation and the discretization of the DARTS cell search space are shown in Figure 2.2. The cell motifs are then stacked to form a deeper network, as specified by a hardcoded architecture template, which is evaluated on a separate test dataset to obtain the final performance. DARTS achieves high-quality architectures through this approach with a reduced computational cost Liu et al. (2019b); Elsken et al. (2019).

The DARTS algorithm is not without its disadvantages. These include the different search and retraining steps that result in two independent networks with limited correlation Cai et al. (2019); Yu et al. (2020), the computational expenses that remain high despite the

improved efficiency that results from differentiable search Dong and Yang (2019), and the local bias that exists in the differentiable cells searched by DARTS Jiang et al. (2019).

Several publications have proposed solutions to the issue of limited correlations between the network architectures of the search and retraining phases. For example, stochastic Neural Architecture Search (SNAS) Xie et al. (2019) argues that the inconsistency between child and parent networks results from the nonlinearity of neural operations, which creates an intractable bias in the loss function that leads to inconsistency between child networks and the parent super-network. The authors' solution is a gradient-based framework that trains network and architecture weights jointly on the same round of backpropagation. As part of this process, one-hot random variables are applied as masks to select operations in the search graph. The result is a stable and more efficient NAS framework that maintains a complete and differentiable pipeline. Progressive DARTS (P-DARTS) Chen et al. (2019) addresses the disparity between the two networks by progressively increasing the depth of the network and decreasing the number of candidate operations during the search phase until it reaches the size of the target model. In addition, this method avoids high computational expense later by gradually dropping lower-scoring candidate operations as the search process progresses.

Cyclic DARTS (CDARTS) Yu et al. (2020) builds upon the concepts introduced by PDARTS by jointly training the search and final evaluation networks and updating the parameters of each with a cyclic method. In each cycle, the search network is translated into an intermediate evaluation network, providing feedback to the search network in the form of distilled features that capture knowledge from the model's structure and parameters. The joint learning of the search and evaluation networks involves independent and joint learning stages. The search and evaluation networks' weights are initialized and pre-trained in the learning phase. In the joint learning phase, the architecture weights (i.e., α) and evaluation network weights are updated based on a joint loss function before updating the search network weights based on a separate loss function and subset of the training data. Thus, this cyclic approach allows for directly evaluating the final deep network.

Other publications have concentrated on the high compute demands that persist despite the efficiency improvements made over previous NAS despite the introduction of differentiable search space. Differentiable Architecture Sampler (GDAS) is a gradient-based

architecture search framework based on Differentiable Architecture Sampling Dong and Yang (2019). Like DARTS, the search is carried out on the cell level represented by a DAG. Singular sub-graphs are sampled from the DAG in which each node only receives one feature from each previous node so that only one path is trained per iteration. Features are sampled for connecting pairs of nodes using a differentiable method. This approach allows cells to be searched via gradient descent, avoiding the memory demands required for optimizing an entire model, as in the case of DARTS. Partially Connected DARTS (PC-DARTS) aims to improve computational efficiency by searching only a subset of channels of the supergraph, bypassing the others using a shortcut. The channel subsets are assumed to be an adequate proxy for evaluating the entire search space of architectures. Channel sampling improves the search process by adding regularization and avoiding local optima. Edge normalization stabilizes the search process by adding a learnable set of edge selection hyperparameters to avoid the instability that results from sampling different combinations of channels at each iteration.

Finally, I-DARTS points out that DARTS is a biased "local" model since softmax-based relaxation is applied to each edge between two nodes to select the highest probability candidate edges Jiang et al. (2019). In DARTS, the search is limited to one edge between each pair of nodes, and edges coming from two different nodes cannot be compared and may be redundant since each node inherits the outputs of all previous nodes. I-DARTS broadens the search space of DARTS by applying a single softmax to all incoming edges in a node. This modification allows for a more extensive range of edge options, multiple edges to be selected between two nodes, and some connections between nodes to be left out. This approach performs exceptionally well on language tasks.

2.4 Global and Hierarchical NAS

Although much of recent NAS literature has shifted its focus towards searching architectures at the micro or cellular level, frameworks for carrying out more efficient searches of entire neural networks at the global level remain desirable. Modular NAS methods' strategy of repeatedly stacking cell motifs limits the complete network architecture's structural diversity

and performance. Templates that dictate the constraints of the architectures and search spaces, including the stacking of network cells, are often designed based on assumptions from existing literature regarding the effectiveness of particular architectures and search spaces. At the same time, the problem of ensuring that the complete network composed of deeper stacks of cell motifs transfers well to the target task remains persistent in NAS literature. Another drawback of cell-based NAS is that networks comprising complex and fragmented cell structures can be inefficient in these contexts, particularly when deploying mobile devices and other real-world hardware targets.

Furthermore, some experimental evidence, such as that unveiled in Wang et al. (2019), suggests that global search spaces may be preferable to cell-based spaces for non-image-related tasks, such as text classification. Hence, NAS solutions that balance the freedom of global search spaces and the high compute costs they require are essential. Global NAS allows for a high degree of freedom and diversity regarding the types of networks that can be discovered. Despite the innovations mentioned in the previous section, search spaces are limited by their high demand for computing resources. Search time and memory demands must be suppressed to conduct searches with minimal search space constraints, such as a global search space.

As discussed in previous sections, early NAS was carried out over global search spaces. Many early evolution and reinforcement learning-based NAS algorithms were prohibitively expensive, a challenge that eventually led to the introduction of cell-based search spaces and gradient-based optimization strategies. ProxylessNAS Cai et al. (2019) was among the first gradient-based NAS methods to search for complete architectures over a substantial candidate set rather than repeating cells by directly learning these architectures on target tasks and hardware. ProxylessNAS is formulated as a path-level pruning process in which an over-parameterized network containing all candidate paths is directly trained, and redundant paths are pruned at the end of training to get a compact architecture. The architecture parameters are binarized to limit the GPU memory consumption required by including all candidate paths so that only one path is active at run time. This strategy reduces the memory required for training by one order of magnitude to that of a compressed model, and the binarized parameters are trained with a gradient-based approach. ProxylessNAS

achieved state-of-the-art performance on a more extensive search space under computational costs similar to regular training and was the first NAS method to learn architectures for deployment to specialized hardware targets.

Other global NAS solutions strove to improve efficiency by curbing the parameters required by discovered architectures by introducing multi-objective loss functions. MNASNet Tan et al. (2019) is a reinforcement learning-based framework for mobile CNN design formulated as a multi-objective optimization problem. This method seeks to optimize a reward function by maximizing the searched networks' accuracy while maintaining a real-world inference latency beneath a fixed constant. Its factorized hierarchical search space partitions the network into blocks for which separate layers are searched and stacked repeatedly. This solution allows for greater flexibility and diversity in layer type without requiring an enormous search space size. Later work Saito and Shirakawa (2019) introduced a penalty term into a dynamic structure optimization method intended to control the complexity of the discovered models. The dynamic structure optimization framework initially used gradient methods to jointly optimize architecture distribution parameters and weights. The authors note that a drawback of the multi-objective optimization strategy of MNASNet and related work is the high computational expense, as this method is based on hyperparameter optimization. Therefore, a penalty term for the layer weights is incorporated into the method's loss-based objective function to limit the complexity of the model. Experiments showed that this penalty term effectively controlled the model's complexity and maintained its performance by removing units that were not significant to improving the performance.

The Differential Neural Architecture Search (DNAS) framework presented in Wu et al. (2018) expands upon the progress made by ProxylessNAS and MNASNet by using a gradient-based approach to find hardware-efficient convolutional neural networks. The Gumbel Softmax technique is leveraged to optimize the architecture distribution using gradient-based techniques such as SGD. A layerwise search is carried out over a large super-network, defined by a fixed macro-architecture template, in which one of nine candidate block operations is selected per layer. These candidate blocks are inspired by MobileNetV2 Sandler et al. (2018) and ShiftNet Yan et al. (2018) and vary based on expansion ratio, kernel, and use of group

convolution. A 'skip' block option is also included to allow a reduction in the network's depth. The method also takes a multi-objective approach by leveraging a loss function that is a combination of both cross-entropy loss and the network's latency loss on a target device, which is calculated by summing the latencies of operators in the network from previously calculated values in a lookup table. FBNets, the family of networks discovered by DNAS, surpass the state-of-the-art models in terms of accuracy and are dramatically smaller and faster than MobileNetV2. FBNets additionally maintain equal accuracy and exceed the accuracy and latency improvements of MNASNet while requiring 420x lower search costs.

Some disadvantages of DNAS are that all network layer candidates must remain in memory as part of the supergraph during the search, and the search space must remain relatively small as the search cost grows linearly with the number of options per layer. A follow-up to DNAS, DMaskingNAS Wan et al. (2020), presents a variant with increased memory and compute efficiency. This solution uses masks that require negligible amounts of memory to represent the channel and input resolution options. In addition, feature maps are reused for all options in the supergraph so that the memory costs remain roughly constant with increasing search spaces. As a result, the search space of DMaskingNAS vastly exceeds that of previous frameworks, including in the search the number of channels, kernel sizes, number of layers, bottleneck types, input resolutions, and expansion rate. In addition, DMaskingNAS expands the search space up to 10^{14} times that of the original DNAS and adds support for search over input resolution and the number of filters. As a result, the searched models, known as FBNetV2s, boast performance exceeding all previous automatically and manually designed models.

Hierarchical NAS methods seek to bridge the gap between cell-based and global search spaces. The key idea behind this category of NAS is for architecture motifs to exist at different levels of hierarchy such that lower-level motifs can be supplied as building blocks of the higher-level motifs. Alternatively, hierarchical NAS might be understood as a NAS approach comprised of an inner search level in which cells or micro features are searched, followed by an outer level in which global features, such as the backbone of the network architecture template, are optimized.

Hierarchical-EAS was one of the first NAS approaches to jointly explore optimizing cells and their arrangement in the complete neural architecture using a hierarchical search space and evolutionary search. The search space was made up of three levels: (1) the set of primitive candidate operations, (2) a directed acyclic graph of possible connections between primitive operations, and (3) the connections between the structures of the second level. The second level of this search space could be considered analogous to the cell-based search space, and the third level to the architecture Liu et al. (2017). Another hierarchical search algorithm, HiNAS Zhang et al. (2019), used a gradient-based approach to design architectures for image denoising. This algorithm could efficiently search for inner cell structures and the whole architecture by stacking the cells at different widths. The inner cell search space was based directly on the continuous relaxation strategy of DARTS. In contrast, a separate search space formed by a super-network of cells with different widths at each layer was used for the outer layer width search. The final architecture was a compact network with only one cell per layer.

Multiple hierarchical NAS models have based their network-level search spaces on the skip-connect heavy backbone structure of Dense Convolutional Networks (DenseNets) Huang et al. (2016). In this feed-forward convolutional architecture, each layer receives the features of all preceding layers as input, while the feature maps output by that layer are passed as input to all subsequent layers. The benefits of DenseNets include avoiding vanishing gradients, encouraging feature reuse, and improving upon the state-of-the-art of many object recognition benchmarks. SuperResolutionNAS Chu et al. (2019) presents a cell-based hierarchical elastic search approach that uses a hybrid evolution and reinforcement learning-based controller to optimize architectures for the super-resolution domain. The algorithm evaluates architectures based on multiple objectives, each derived using incomplete training: (1) a quantitative test set performance, (2) a quantitative measure of the computational cost based on the number of mult-adds, and (3) the number of parameters. The micro search space comprises a variety of candidate operations and parameters such as 2D convolutions, grouped convolutions, inverted bottleneck blocks, repeated blocks, in-cell residual connections, and different kernel sizes and numbers of channels. The macro search space resembles that of DenseNets, and its purpose is to discover the 'backbone' of connections between the

cells. Another publication O'Neill et al. (2021) unveiled an evolutionary NAS method that searches the space of all possible networks between a standard feed-forward convolutional neural network and DenseNet. The estimation and ranking time of the networks are reduced through the use of a low-fidelity performance estimator. This approach can discover networks with greater accuracy than DenseNet and demonstrates that removing some skip connections in DenseNet can be beneficial.

Other hierarchical NAS frameworks base their network-level search spaces on Convolutional Neural Fabrics. Also known as super-network "fabrics" or fabric-like search spaces, these structures embed many potential network architectures. Layers of these networks are locally connected at different scales and channels, and the only hyperparameters are the number of channels and layers. Individual child architectures are sampled as paths, and the weights between paths are shared as in one-shot algorithms. Auto-DeepLab was one example of a hierarchical NAS algorithm that leveraged "fabric-like" spaces Liu et al. (2019a). This framework represented a two-level hierarchical, continuous, and differentiable joint search approach for simultaneously searching cell and network-level architectures for semantic segmentation. The inner cell level search space was consistent with ENAS and its derivatives. The network-level search space was formulated as a simple L-layer trellis that began with a two-layer stem structure that downsampled the spatial resolution by a factor of two at each level of that dimension. This search space was general enough to cover many architecture designs, including U-like networks, completing the search efficiently within a few GPU days, outperforming several state-of-the-art models. Another NAS implementation, block-wisely Self-supervised Neural Architecture Search (BossNAS), introduced HyTra, a fabric-like search space for its self-supervised neural architecture search method. HyTra's search space comprised a 2-dimensional fabric parameterized by the scale of the inputs and depth of the search network. At each fabric node, the algorithm selects between residual bottleneck and vision transformer operations Li et al. (2021a).

In summary, the neural architecture search field was developed to discover neural network architectures for specific tasks automatically. The earliest NAS methods relied on evolution and reinforcement learning-based search strategies. Although these strategies could successfully discover state-of-the-art networks, they lacked efficiency, requiring thousands

of GPU hours. Hence, modular search spaces and gradient-based search strategies that leveraged parameter sharing, such as the DARTS method, were developed. Since gradient-based NAS methods like DARTS require searching a large over-parameterized proxy model with limited correlation to the final network, follow-up implementations such as PDARTS and CDARTS strive to address this discrepancy. Meanwhile, a diverging branch of NAS approaches continues to concentrate on searching within global and hierarchical search spaces that balance high search space flexibility with reasonable compute costs by pursuing alternate methods for reducing computational overhead using methods like multi-objective loss functions and higher performance operations like MBConv blocks. This results in a gap in the literature in which innovations that have improved the performance and efficiency of one branch remain unadopted by the other. Furthermore, these implementations have primarily been developed for convolutional neural networks with traditional photographic image classification tasks in mind.

Chapter 3

ICDARTS

This chapter introduces ICDARTS Herron et al. (2022a), a version of the CDARTS algorithm that has been modified to improve its stability. It also proposes additional experiments, including an ablation study and improvements to its search space. In the original CDARTS, the search and evaluation network weights were trained on separate datasets. Hence, the objective of the joint loss function was to overcome the discrepancies between the continuous search and discrete evaluation networks while ensuring that networks trained on separate datasets performed similarly on a per-example basis. In ICDARTS, the joint training phase of CDARTS is modified to eliminate the evaluation network weights' dependency on those of the search network. This change results in a consistent loss function associated with training the evaluation network across both the search and retraining phases.

Additionally, the objective functions of the search phase networks are modified so that both networks' weights are only optimized on the training dataset. Meanwhile, the search network training criteria are revised so that the update of its weights depends on both the training loss and feature distillation loss, defined as the distance between the logits of the two networks. This change reduces the burden of the feature distillation loss in producing similarity between the outputs of the search and evaluation networks since the search and evaluation network weights are trained on the same dataset. The result of all of these changes is a modified version of CDARTS with improved stability and consistency in performance.

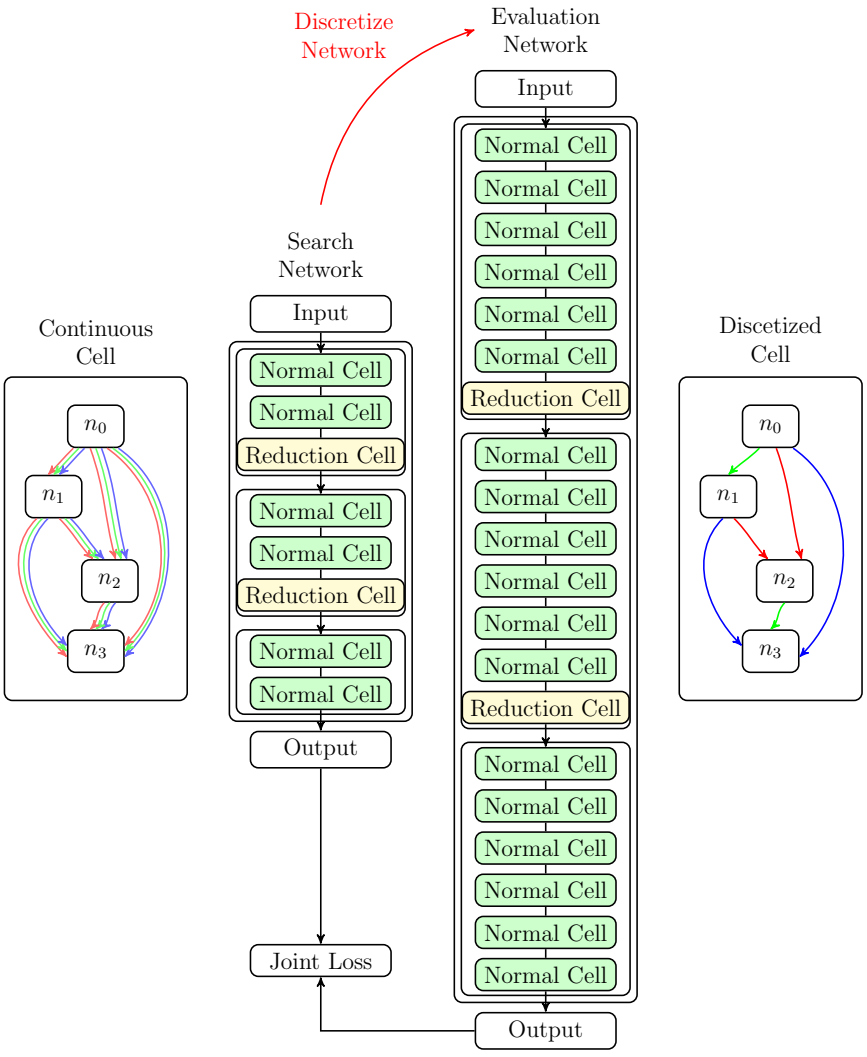


Figure 3.1: Overview of the CDARTS NAS algorithm, including search (left) and evaluation (right) networks with examples of continuous and discretized cells.

3.1 CDARTS

This subsection reviews the CDARTS algorithm presented in (Xu et al., 2020).

3.1.1 CDARTS Architecture

Figure 3.1 overviews the CDARTS algorithm, illustrates the search and evaluation network templates, and shows examples of continuous and discretized cell structures. In both DARTS and CDARTS, the architectures of the search and evaluation networks are composed of cell motifs represented by directed acyclic graphs (DAG), which serve as the building blocks of both networks. As discussed in (Liu et al., 2018a) and (Liu et al., 2019b), each cell graph consists of N nodes, each of which denote some feature representation. Each directed edge i, j within the graph represents a particular operation, $o^{(i,j)}$, that is applied to a node, x_i, to produce x_j. The operation space of DARTS consists of operations with (e.g., convolution) and without (e.g., maximum and average pooling operations, skip connections, and *zero* operations) learned weights. DARTS achieves a continuous search space by weighting the candidate operations at each edge by their corresponding α values:

$$\bar{o}^{(i,j)}(x_i) = \sum_{o \in O} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} \exp(\alpha_{o'}^{(i,j)})} o(x_i) \tag{3.1}$$

These $\alpha^{(i,j)}$ values consist of vectors of size $|O|$ that parameterize the candidate operation strength for each edge in the graph. Each cell accepts inputs from two previous cells, c_{k-2} and c_{k-1}. Each node's output is computed by taking the weighted sum of the outputs of the preceding nodes and outputs of the two previous cells: $x_j = \sum \hat{o}^{(i,j)(x_i)}$. The outputs of each node in a cell are concatenated to produce the final output of the cell. Two types of cells are optimized during the search phase: normal and reduction. The normal cells produce outputs with the exact spatial dimensions as their inputs. In contrast, the reduction cells produce outputs with spatial dimensions that have been reduced by a factor of two by applying a stride of 2 two in its operations (Zoph et al., 2018).

In order to generate the evaluation networks, the continuous search cells are discretized by removing all but the edges with top $k = 2$ alpha values for each node, such that $o^{(i,j)} =$

$\arg\max_{o \in O} \alpha_o^{(i,j)}$. These cells are then copied and stacked to produce a deeper network specified by a template (again, see 3.1) (Zoph et al., 2018; Real et al., 2019; Liu et al., 2018a). In addition to the network structure pictured in 3.1, both networks include an auxiliary head structure, which combines the output of the final normal cell with that of both reduction cells to get the network's final output (Yu and Peng, 2020).

3.1.2 CDARTS Algorithm

As discussed in (Liu et al., 2019b), the DARTS algorithm searches optimal cell structures for the entire evaluation network by using stochastic gradient descent to optimize connection weights within the continuous cells that form the search network. The objective of CDARTS' search phase is to identify the connection weights, α, and search network weights, w_s, that satisfy the following bilevel optimization problem:

$$\begin{aligned} &\min_{\alpha} L_{val}(w_s^*, \alpha) \\ s.t.\ &w_s^*(\alpha) = \arg\min_{w_s} L_{train}(w_s, \alpha) \end{aligned} \tag{3.2}$$

The connection weights, α, are optimized given a validation dataset, but the search network weights, w_s, are learned given a separate training dataset. Afterward, a retraining phase takes place, in which a deeper evaluation network is generated by discretizing the cells discovered in the search phase. The network is retrained from scratch and evaluated on a new test dataset (Liu et al., 2019b; Yu and Peng, 2020)

CDARTS expands upon DARTS by introducing a cyclic feedback mechanism between the search network and an intermediate version of its evaluation network during its search phase. The intermediate evaluation network is generated at the beginning of each iteration of the algorithm and is intended to resemble the final evaluation network of the retraining phase. This innovation allows for the search and evaluation networks to be optimized jointly and enables them to mirror each other in terms of performance and learned features. The following equation represents the joint optimization process, given the search and intermediate evaluation network weights, w_S and w_E and connection weights, α:

$$\begin{aligned} &\min_{\alpha} L_{val}(w_E^*, w_S^*, \alpha) \\ s.t. \quad &w_S^* = \arg\min_{w_S} L_{train}(w_S, \alpha) \\ &w_E^* = \arg\min_{w_E} L_{train}(w_E, \alpha) \end{aligned} \tag{3.3}$$

CDARTS consists of two phases: *pre-training* and *joint learning*, which are repeated cyclically until convergence. In the *pre-training* phase, the weights of the search and intermediate evaluation networks are trained for a limited number of epochs. The search network weights w_S are optimized according to the following equation:

$$w_S^* = \arg\min_{w_S} L_{train}^S(w_S, \alpha) \tag{3.4}$$

where L_{train}^S denotes the cross entropy loss function. The intermediate evaluation network is then generated given the learned α weights by following the same discretization procedure used in DARTS. The intermediate evaluation network weights w_E are finally optimized on the validation set according to the loss function:

$$w_E^* = \arg\min_{w_E} L_{val}^E(w_E, \bar{\alpha}) \tag{3.5}$$

where $\bar{\alpha}$ represents the discretized cell architetures resulting from α.

During the *joint training* phase, the α and w_E weights are updated based on the cross entropy losses of the search and evaluation networks and a soft-target cross-entropy loss, which measures the distance between the logits of the search and evaluation networks. The following equation represents this joint optimization task:

$$\begin{aligned} \alpha^*, w_E^* = \arg\min_{\alpha, w_E} &L_{val}^S(w_S^*, \alpha) + L_{val}^E(w_E, \bar{\alpha}) \\ &+\lambda L_{val}^{S,E}(w_S^*, \alpha, w_E, \bar{\alpha}) \end{aligned} \tag{3.6}$$

By minimizing the $L_{val}^S(w_S^*, \alpha)$ term, the α parameter is optimized given the fixed search network weights w_S^*. The $L_{val}^E(w_E, \bar{\alpha})$ variable serves to optimize the evaluation network weights given $\bar{\alpha}$ as a fixed parameter. Finally, the soft-target cross-entropy loss term, $L_{val}^{S,E}(w_S^*, \alpha, w_E, \bar{\alpha})$ allows for the transfer of knowledge from the evaluation network to

the search network by applying the features learned from the evaluation network to the α parameter of the search network. This term is formulated as:

$$L_{val}^{S,E}(w_S^*, \alpha, w_E, \bar{\alpha}) = \frac{T^2}{N} \sum_{i=1}^{N} (p(w_E, \bar{\alpha}) log(\frac{p(w_E, \bar{\alpha})}{q(w_S^*, \alpha)})) \tag{3.7}$$

where N is the number of training samples, T is the temperature coefficient, and p and q are the feature logits of the search and evaluation networks, respectively. p and q are calculated given the features of the search and evaluation networks f_i^S and f_i^E:

$$\begin{aligned} p(w_E, \bar{\alpha}) &= \frac{\exp(f_i^E/T)}{\sum_j \exp(f_j^E/T)}, \\ q(w_S^*, \alpha) &= \frac{\exp(f_i^S/T)}{\sum_j \exp(f_j^S/T)} \end{aligned} \tag{3.8}$$

The result of this *joint training* phase is sufficient knowledge transfer between the search and evaluation networks that ensures both perform similarly and learn similar features (Yu and Peng, 2020).

3.2 ICDARTS

3.2.1 Algorithm Updates

CDARTS strives to ensure that the continuous search cells optimized within its search network effectively translate to discretized cells within a deeper evaluation network that performs well and learns similar features. However, when reviewing the CDARTS algorithm and its loss functions, multiple issues were identified that limited correlation between the networks optimized in the search and evaluation phases and introduced changes that address each.

It was first noted that the loss function used for updating the intermediate evaluation network during the search phase was formulated so that the network's weights depend on both the search network's loss and the soft-target cross-entropy loss. This loss function contrasts with the loss function used for training the evaluation network during the retraining phase, which depended only on the evaluation network's loss. In order to remove the dependence of

Algorithm 1 ICDARTS Search Phase

Input: Datasets *train* and *val*, search and evaluation iterations S_S and S_E, update iterations S_U, architecture hyperparameter α, and weights w_S and w_E for search S and evaluation E networks
Output: Evaluation network E
Initialize α randomly
Initialize w_S
for each search step $i \in [0, S_S]$ **do**
  if $iModS_U$ **then**
    Discretize α to $\bar{\alpha}$ by selecting the top k
    Generate E with $\bar{\alpha}$
    for each evaluation step $j \in [0, S_E]$ **do**
      Calculate L_{val}^{E}
      Update w_E according to Eq. 3.5
    end for
  end if
  Calculate L_{val}^{S}, L_{val}^{E}, and $L_{val}^{S,E}$
  Update α according to Eq. 3.9
  Calculate L_{train}^{S} and $L_{train}^{S,E}$
  Update w_S according to Eq. 3.10
  Calculate L_{train}^{E}
  Update w_E according to Eq. 3.11
end for

the search phase evaluation network on the additional terms, CDARTS' joint optimization approach is modified so that the weights of the evaluation network are no longer dependent on the loss of the search network but only that of the evaluation network.

Next, the soft-target cross-entropy loss term, $\lambda L_{val}^{S,E}(w_S, \alpha, w_E, \bar{\alpha})$, is shifted to the equation for updating the search network weights. This change allows this term's retention in the joint learning phase and better facilitates knowledge transfer from the evaluation network's weights to those of the search network.

A final issue with the CDARTS algorithm is that it optimizes the search and evaluation networks' weights on two separate datasets. This approach injects unnecessary bias into the joint learning phase, particularly in the case of the soft-target cross-entropy loss term, where CDARTS is attempting to get the two networks to produce the same output even though they are trained on separate datasets. Based on this observation, the function for updating

the evaluation network weights is modified to depend on the evaluation network's loss on the training dataset rather than the validation dataset.

The loss functions for training the α, w_S, and w_E weights are now:

$$\alpha^* = \arg\min_{\alpha} L_{val}^{S}(w_S^*, \alpha) + \lambda L_{val}^{S,E}(w_S^*, \alpha, w_E^*, \bar{\alpha}) \tag{3.9}$$

$$w_S^* = \arg\min_{w_S} L_{train}^{S}(w_S, \alpha) + \lambda L_{train}^{S,E}(w_S, \alpha, w_E, \bar{\alpha}) \tag{3.10}$$

$$w_E^* = \arg\min_{w_E} L_{train}^{E}(w_E, \bar{\alpha}) \tag{3.11}$$

Algorithm 1 presents the algorithm for the ICDARTS search phase and shows how it incorporates the reformulations to the CDARTS loss functions as first introduced in (Herron et al., 2022b). The result of these changes is a search phase evaluation network that is a better proxy for the retraining phase. The search phase evaluation network no longer depends on a loss term that will be unavailable during retraining. Additionally, the soft target cross-entropy loss term is now used for updating the search network's weights so that feedback from the evaluation network is supplied to the search network, which better aligns with the original intended purpose of this term. Finally, training the networks' weights on the same dataset ensures they perform similarly rather than forcing them to produce the same output given two different datasets.

3.2.2 Enabling None Layers in Discretized Networks

The original DARTS method (Liu et al., 2019b) includes *zero* as a layer choice candidate operation. This option allows the NAS method to choose a layer that will produce no output. However, in DARTS and its derivatives, (Chen et al., 2021b; Yu and Peng, 2020), the *zero* layer choice is not allowed when the network candidate operations are discretized as this option is constant and has a gradient of zero. Thus, its corresponding α weights cannot properly ascertain the importance of this operation. In practice, a large α value and small α value would produce the same output for the *zero* layer. Since the corresponding α value

is unlearnable, it is unclear why this operation choice is included during the search phase. In the following section, experimental results demonstrate comparable performance without including this operation during the search phase. Also explored is an alternative formulation of this operation that outputs randomly generated activations during search and is replaced with a no-op when the evaluation network is generated. The reformulation allows the alpha weight corresponding to the *zero* operation to be learned without contributing to extracting features from previous layers. This approach assumes that if a layer contributes less than random noise, it should not be included in the final evaluation network.

Five different experiments, each using a different configuration of *zero* operation return types, were conducted to assess the effectiveness of ICDARTS both with and without the *zero* operation choice. These configurations are all listed in Table 3.1. The V0 configuration is the same as the one used in the original CDARTS paper, in which the *zero* option is only included in the search network but ignored at discretization so that the deep evaluation networks and the final network used in retraining have no *zero* layers. However, configurations $V1 - V4$ use the same number of layer options for the continuous search network and the discrete evaluation network. The traditional *zero* option returns a tensor of zeroes the size and shape of the input, and the *random* option similarly returns a tensor of uniform random values for each input. In addition to the experiments run using the CDARTS and ICDARTS search procedures, a separate set of experiments is run in which network cell motifs are randomly initialized eight times for each *zero* operation return protocol. These experiments aim to provide baseline performance results to contrast those of architectures discovered by both algorithms.

3.3 Ablation Studies

After incorporating the initial set of improvements to the CDARTS algorithm, two ablation studies were carried out on the ICDARTS search space and algorithm. The first was on the search space template of ICDARTS, including its operation choices, auxiliary heads, stemming layers, and reduce cells. Table 3.5 contains a comprehensive listing of each ablation

and any operations used to replace the ablation. Note that the none operation is left out of the set of operation choices by default.

The second ablation study was of the algorithmic changes to the CDARTS algorithm that resulted in ICDARTS. This study considered two combinations of incremental improvements to the original algorithm. The details of both are listed in Table 3.7.

3.4 Experiments & Results

3.4.1 Datasets

For each change to the ICDARTS algorithm and search space, the architecture search phase is conducted on the CIFAR-10 dataset, and the resulting network architecture is retrained on CIFAR-10. The networks produced before and after the initial algorithmic changes that resulted in the ICDARTS algorithm and the experiments on the *zero* operation were additionally retrained and evaluated on the CIFAR-100 dataset. The CIFAR-10 and CIFAR-100 image classification benchmarks consist of 10 and 100 classes, respectively, and both are comprised of $50K$ training and $10K$ testing images of resolution 32×32.

3.4.2 Search & Evaluation Settings

At the beginning of the search phase, the original training set is divided into two datasets of equal size, denoted by *train* and *val*, as in (Liu et al., 2019b) and (Yu and Peng, 2020). As previously discussed, in the ICDARTS algorithm, the *train* partition is used for updating the weights of both the search and evaluation networks. At the same time, *val* is reserved for updating the α weights. The search phase runs for 30 epochs, not including two epochs for pre-training the search network and 1 for warming up the intermediate evaluation network each time it is generated. The search and evaluation weights, w_S and w_E are updated using separate SGD optimizers with learning rates of 0.08, decay rates of 3×10^{-4}, and momentum settings of 0.9. The α weights are updated using an Adam optimizer (Kingma and Ba, 2015) with a learning rate of 3×10^{-4}, decay rate of 0, and momentum βs of $0.5, 0.999$.

Table 3.1: ICDARTS 'Zero' Operation Return Type Configurations

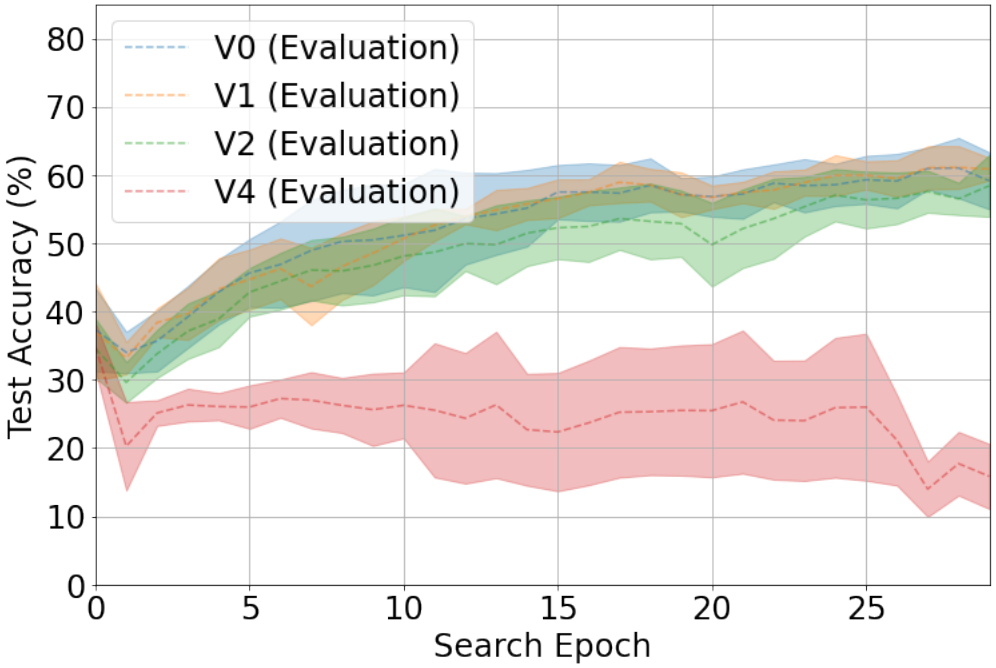
Zero Return Configuration	**Zero Return Type**		
	Search	***Evaluation***	***Retraining***
V0	zero	not included	not included
V1	not included	not included	not included
V2	random	random	random
V3	random	random	zero
V4	random	zero	zero

Table 3.2: CIFAR-10 Retraining Test Set Accuracies of Networks Produced by CDARTS, ICDARTS, and Random Initialization using Different *Zero* Return Configurations (see Table 3.1).

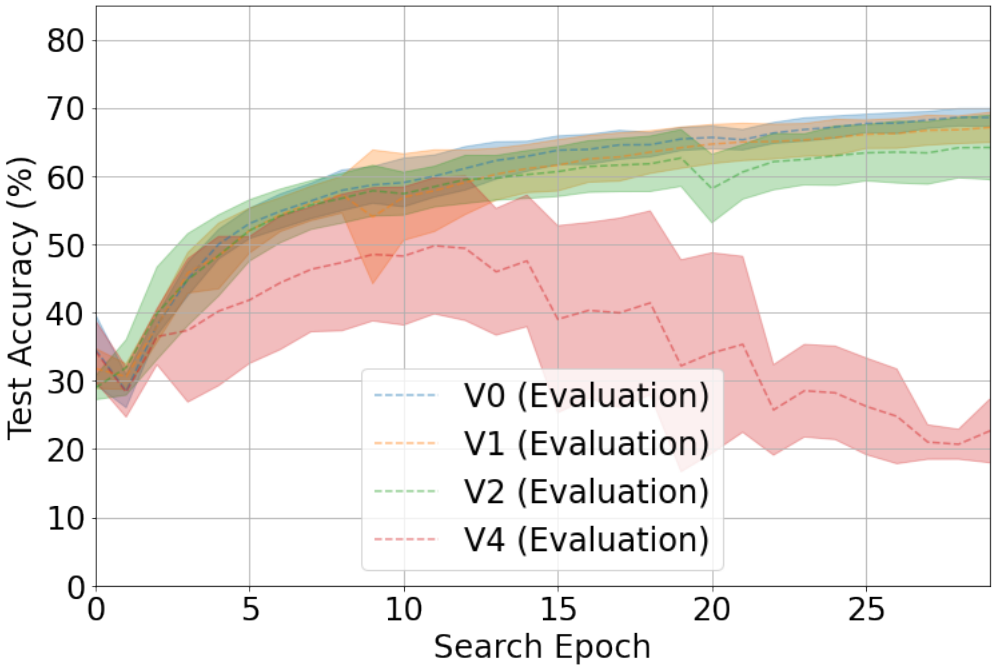
Zero Return Configuration	**Algorithm**		
	Random	***CDARTS***	***ICDARTS***
V0	96.78 (0.39)	97.17 (0.21)	96.99 (**0.16**)
V1	96.57 (0.32)	97.01 (0.24)	97.10 (**0.19**)
V2	96.52 (0.37)	96.97 (0.27)	96.92 (**0.15**)
V3	96.58 (0.31)	96.96 (0.28)	96.93 (**0.14**)
V4	96.58 (0.31)	23.68 (29.71)	90.00 (**12.72**)

Table 3.3: CIFAR-100 Retraining Test Set Accuracies of Networks Produced by CDARTS, ICDARTS, and Random Initialization using Different *Zero* Return Configurations (see Table 3.1).

Zero Return Configuration	**Algorithm**		
	Random	***CDARTS***	***ICDARTS***
V0	81.11 (1.39)	83.66 (**0.29**)	83.10 (0.65)
V1	80.54 (1.05)	83.13 (0.71)	83.33 (**0.19**)
V2	80.63 (1.26)	83.19 (0.68)	83.19 (**0.21**)
V3	80.84 (1.13)	83.05 (0.72)	83.15 (**0.36**)
V4	80.84 (1.13)	10.98 (**25.91**)	63.13 (28.35)

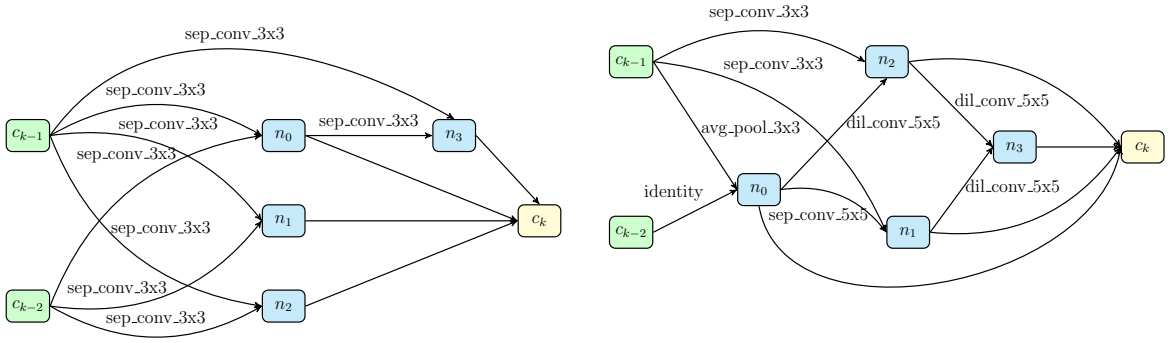


(a) CDARTS Evaluation Accuracies



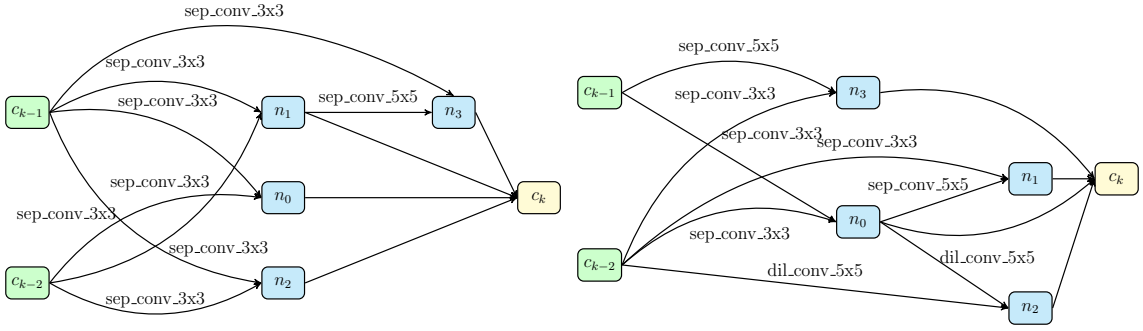
(b) ICDARTS Evaluation Accuracies

Figure 3.2: CDARTS and ICDARTS Search Phase Evaluation Network CIFAR-10 Test Accuracy Curves Given Different *Zero* Operation Configurations (see Table 3.1).



(a) Normal Cell **(b)** Reduction Cell

Figure 3.3: CDARTS V0 normal and reduction cells from the network with the best overall test accuracy.



(a) Normal Cell **(b)** Reduction Cell

Figure 3.4: ICDARTS V1 normal and reduction cells from the network with the best overall test accuracy.

The evaluation stage involves retraining the discovered architectures for 600 epochs on the entire training dataset and evaluating the retrained network on the test dataset. The retraining procedure closely resembles that of the original CDARTS paper. The batch size is set to 128, and an SGD optimizer is employed with a learning rate of 0.025, momentum of 0.9, and weight decay of 5×10^{-4}. As in the search phase, this optimizer is paired with a cosine annealing learning rate scheduler. Following the approach of (Pham et al., 2018; Zoph et al., 2018; Liu et al., 2018a), the training dataset is augmented with a cutout regulation length of 16 (DeVries and Taylor, 2017), the drop path rate of the evaluation network is set to 0.3, and the auxiliary towers to 0.4. The search and evaluation phases of each experiment discussed in this publication were each run eight times unless indicated otherwise in their result tables.

The curves shown in Figure 3.2 depict the test set accuracies of the evaluation networks throughout the search phases of CDARTS and ICDARTS. Each curve has been plotted with a 95% confidence interval across the runs of each configuration. Search network pre-training epochs are not included in these graphs. The lower standard deviation of the revised algorithm is evidence that the revised approach gives results with improved consistency and stability across different initializations. The accuracy disparity between the search and retraining phases is expected due to the differences in data augmentation and the number of epochs trained.

The evaluation network curve for the V4 search space configuration, which includes the *zero* operation in the search phase evaluation network but not the search network, shows inferior stability and difficulty learning. This outcome is likely the product of the difference in behavior between the search and evaluation networks during the search process, which results in a feedback loop that produces an increasing number of *zero* operations in the discretized evaluation network.

Tables 3.2 and 3.3 list the average and standard deviation test set accuracies of the evaluation networks on CIFAR-10 and CIFAR-100. Note that the reported values were obtained at the end of the retraining cycle rather than the best test performance obtained at any point during retraining (as was reported in (Liu et al., 2019b) and later (Yu and Peng, 2020)). Hence, the presented results measure a typical run's performance rather

than report the performance of the best outlier. The accuracy results of the architecture produced randomly and by the CDARTS and ICDARTS algorithms show that the ICDARTS networks achieved similar mean accuracies to that of the CDARTS network but with much smaller variation in performance, demonstrating stability improvements and results that are potentially more reproducible.

3.4.3 Ablation Studies

Table 3.5 and Figures 3.5 and 3.6 show the results of the ablation study of the ICDARTS template. Note that the inference latencies listed in Table 3.5 were obtained by calculating the average per batch inference times on the test dataset before retraining. Additionally, the operation type and cell depth frequencies shown in Figures 3.5 and 3.6 are represented as the totals in each evaluation network in order to account for the difference in the number of normal and reduce cells in each network. For the cell operation choice ablations, only the ablation of the pooling operations improved the average retraining accuracy, likely because this resulted in less competition with better-performing operations and the selection of more identity operations. The retraining accuracies for the rest of the ablations fell below that of the original, with the ablation of the separable convolutions yielding the lowest accuracies for this category of ablation.

On the other hand, each cell operation choice ablation improved average latencies, except for the dilated convolution ablation. This trend likely occurred because removing this option caused the algorithm to favor the computationally expensive separable convolutions. The ablation of the separable convolution operations improved the retraining latencies by the most significant margin, although it also produced the worst average retraining accuracy.

The ablation of the auxiliary heads also resulted in lower retraining accuracies. However, this ablation has higher latencies, a finding that might be explained by this ablation resulting in ICDARTS favoring a deeper cell structure.

The stemming layer ablations also offered no improvement in retraining accuracies and worse inference latencies. The only exception was when the layer was replaced with an identity operation so that the number of input channels was equal to that of the input images, which resulted in a network with fewer parameters and, hence, a lower latency.

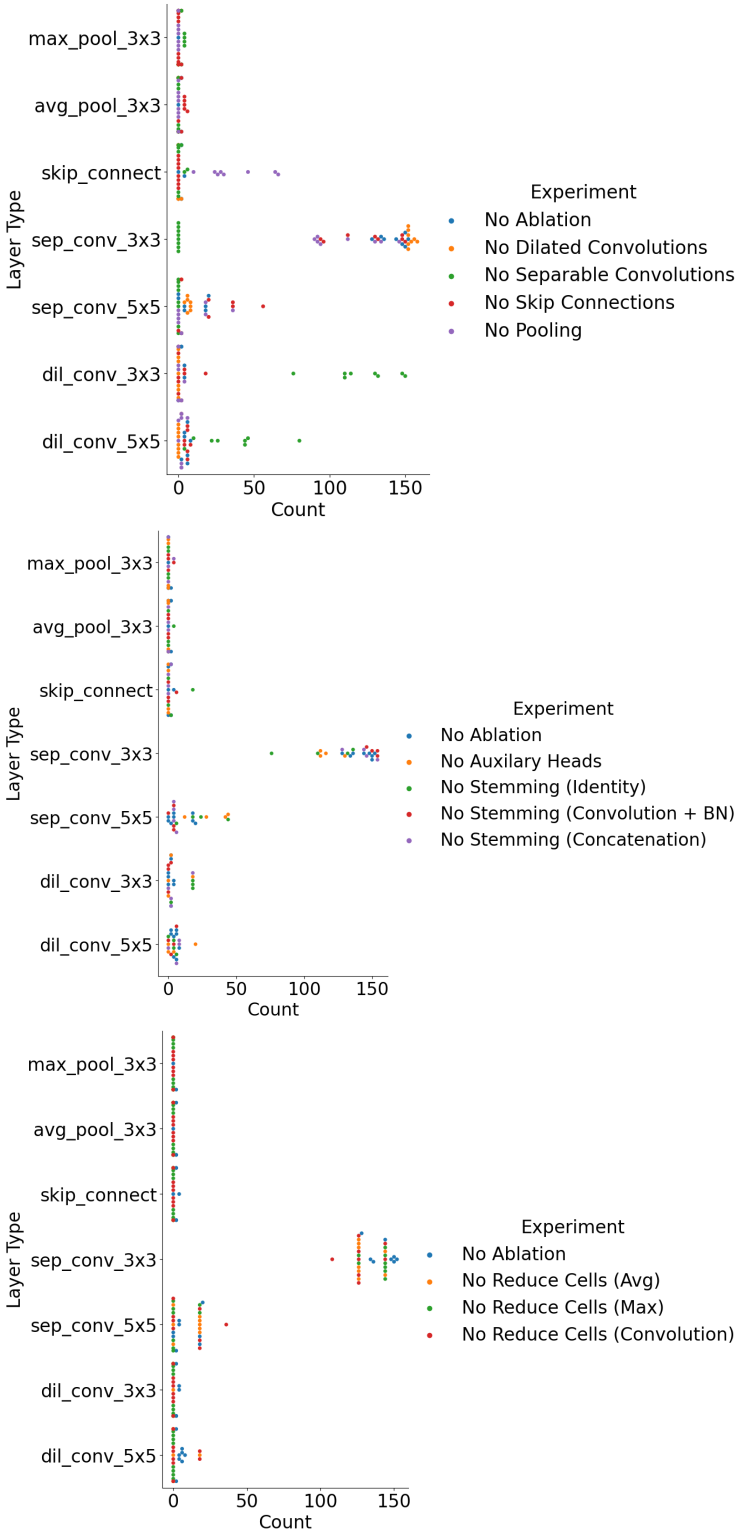


Figure 3.5: ICDARTS Template Ablation Per Network Layer Type Frequencies

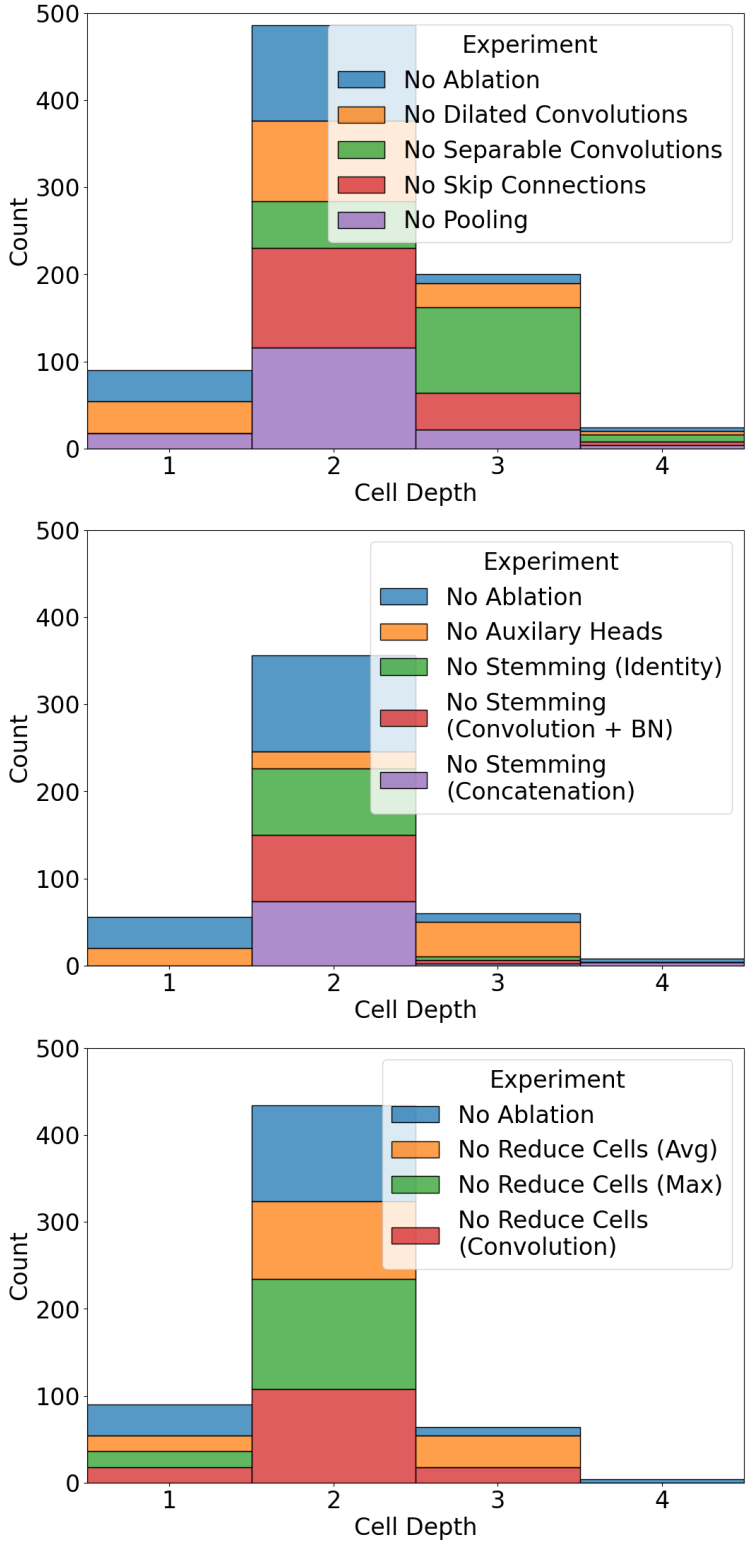


Figure 3.6: ICDARTS Template Ablation Per Network Cell Depth Frequencies

Finally, the accuracies from the reduce cell ablations were also worse than that of ICDARTS. On the other hand, the average inference latencies varied, with the case in which reduce cells were replaced with convolutions, with stride two providing the only latency improvement.

The results of the algorithmic ablation study listed in Table 3.7 generally demonstrate stability improvements with the addition of each modification to the original algorithm on both routes. The inference latencies tend to slow with each change. The exception to this pattern was route A, in which the switch to updating both network weights on the training set was not made until the final step. However, as shown in the tables, the slower latencies were eventually rectified with the final modification to the algorithm on both routes.

Table 3.4: CIFAR-10 Retraining Test Set Accuracies

Ablation	Retraining Accuracy	Inference Latency (batch/s)
CDARTS (No Zero)	97.13 (0.14)	0.10 (0.01)
Pooling	**97.19 (0.23)**	0.09 (0.02)
Identity	96.96 (0.35)	0.11 (0.02)
Dilated Convolutions	97.02 (0.26)	0.10 (0.01)
Separable Convolutions	96.15 (0.27)	**0.07 (0.00)**
Auxiliary Heads *	96.86 (0.14)	0.11 (0.00)
Stemming (Identity) *	80.01 (2.18)	0.09 (0.02)
Stemming (Convolution + Batch Norm) *	96.81 (0.38)	0.11 (0.03)
Stemming (Concat) *	92.40 (0.33)	0.16 (0.00)
Reduce (Avg Pooling)	96.94 (0.19)	0.12 (0.07)
Reduce (Max Pooling)	96.87 (0.06)	0.10 (0.03)
Reduce (Convolution)	96.77 (0.12)	0.09 (0.01)

Table 3.5: ICDARTS Template Ablations and Replacements

Ablation	**Replacement**
Pooling Operations	-
Identity Operation	-
Dilated Convolution Operations	-
Separable Convolution Operations	-
Auxiliary Heads	-
Stemming Layer	Identity
	Convolution (k=1), Batch Norm
	Concatenate Inputs
Reduce Cells	Average Pooling
	Max Pooling
	Convolution (k=1, stride=2)

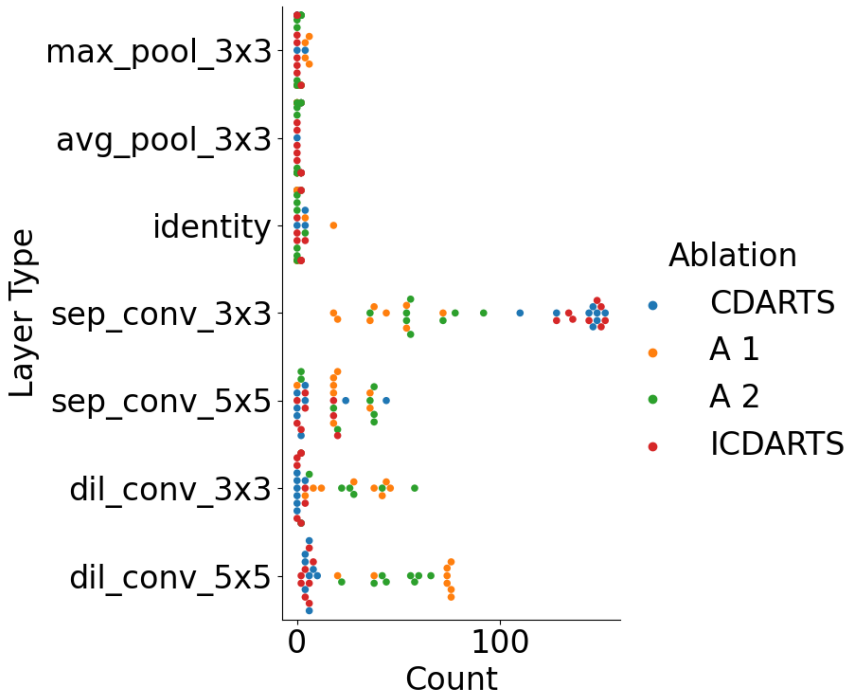
Table 3.6: ICDARTS Algorithmic Ablation Study Routes with Loss Function Changes

Algorithm	Loss Function	
	Route A	Route B
CDARTS	$\alpha \rightarrow L_{val}^{S} + L_{val}^{S,E}$ $w_E \rightarrow L_{val}^{E} + L_{val}^{S,E}$ $w_S \rightarrow L_{trn}^{S}$	
Ablation 1	$\alpha \rightarrow$ *No Change* $w_E \rightarrow$ *No Change* $w_S \rightarrow L_{trn}^{S} + L_{trn}^{S,E}$	$\alpha \rightarrow$ *No Change* $w_E \rightarrow L_{trn}^{E} + L_{trn}^{S,E}$ $w_S \rightarrow$ *No Change*
Ablation 2	$\alpha \rightarrow$ *No Change* $w_E \rightarrow L_{val}^{E}$ $w_S \rightarrow$ *No Change*	$\alpha \rightarrow$ *No Change* $w_E \rightarrow L_{trn}^{E}$ $w_S \rightarrow$ *No Change*
ICDARTS	$\alpha \rightarrow$ *No Change* $w_E \rightarrow L_{trn}^{S,E}$ $w_S \rightarrow$ *No Change*	$\alpha \rightarrow$ *No Change* $w_E \rightarrow$ *No Change* $w_S \rightarrow L_{trn}^{S} + L_{trn}^{S,E}$

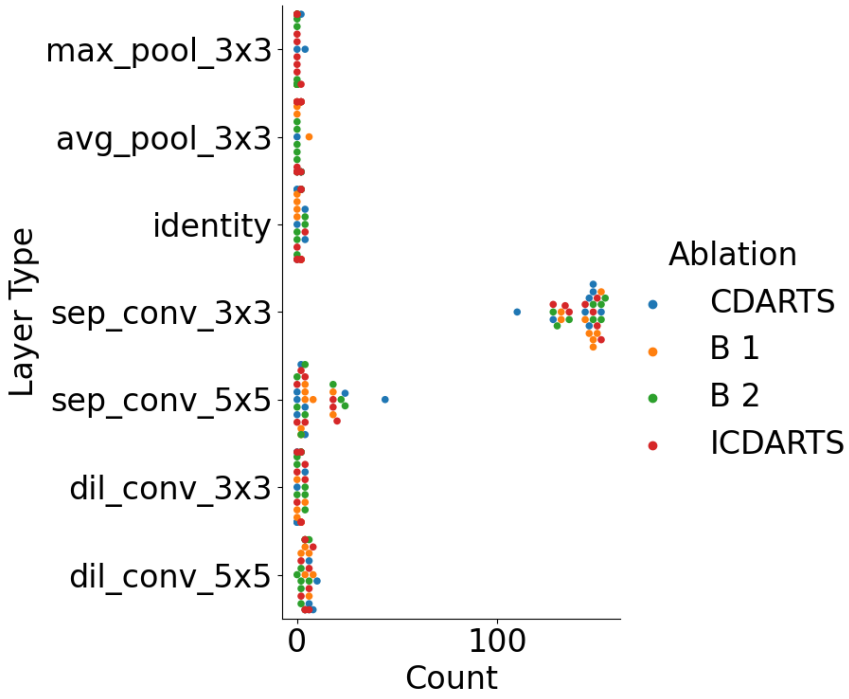
Note: *No Change* indicates no change from the loss functions in the previous row.

Table 3.7: ICDARTS Algorithmic Ablation CIFAR-10 Retraining Test Set Accuracies

Ablation	**Retraining Accuracy**	**Inference Latency (batch/s)**
CDARTS (No None)	96.94 (0.30)	0.09 (0.01)
A1	96.35 (0.48)	**0.08 (0.00)**
B1	96.99 (0.22)	0.09 (0.01)
A2	96.69 (0.22)	**0.08 (0.00)**
B2	97.05 (0.18)	0.11 (0.00)
ICDARTS	**97.13 (0.14)**	0.10 (0.01)

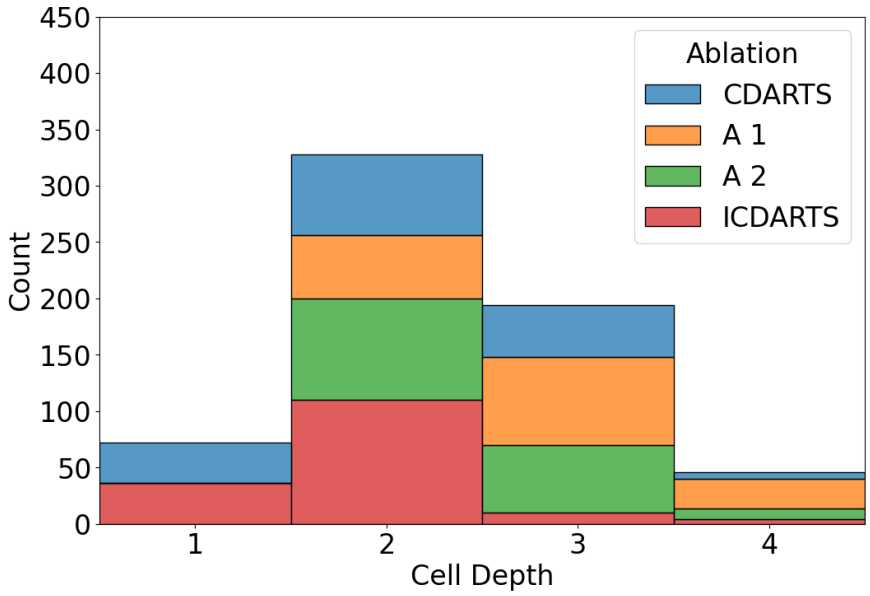


(a) Route A

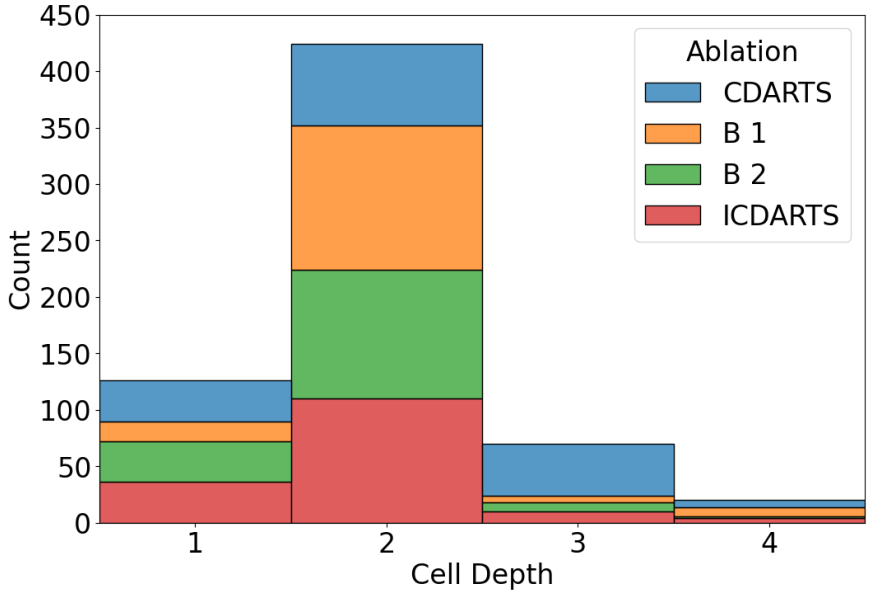


(b) Route B

Figure 3.7: ICDARTS Per Network Layer Type Frequencies for Algorithmic Ablation Study Routes A and B. The details of each route are listed in Table 3.6.



(a) Route A



(b) Route B

Figure 3.8: ICDARTS Per Network Cell Depth Frequencies for Algorithmic Ablation Study Routes A and B. The details of each route are found in Table 3.6.

Chapter 4

Expanding the Search Space of ICDARTS

Upon completing the ablation study, several methods for expanding the limited search space of ICDARTS were explored. This process involved experimenting with alternative operation sets with varying complexities and cell discretization approaches that expanded the space of architectures that could be discovered during the search phase. The work presented in this section further culminated in a novel tournament-based approach for incorporating dynamic search spaces in ICDARTS and a multiobjective version of ICDARTS that produces networks that effectively balance predictive performance with lower compute costs.

4.1 Alternate Operation Spaces

Alternative operation spaces for ICDARTS were first explored by curating three additional operation search spaces for ICDARTS. Each operation space comprised various complexities, ranging from the most basic operations to mobile convolution blocks from current state-of-the-art vision models. Each operation space and a brief description are listed in Table 4.1. Note that pooling and identity operations were included in each of the four operation spaces, as they are all simple operations yet have appeared alongside more complex operations, such as in operation space 3. The minimum convolution operation consists of a convolution

Table 4.1: ICDARTS Operation Spaces

Search Space	Category	Description	Operation	Parameters
1	Basic Operations	Basic building blocks of more complex operations.	Convolution	$k = 3, 5$
			Depthwise Convolution	$k = 3, 5$
			ReLU	
			Leaky ReLU	
			Sigmoid	
			Tanh	
			BatchNorm	
2	Simple Operations	Operations from early NAS literature including NASNet Zoph et al. (2017).	Minimum Convolution	$k = 3, 5$
			Standard Convolution	$k = 3, 5$
			$N \times 1$ Convolution	$k = 7, 9$
3	DARTS Operations	Widely-adopted search space used by DARTS and its derivatives.	Identity*	
			Max Pooling*	$k = 3$
			Average Pooling*	$k = 3$
			Separable Convolution	$k = 3, 5$
			Dilated Convolution	$k = 3, 5$
4	MBConv Blocks	State-of-the-art mobile convolution blocks.	MBConv Sandler et al. (2018)	$k = 3$
			MBConv Tan and Le (2019)	$k = 3$; $g = 1$
			Fused-MBConv Tan and Le (2021)	$k = 3$; $g = 1$

Note: Operations marked with * are included in all search spaces.

Table 4.2: Dynamic Search Combined Operation Space

Operation	**Parameters**	**Operation**	**Parameters**
ReLU	-	Leaky ReLU	-
Sigmoid	-	Tanh	-
Batch Norm	-	Identity	-
Convolution	$k = 3, 5$	Max Pooling	$k = 3, 5$
Avg Pooling	$k = 3, 5$	Minimum Convolution	$k = 3, 5$
Standard Convolution	$k = 3, 5$	$N \times 1$ Convolution	$k = 7, 9$
Separable Convolution	$k = 3, 5$	Dilated Convolution	$k = 3, 5$
MBConv	$k = 3, 5$	MBConvV2	$k = 3, 5$ $g = 1, 4, 6$
Fused MBConv	$k = 3$ $g = 1, 4, 6$		

Algorithm 2 ICDARTS Dynamic Search Algorithm

Input: Number of Tiers T, Pool of Layer Operations O, Maximum Operations per Set of Edges O_{max}
Output: Optimal Operations, o for Each Set of Edges
for each $t \in T, ...1$ **do**
  for each $r \in 1, ..., 2t$ **do**
    if $t = 3$ **then**
      Randomly select O_{max} layer options $o_{t,r}$ for each set of edges given O
    else
      Set $o_{t,r}$ to $o_{t-1,2r} + o_{t-1,2r+1}$ for each set of edges
    end if
    Run ICDARTS to optimize $o_{t,r}$ for each set of edges based on learned α values
    if $t = 1$ **then**
      Return $o_{t,r}$ for each set of edges
    else
      Update $o_{t,r}$ to top 50% of $o_{t,r}$ by α value for each set of edges
    end if
  end for
end for

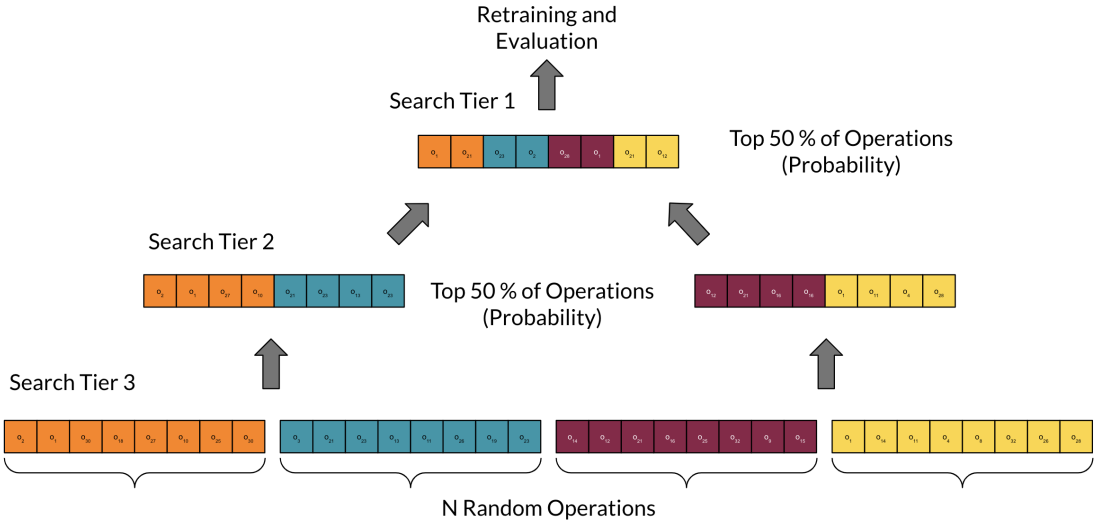


Figure 4.1: Tournament-Style Dynamic Search Space Algorithm for ICDARTS Overview with Tiers

followed by a batch norm and the standard convolution operation is defined by a ReLU activation function followed by a convolution and batch norm.

The results from running ICDARTS with different operation spaces are shown in Table 4.1 and Figures 4.2 and 4.3. Operation space 2 achieved the highest average retraining accuracy despite using simpler operations than the default operation space 3. However, the accuracies of the networks produced using operation space 2 varied more than those made using the original operation space. This outcome might be explained either by ICDARTS' tendency to favor slightly deeper cell architectures with this operation space than with operation space three or by this operation space's diverse yet competitive selection of operation choices that appears to balance operation complexity and performance, resulting in the search algorithm preferring cells composed of a diverse selection of simpler operations. By contrast, the cells produced by the other operation spaces tended to favor one operation choice above all others. The networks created with operation space 1 selected the convolution with kernel size five operation and slightly deeper cell architectures than those produced by the original operation space. However, these architectures yielded the worst accuracies of any operation space. Operation space 4, which used the most complex operations and favored the MBConvV1 operation, produced the second lowest accuracies. However, the poor accuracies and inference latencies of the network discovered using this operation space could be explained by the high computational demands of its operations. This limitation resulted in fewer operation choices being included in its set of operation choices. Furthermore, the networks produced using this set of complex operations may have benefited from additional search and retraining time.

After evaluating the efficacy of ICDARTS on different operation spaces, the operations of all four operation spaces and additional operations left out of the previous operation spaces due to memory constraints were combined into one comprehensive operation space (see Table 4.2). A novel, tournament-style algorithm for incorporating dynamic operation spaces into ICDARTS was implemented to ensure that ICDARTS could efficiently traverse this ample operation space. This algorithm was partially inspired by other dynamic operation space methods, including (Li et al., 2021b) and (Shaw et al., 2019). As shown in Algorithm 2 and Figure 4.1, the algorithm begins by randomly selecting O_{max} operations from the set of all operations, O, for each set of cell edges. Four independent games, or runs of ICDARTS'

Table 4.3: ICDARTS Expanded Search CIFAR-10 Retraining Test Set Accuracies and Inference Latencies

Operation Space	Retraining Accuracy	Inference Latency (s/batch)
1*	95.08 (0.39)	0.11 (0.02)
2*	**97.29 (0.29)**	0.10 (0.02)
3	97.13 (0.14)	0.10 (0.01)
4*	96.55 (0.21)	0.21 (0.01)
Dynamic*	**97.25 (0.06)**	0.10 (0.01)

Note: Results from 6 runs are indicated by *.

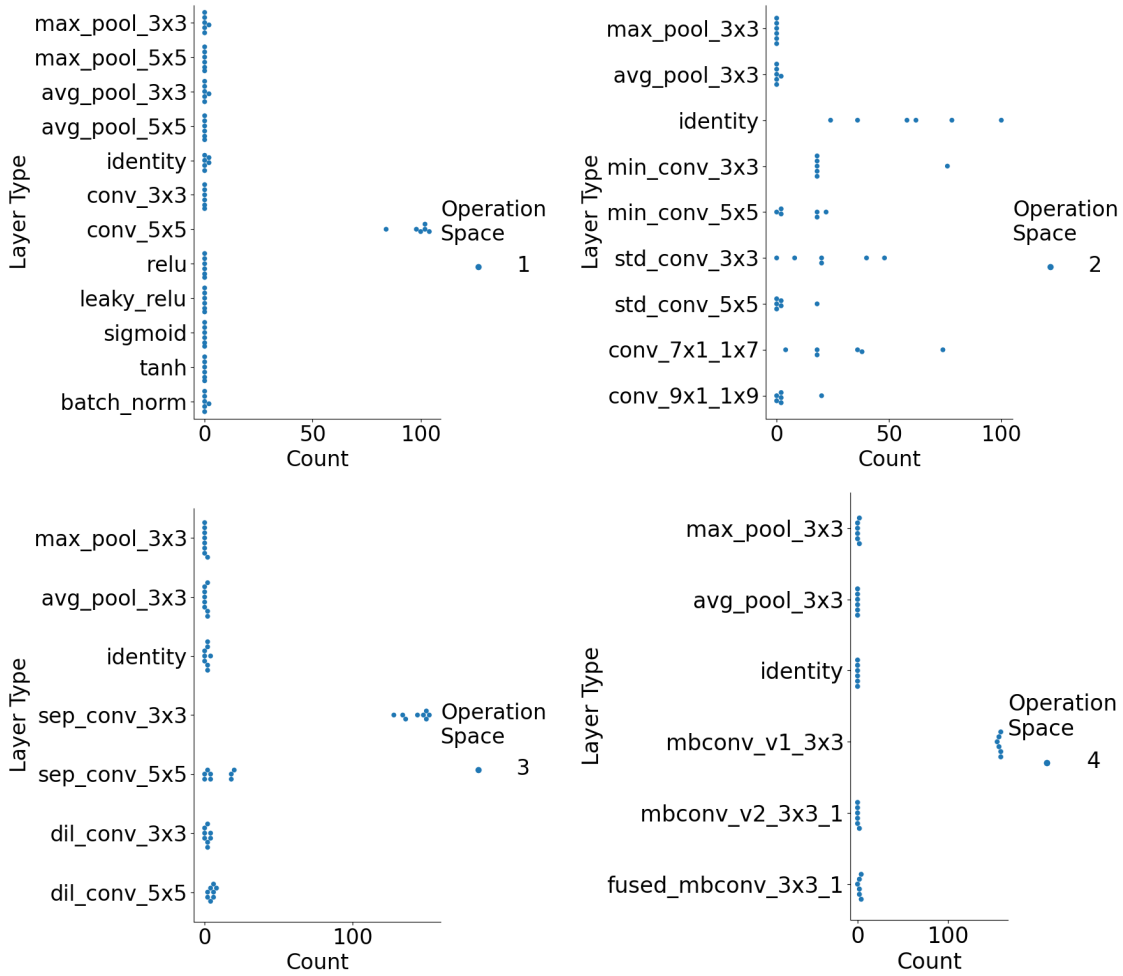


Figure 4.2: ICDARTS Alternative Operation Spaces Per Network Layer Type Frequencies. The details of each operation space are listed in Table 4.1.

search phase, then search networks given operation spaces formed by randomly selecting from the master set of operations listed in 4.2. Upon completion of the initial tier of the search phase, the operations corresponding to the top 50% of alpha values are retained. The algorithm then proceeds to the next tier, forming the per-node operation sets of the two games in this tier by combining the top operations from the first and second and third and fourth games of the previous tier and running the ICDARTS search phase from the beginning on both operation sets. The process repeats through the topmost tier, producing the final evaluation network evaluated in the retraining phase.

The average accuracy and inference latency of ICDARTS using the dynamic operation space algorithm are listed alongside those of the other operation spaces in Table 4.3. Additional results from each tier of the search algorithm are displayed in Figures 4.4 and 4.5. The tournament-style dynamic operation space algorithm applied to the master operation space of 4.2 achieved an accuracy second only to that of operation space two but with one of lowest standard deviations of any experiment performed on ICDARTS and an inference latency on par with that of ICDARTS given the second and third operation spaces. As shown in Figure 4.4, the cells produced by the final tier of the algorithm most favored the tanh, MBConv, and simple, standard, and separable convolution operations. All of these operations were among the top in their derivative operation spaces, except the tanh activation and the standard convolutions, possibly due to the ability of these operations to pair well with others. Figure 4.5 reveals that deeper cell architectures were favored in the lower tiers, in which the cell operations varied most since they were randomly selected for each cell at this level. However, as the architectures converged towards selecting the most optimal layer operations in the higher tiers, shallower architectures were favored. This pattern suggests that, when given a large and diverse operation space of candidate operations, the effectiveness of layer choices may be more important than cell depth for designing an optimal network architecture. This outcome may also explain how this operation space produced networks with relatively low inference latencies and suggests that this algorithm may prove helpful for discovering networks with both low inference latencies and high generalization abilities.

4.2 Alternate Search Cells

After evaluating alternative operation spaces for ICDARTS, methods for further expanding ICDARTS' search space by expanding the capacity of its search cells were considered.

To explore expanding the capacity of search cells, ICDARTS run with its number of nodes per cell decreased to 3 and increased to 5. The results are displayed in Table 4.4 and Figures 4.6 and 4.6. As was expected, ICDARTS networks that used three nodes per cell underperformed compared to those produced by the original configuration of ICDARTS. Networks with five nodes per cell also performed worse when retrained for the standard 600 epochs and comparatively but with more variation in results when trained for an additional 150 epochs.

Since simply increasing the number of nodes per cell failed to produce improved results, alternative methods for expanding the capacity of ICDARTS' cell search space were explored, specifically, alternative methods for discretizing search cells that increased the number and diversity of networks in its search space. As previously discussed, the search cells optimized in CDARTS and ICDARTS were initially introduced in DARTS. In these cells, a softmax operation is applied only to edges originating at the same node. Of these edges, only the one with the highest softmax values can be selected as an input to a cell (see Figure 4.8a). Allowing a maximum of one output of a previous node to be input to a current node eliminates any potential edge candidates from the same cell that may be more suitable than those from a different cell, thereby limiting the number of cell structures the search algorithm can discover. Based on this observation, Jiang et al. (2019) presented the I-DARTS approach, in which all incoming edges to a node are compared equally by applying one softmax operation across all their weights. k edges are then selected from among these with equal weight, regardless of whether or not they originate from different preceding nodes (see Figure 4.8b). The advantages of this approach are that it increases the amount of cell architectures that can be discovered and allows all incoming edges to be compared fairly. After implementing a version of the I-DARTS cells and incorporating them into ICDARTS, a novel variant of this discretization approach, XDARTS, was also implemented to expand the search space of the cells further and to compensate for any noise or instability caused

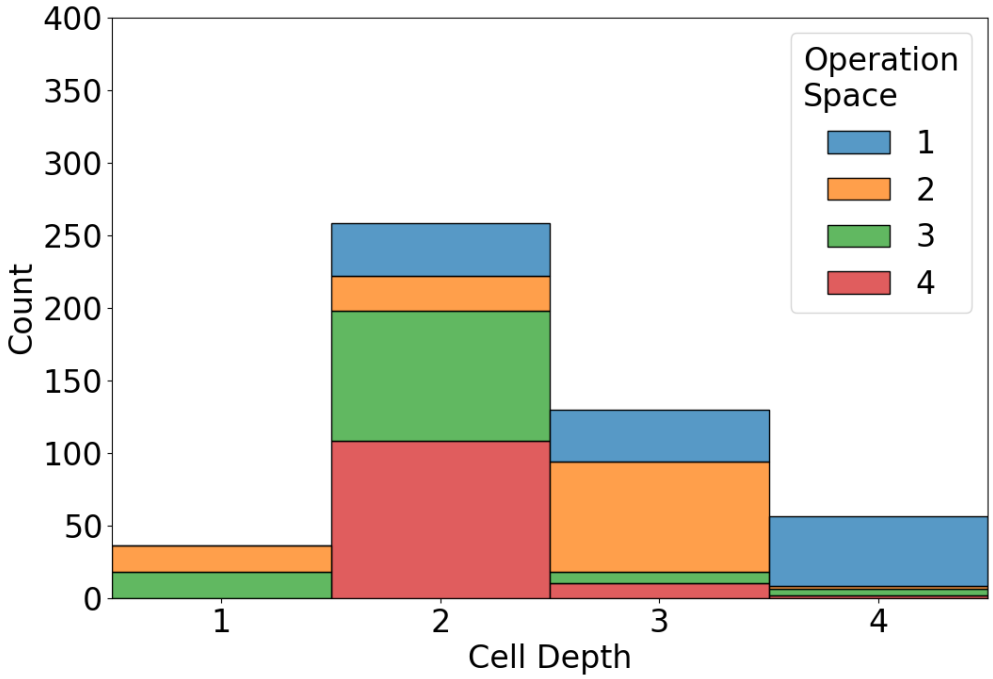


Figure 4.3: ICDARTS Alternative Operation Spaces Per Network Cell Depth Frequencies The details of each operation space are listed in Table 4.1.

Table 4.4: ICDARTS with Varying Numbers of Cells per Node CIFAR-10 Retraining Test Set Accuracies and Inference Latencies

Nodes Per Cell	Retraining Accuracy	Inference Latency (s/batch)
3	96.91 (0.26)	0.09 (0.01)
4	**97.13 (0.14)**	0.10 (0.01)
5	97.05 (0.24)	0.11 (0.02)
5*	97.12 (0.27)	0.11 (0.02)

Note: * denotes networks that were retrained for 750 epochs.

Figure 4.4: ICDARTS Dynamic Operation Space Algorithm Tiers Per Network Layer Type Frequencies

by selecting from a much larger and linearly increasing set of edges at each depth. The number of which at each depth equals the number of preceding nodes times the number of operations, rather than remaining constant at the number of candidates in its operation set, as was the protocol with the DARTS cells. With this discretization approach, the number of edges selected as inputs to a node equals the number of nodes preceding it. Thus, the number of edges selected at each node grows alongside its number of prospective input edges (see Figure 4.8c).

Table 4.5 and Figure 4.9 show that the I-DARTS cells produced results significantly worse than the I-DARTS cells in terms of accuracy and stability, while the opposite was the case with the XDARTS cells even though the depths and operation choices of the searched DARTS and I-DARTS cells are similar, with the I-DARTS cells even tending to be slightly deeper than the DARTS cells. The results from the XDARTS cells demonstrate that the stability and performance reductions resulting from applying softmax functions over increasing numbers of edges in the I-DARTS cells can be rectified by allowing the selection of additional edges based on a node's depth in its respective cell.

4.3 Multi-Objective ICDARTS

Although the networks produced using XDARTS were significantly more stable and accurate than those produced with the original discretization method, they also had high retraining latencies. A multiobjective version of ICDARTS was developed to address this problem by encouraging the algorithm to prefer networks that balance accuracy with low compute costs.

Taking inspiration from Tan et al. (2019) and Wu et al. (2018), the multiobjective version of ICDARTS combines the original multiobjective loss function with a penalty term. A novel algorithm for estimating the latency of search networks was devised to calculate the penalty term. The algorithm requires the architecture weights of each cell type, α, and a lookup table of expected latency weights associated with each operation ICDARTS' original search space. The algorithm was designed to be used alongside the I-DARTS and XDARTS discretization protocols due to their simplicity and equal weighting of all incoming edges to a node for selection.

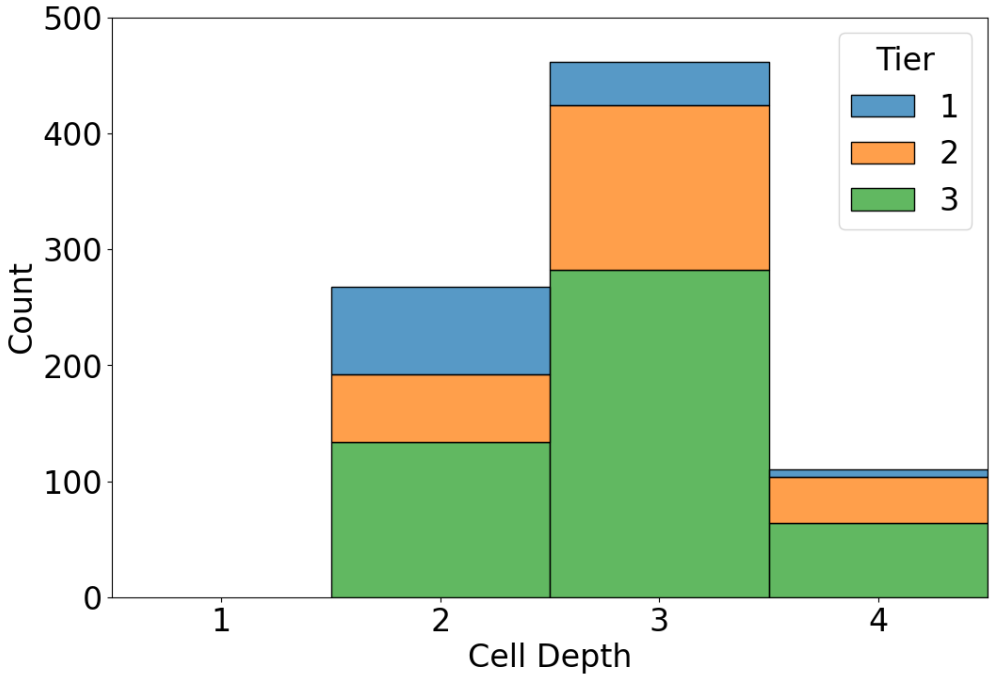


Figure 4.5: ICDARTS Dynamic Operation Space Algorithm Tiers Per Network Cell Depth Frequencies

Table 4.5: ICDARTS Alternative Cells CIFAR-10 Retraining Test Set Accuracies and Inference Latencies

Cell Type	Retraining Accuracy	Inference Latency (s/batch)
DARTS	97.13 (0.14)	0.10 (0.01)
I-DARTS	96.87 (0.27)	0.10 (0.01)
XDARTS	**97.21 (0.09)**	0.17 (0.01)

Table 4.6: Multiobjective ICDARTS Operation Latency Weights

Operation	Latency Weight
Avg Pool	0.06
Max Pool	0.04
Sep Conv 3x3	0.24
Sep Conv 5x5	0.36
Dil Conv 3x3	0.11
Dil Conv 5x5	0.15
Identity	0.04

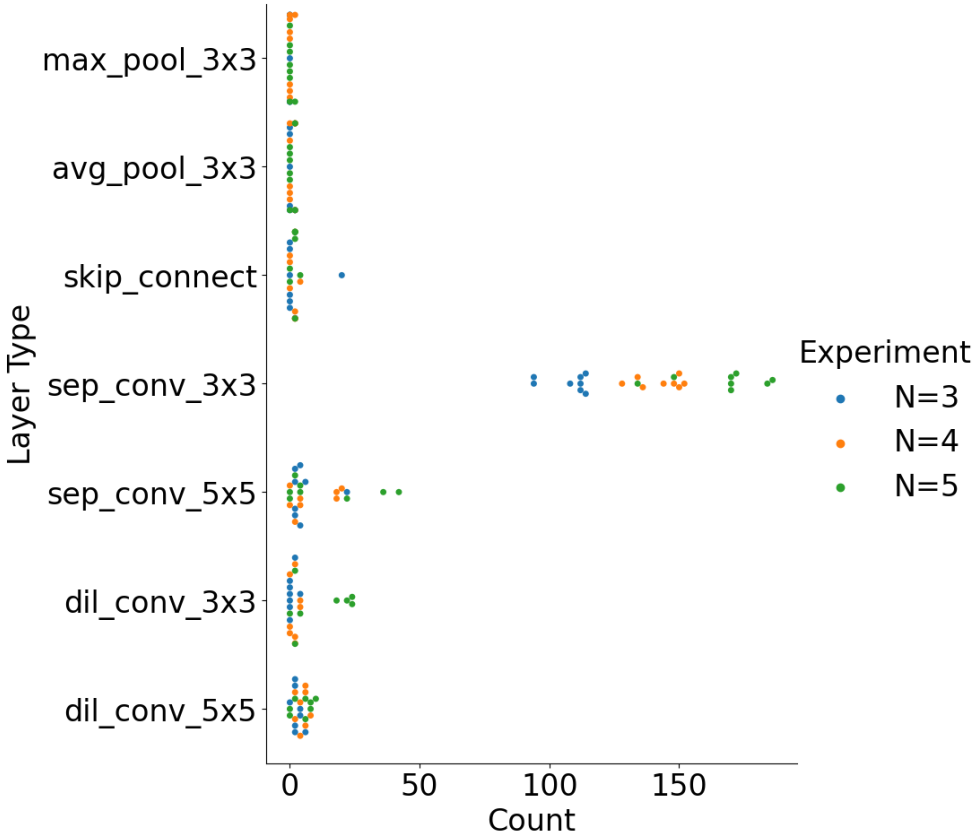


Figure 4.6: ICDARTS with Varying Numbers of Cells, N, per Node Per Network Layer Type Frequencies.

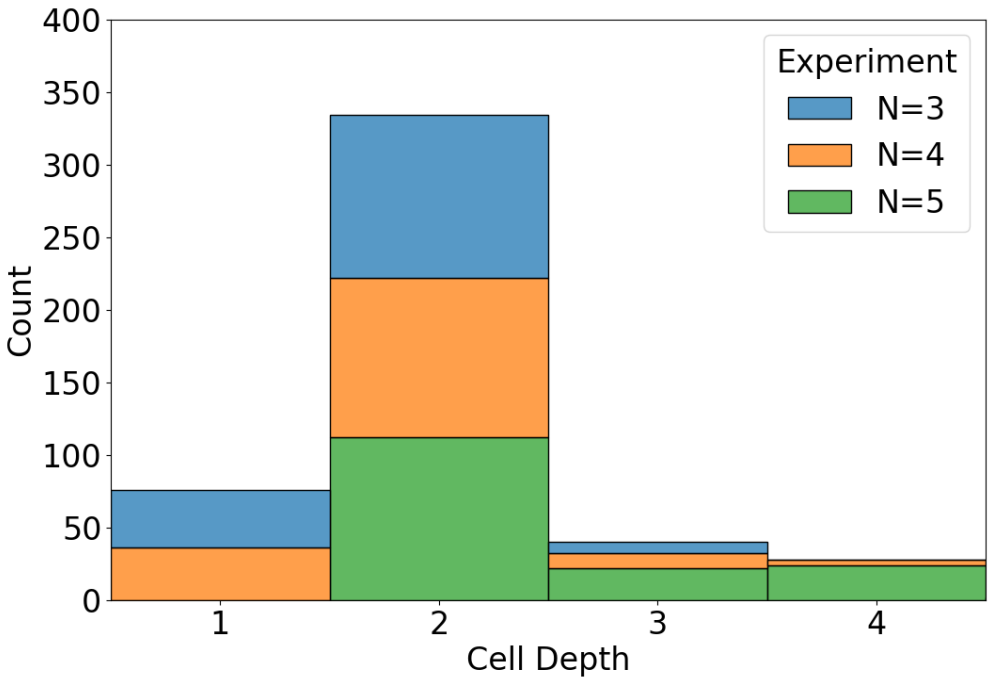
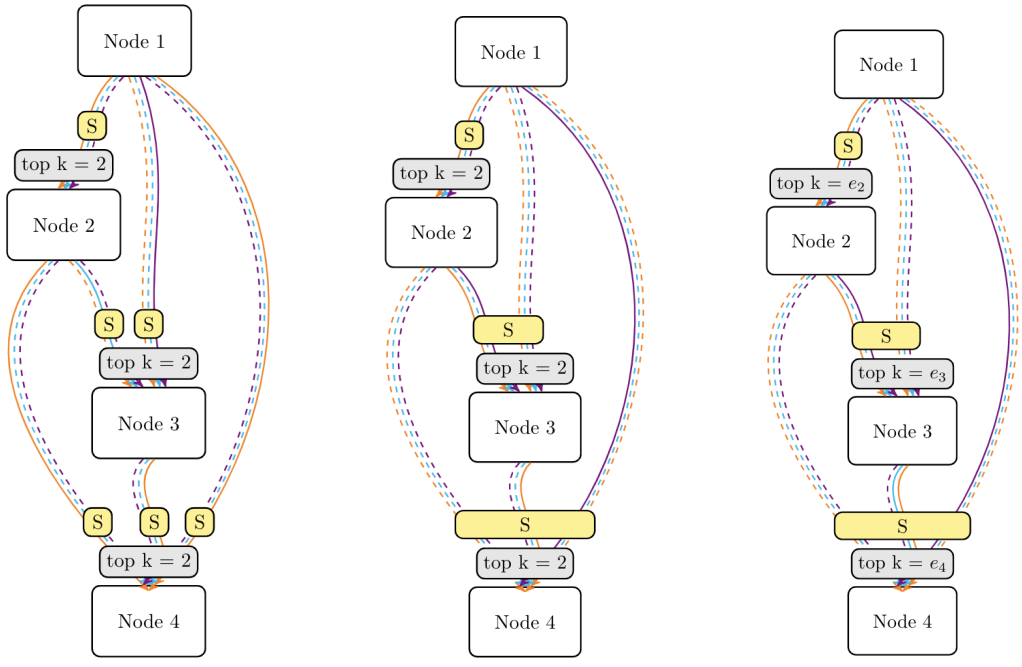


Figure 4.7: ICDARTS with Varying Numbers of Cells, N, per Node Per Network Cell Depth Frequencies.



(a) DARTS **(b)** I-DARTS **(c)** XDARTS

Figure 4.8: DARTS Cell Discretization Approaches

To derive the weights for each operation in the search space, a network previously produced by ICDARTS was selected, and its pre-training inference latency was averaged across twenty runs to obtain a baseline latency value. Then, using the same network, the average latencies of networks with each operation in the ICDARTS search space repeated ten times in the stemming layer were obtained using the same approach. The average latency of the baseline network was then subtracted from those of the networks corresponding to each layer choice to get the final latency weights for each operation. The resulting differences in latencies were finally normalized to get the latency weights for each operation, which are listed in Table 4.6.

The algorithm for calculating the latency penalty term then estimates the latencies of each cell node by weighting the softmax of alpha values corresponding to the incoming edges by the latency weights, factoring in the estimated latencies of previous cells when applicable. The maximum expected latencies of each cell are then obtained by multiplying the expected node latencies by the softmax of these values. Finally, the expected network latency is calculated by summing the maximum estimated latencies for each cell in the search network. This algorithm is shown in its entirety in Algorithm 3. The expected search network latency penalty term, P, is then weighted by a penalty coefficient, δ, and added to the search network's loss function to create a multiobjective version of ICDARTS:

$$w_S^* = \arg\min_{w_S} L_{train}^S(w_S, \alpha) + \lambda L_{train}^{S,E}(w_S, \alpha, w_E, \bar{\alpha}) + \delta P(\alpha) \tag{4.1}$$

Table 4.7 and Figures 4.11 and 4.12 show the results from running ICDARTS using both the XDARTS discretization strategy and the expected latency term with three different penalty coefficients, δ. Table 4.7 shows that the penalty term successfully reduced the inference latencies when δ was equal to 0.05 and 0.1. Surprisingly, the runs that used these δ values achieved higher average accuracies than those using a δ value of 0.0. This outcome may have occurred because the penalty term pressured the search algorithm to prefer shallower cells and less computationally expensive operations, such as dilated convolutions and identity operations, over separable convolutions. These attributes combined resulted in networks that converged to higher accuracies within 600 retraining epochs. The results also reveal

Algorithm 3 ICDARTS Latency Penalty Calculation Algorithm

Input: Latency Lookup Table L_T; Architecture Hyperparameters α, Number of Cell Types N_C, Number of Nodes per Cell N_N, Cell Counts for Each Type C
Output: Total Expected Network Latency P
Initialize P to zero
for each $i \in 0, ...N_C$ **do**
    Initialize l^c to zeros
    for each $j \in 0, ...N_N$ **do**
        Initialize N_E to $j + 2$
        Initialize l^n to zeros
        for each $k \in 0, ...N_E$ **do**
            Update l^n_k to L^T
            if $k \geq 2$ **then**
                Update l^n_k to $l^n_k + l^c_{k-2}$
            end if
        end for
        Update l^c_j to $l^c_j + \sum(l^n \alpha_{ij})$
    end for
    Update P to $P + C_i \times (\sum(l^c softmax(l^c))$
end for

an additional advantage of XDARTS cells: their ability to achieve high-quality results with shallow cells.

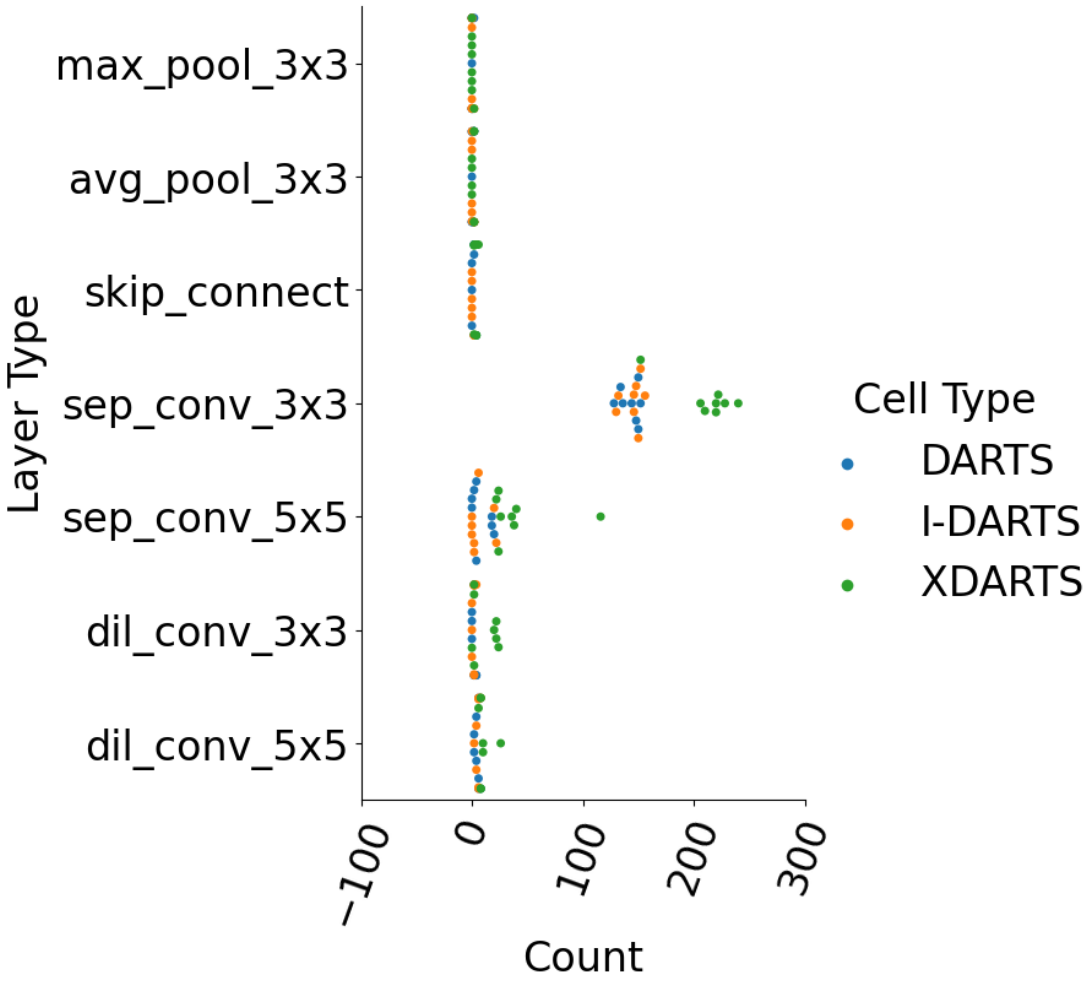


Figure 4.9: ICDARTS Per Network Layer Type Frequencies: Alternative Cells

Table 4.7: ICDARTS with XDARTS cells combined with Latency Loss Function CIFAR-10 Retraining Test Set Accuracies and Inference Latencies

Latency Coefficient, δ	**Retraining Accuracy (%)**	**Inference Latency (s/batch)**
0.0	97.21 (0.09)	0.17 (0.01)
0.01*	97.26 (0.10)	0.17 (0.01)
0.05*	**97.27 (0.18)**	0.14 (0.01)
0.1*	97.26 (0.18)	**0.13 (0.01)**

Note: Results from 6 runs are indicated by *.

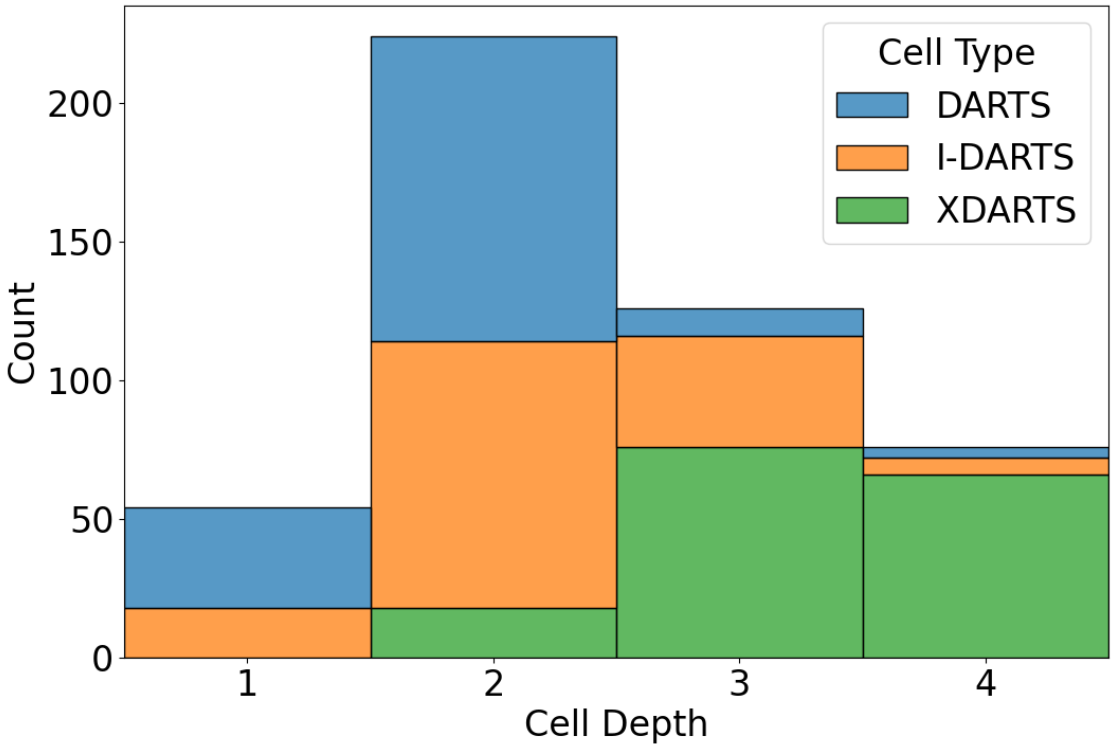


Figure 4.10: ICDARTS Alternative Cells Per Network Depth Frequencies

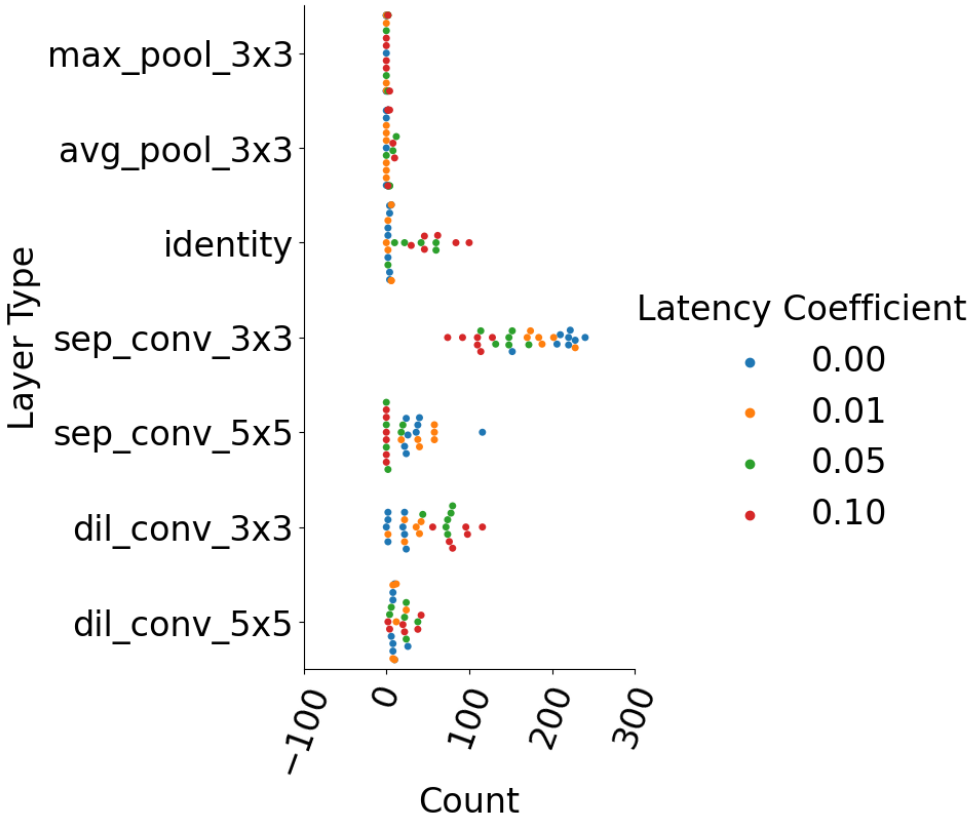


Figure 4.11: ICDARTS with XDARTS cells using latency loss function per network layer type frequencies. Figure 4.8 depicts each cell discretization protocol, and Algorithm 3 shows the latency estimation algorithm.

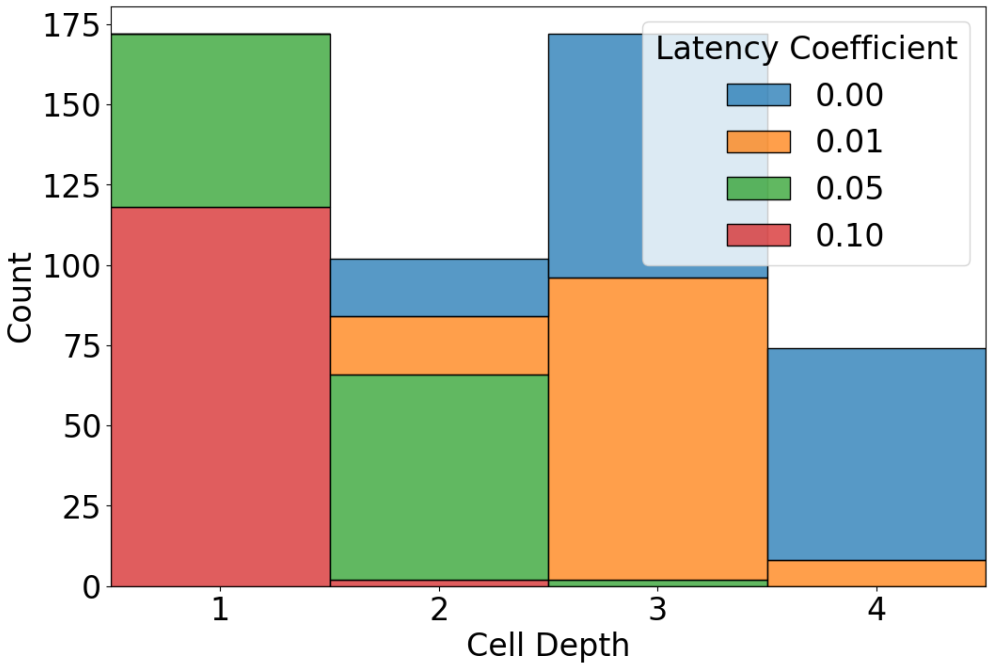


Figure 4.12: ICDARTS alternative operation spaces per network cell depth frequencies. The details of each operation space are listed in Table 4.1.

Chapter 5

ICDARTS for Alternative Benchmarks

This chapter discusses the results of applying the best of each category of improvements to CDARTS to two scientific datasets.

5.1 Stronglens Finding Challenge

The first scientific task used to evaluate the improvements to CDARTS was the Gravitational Lens Finding Challenge (SGLFC) 2.0. This astrophysics dataset consists of image candidates classified as members or non-members of a class of phenomena known as strong gravitational lenses, depending on their properties. Strong gravitational lenses represent rare scenarios in which a galaxy or quasar is so closely aligned with one or more galaxies that it creates multiple, highly distorted images of the foreground object. The lenses can take different forms depending on their sources and are often identified by visual inspection. They have provided valuable information to scientists, including how dark matter is distributed in galaxies, information for measuring cosmological parameters such as the Hubble constant, and for studying the structure of quasars. Since an ever-increasing magnitude of lens data is being generated, automated detection methods properly evaluated for predictive ability and bias for finding these rare objects are crucial for scientific discovery Metcalf et al. (2019).

The SGLFC was introduced for this reason. The most current version of this challenge, SGLFC 2.0, consists of 100,000 simulated images of strong lens candidates in 4 bands. Three bands, 'NISP J,' 'NISP Y,' and 'NISP H,' consist of 66 by 66 pixel images, and the final band, 'VIS,' consists of 200 by 200 pixel images. Candidate images are classified as lenses based on whether or not they satisfy a set of criteria in a provided log of lens properties. Figure 5.1 shows heatmaps of the four bands of a positive sample. 80%

Before running the improved versions of CDARTS on this dataset, the default template of ICDARTS was modified to accommodate the increased memory demands of the much larger dataset. First, the evaluation network depths were reduced to 14, with the reduction cells placed at the 5th and 10th layers. The search phase was then run for 40 epochs, with ten pretraining epochs for the search network and three warmup epochs for the evaluation network. The network weights are updated with a learning rate of 0.8 and decay rate of 1.0×10^{-5}, and the alpha weights were updated with a learning rate of 3.0×10^{-4} and decay rate of 1.0×10^{-5}. The retraining phase was run for 250 epochs, and the network was trained with a learning rate of 0.25 and a decay rate of 3.0×10^{-5}. Before training, the samples from each band were normalized using min-max scaling. Then, in the stemming layer of the search and evaluation networks, the four bands were combined despite their differing dimensions by reducing the dimension of the 'VIS' band to match that of the NISP 'J,' 'H,' and 'Y' bands and the four bands were concatenated together to serve as single input with four channels.

Three different metrics were used for evaluating the performance of ICDARTS on the SGLFC 2.0 dataset. The first was the traditional accuracy metric, the second was the area under the receiver operating characteristic curve (ROC AUC), and the third was the F_β score. The F_β score was the recommended scoring metric for the SGLFC 2.0 dataset and is defined as

$$F_\beta = (1 + \beta^2)\frac{P \times R}{\beta^2 P + R} \tag{5.1}$$

where P is precision, R is recall, and β qualifies the relative performance of precision and recall changes. To evaluate the performance of searched networks using the F_β score, the

maximum value the F_β score reaches for any threshold, p:

$$F_\beta = \max_p F_\beta(p)$$

Since actual lenses are rarer in real-world data than in the simulated dataset, by a factor of about 1000, β^2 is set to 1.0×10^{-3} to ensure an adequately high precision or a low rate of false positives Metcalf (2019).

Table 5.1 and Figures 5.2 and 5.3 display the results of applying CDARTS and the best configurations of ICDARTS to the SGLFC 2.0 dataset. As reported in Table 5.1, the highest retraining F_β scores were achieved by running ICDARTS on a dynamic operation space, and the lowest F_β scores came from the CDARTS runs. The rankings of the configurations are very different when considering the ROC AUC and accuracy scores, of which ICDARTS obtained the best results when using the second operation set, and the worst results came from the original configuration of ICDARTS and ICDARTS using the dynamic operation space. These differences in performance metrics suggest that although configurations with high retraining ROC AUC and accuracy results perform better on the simulated data, the configurations with the highest F_β scores should provide more reliable results on real-world data. The standard deviations of each performance metric of the ICDARTS runs were lower than those of the CDARTS runs, with the only exception being ICDARTS using the dynamic operation space. The latency values from this configuration were also the highest overall. Both of these outcomes were likely due to this configuration favoring deeper cells and various operations, including many MBConv blocks. The configurations that produced the lowest F_β scores, ICDARTS and ICDARTS using operation space 2, trended towards shallower cells. The contrast between the performances of ICDARTS on the second operation space, the original operation space, and the dynamic operation space reveals that a diverse operation space with more complex operations, such as MBConvs, is better suited to this problem.

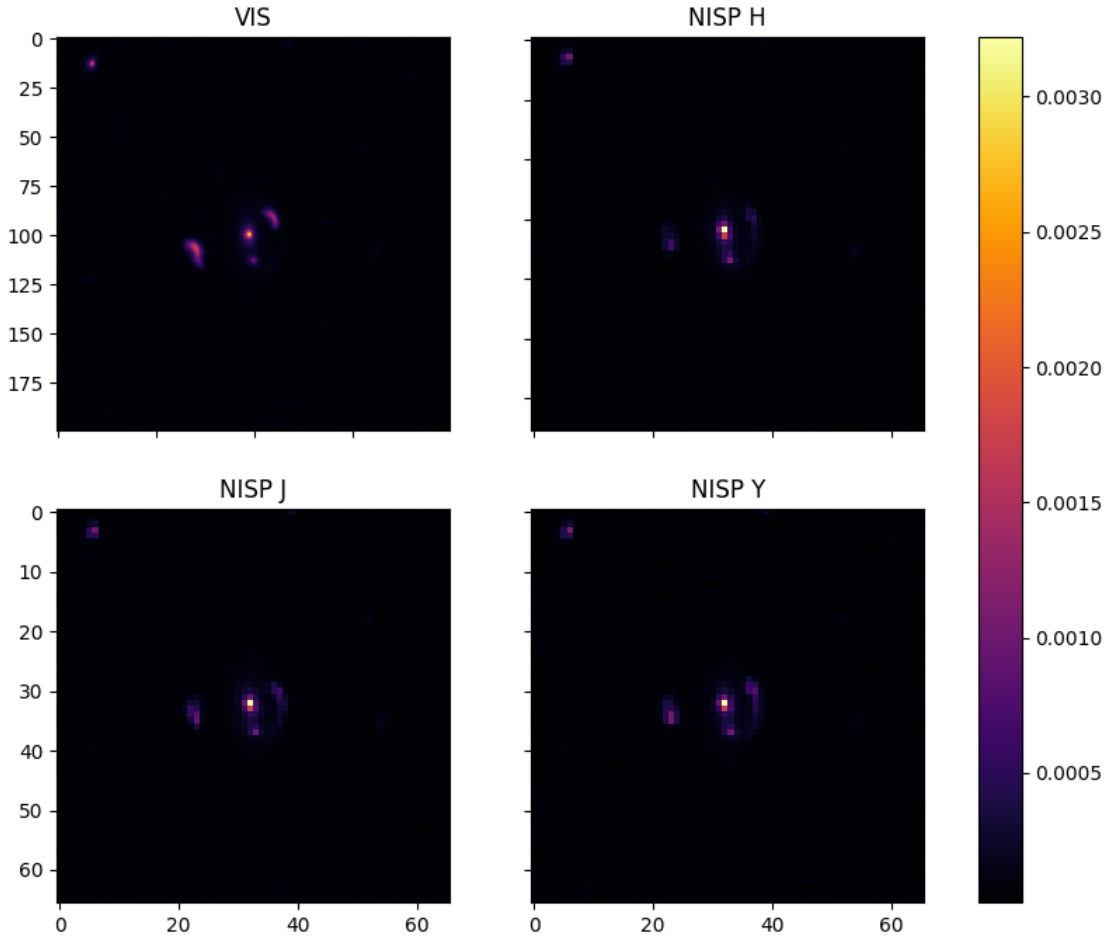


Figure 5.1: Heatmaps taken from normalized 'VIS' and NISP 'J,' 'H,' and 'Y' bands of one positive sample from the Stronglens Finding Challenge 2.0.

Table 5.1: Stronglens Finding Challenge Retraining Test Set Accuracies and Inference Latencies

Configuration	Retraining F_β	Retraining ROC AUC	Retraining Accuracy (%)	Inference Latency (s/batch)
CDARTS	0.79 (0.06)	0.73 (0.03)	70.54 (3.13)	**1.40 (0.05)**
ICDARTS	0.83 (0.06)	0.72 (0.02)	69.48 (2.14)	1.46 (0.00)
ICDARTS Operation Space 2	0.83 (0.05)	**0.75 (0.02)**	**71.48 (1.38)**	1.47 (0.02)
ICDARTS XDARTS Cells $\epsilon = 0.05$	0.85 (0.06)	**0.75 (0.02)**	70.33 (1.70)	1.45 (0.13)
ICDARTS Dynamic Operation Space	**0.88 (0.07)**	0.72 (0.04)	68.11 (2.15)	1.93 (0.15)

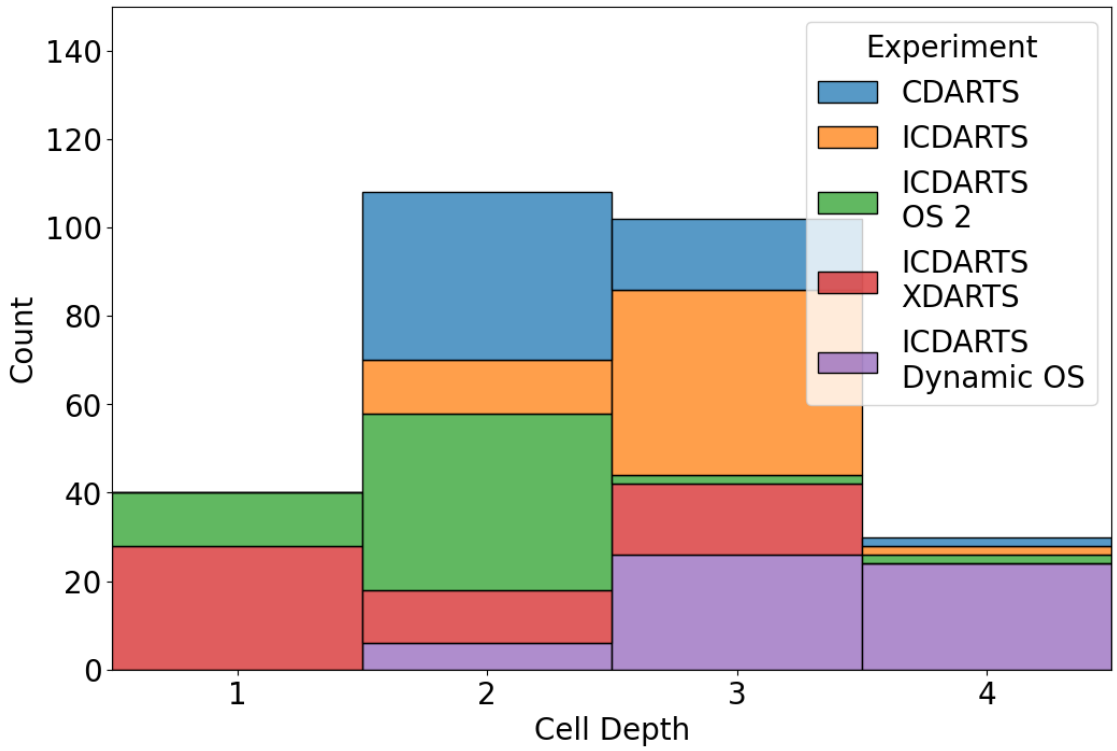


Figure 5.2: SGLFC 2.0 per network cell depth frequencies of networks searched by CDARTS and the best configurations of ICDARTS.

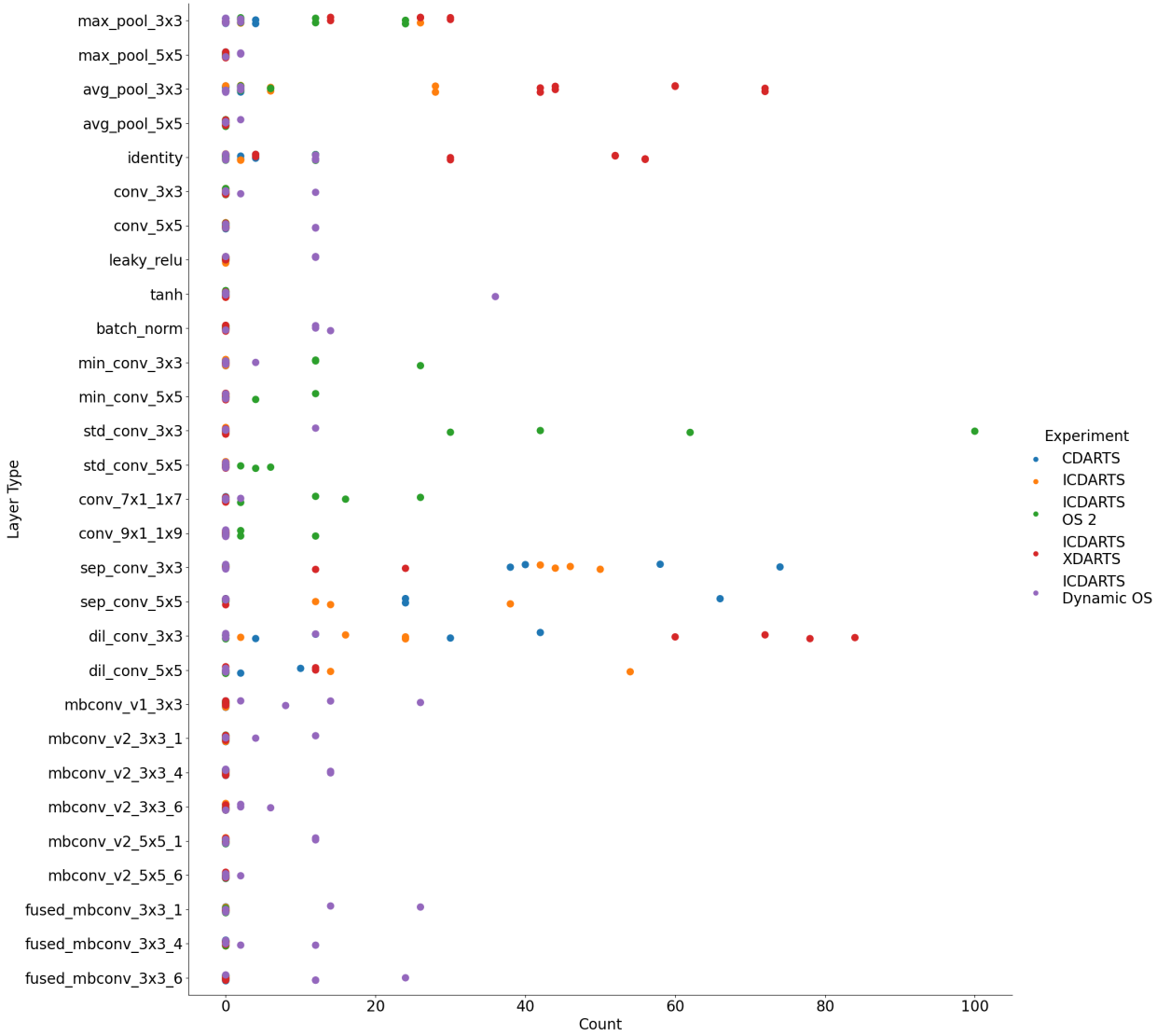


Figure 5.3: SGLFC 2.0 per network operation frequencies of networks searched by CDARTS and the best configurations of ICDARTS.

5.2 Electron Microscopy Dataset

CDARTS and the best overall configurations of ICDARTS were also evaluated on an electron microscopy (EM) dataset consisting of a 1065x2048x1536 stack of images representing a 5 square micrometer section of the brain's hippocampus region. Graham Knott and Marco Cantoni originally compiled this dataset at the Swiss Federal Institute of Technology Lausanne to accelerate neuroscience research. Two sub-volumes of the dataset, consisting of 165 slices of the image stack, have been annotated with the locations of mitochondria and synapses Lucchi et al. (2013).

To convert the stacks of images in the training and test subvolumes into training and test datasets that could be used for optimizing CDARTS and ICDARTS networks, 32 by 32 windows were randomly cropped from the slices of the training and test subvolumes. For the training set, 300 crops were taken per slice, and for the testing set, 60 crops were taken.

To ensure that the dataset was balanced, positive and negative samples were alternatively selected by repeatedly cropping windows from the image slice until obtaining a sample from the desired class. A cropped image sample was considered a positive if more than 7 of its 16 center pixels corresponded to the regions of mitochondria or synapses. Samples were marked as negative if no positive pixels were found in its 16 center pixels. Figure 5.4 shows some examples of positive and negative samples. Following normalization of image crops using min-max scaling, CDARTS and the best configurations of ICDARTS were then run on this dataset using the search and retraining settings used for the CIFAR dataset, except for the number of input channels, which was changed from 3 to 1, the learning rate of the search phase networks set to 0.2, and the learning rate of the evaluation phase network set to 6.25×10^{-3}.

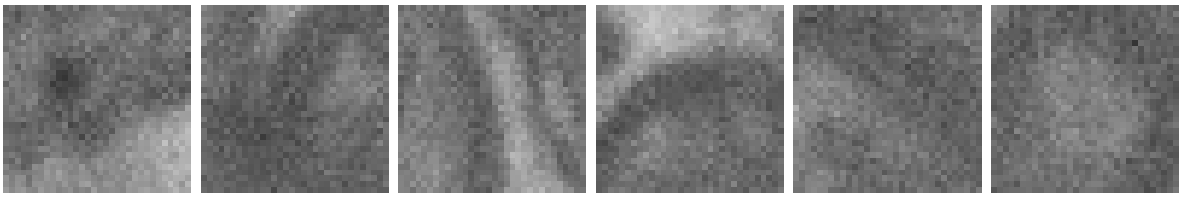
Table 5.2 and Figures 5.5 and 5.6 show the results of applying CDARTS and the best configurations of ICDARTS to the EM dataset. Table 5.1 shows that ICDARTS using a dynamic operation space produced the best overall results regarding average retraining accuracies. The networks produced by this configuration were slightly deeper than those produced by other configurations (see 5.5) and commonly preferred ReLU activations, MBConv V1, and Fused MBConv operations. The configuration of ICDARTS that used

XDARTS cells achieved the lowest average inference latency, likely due to this configuration's preference for shallower cells.

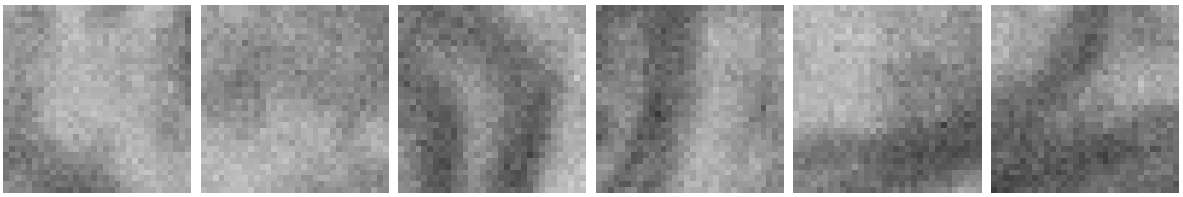
5.3 Comparing Architectures Across Tasks

This section compares the best architectures produced for the CIFAR-10, SGLFC 2.0, and EM datasets. Figure 5.7 shows the per network cell depth frequencies of CDARTS and each of the best ICDARTS configurations on the CIFAR-10, SGLFC 2.0, and EM. Although the distributions of cell depth frequencies varied across configurations, the cells discovered for the EM dataset tended to be deeper for most configurations. Likewise, Figures 5.8 and 5.9 show each search method's per network operation frequencies on each of the three datasets. The configurations that relied on the original operation space preferred a relatively diverse selection of operations when searching networks for the SGLFC 2.0 and EM datasets, in contrast to the networks searched for CIFAR-10, which overwhelmingly preferred separable convolutions. The runs of ICDARTS on operation space two on the SGLFC 2.0 and EM tasks overwhelmingly favored standard convolutions. By contrast, CIFAR-10 more strongly favored identity and $N \times 1$ convolution operations. The runs of ICDARTS using the dynamic operation space all tended to favor MBConv and activation operations. With the CIFAR-10 dataset, MBConv V1 operations were most favored, while the SGLFC 2.0 and EM tasks preferred a more diverse selection of MB Conv V1 and Fused MBConv operations.

Figures 5.10, 5.11, and 5.12 show the normal and reduction cells from the networks with the best overall performance on the CIFAR-10, SGLFC 2.0, and EM tasks. The cells from the best overall network on CIFAR-10, which ICDARTS produced on the second operation space, achieved a final retraining test accuracy of 97.62%. These cells comprise a diverse set of operations from this operation space and an architecture of depth 3 in the case of the reduction cell. The top-performing network on the SGLFC 2.0 dataset was discovered by running ICDARTS with the dynamic operation space, and its final retraining F_β score was 96.62. These cells consisted of many MBConv operations with some batch norms and activation operations. In addition to these features, its normal cell was four nodes deep.



(a) Positive Samples



(b) Negative Samples

Figure 5.4: Examples of electron microscopy image samples. Images belonging to the positive class contain significant regions corresponding to mitochondria or synapses within their four-by-four center region, while samples belonging to the negative class contain no positive pixels within their center region.

Table 5.2: EM Dataset Retraining Test Set Accuracies and Inference Latencies

Configuration	**Retraining Accuracy (%)**	**Inference Latency (s/batch)**
CDARTS	96.80 (0.58)	0.09 (0.01)
ICDARTS	97.01 (0.05)	0.09 (0.02)
ICDARTS Operation Space 2	96.93 (0.09)	0.10 (0.02)
ICDARTS XDARTS Cells $\delta = 0.05$	96.98 (0.21)	**0.08 (0.01)**
ICDARTS Dynamic Operation Space	**97.03 (0.09)**	0.12 (0.01)

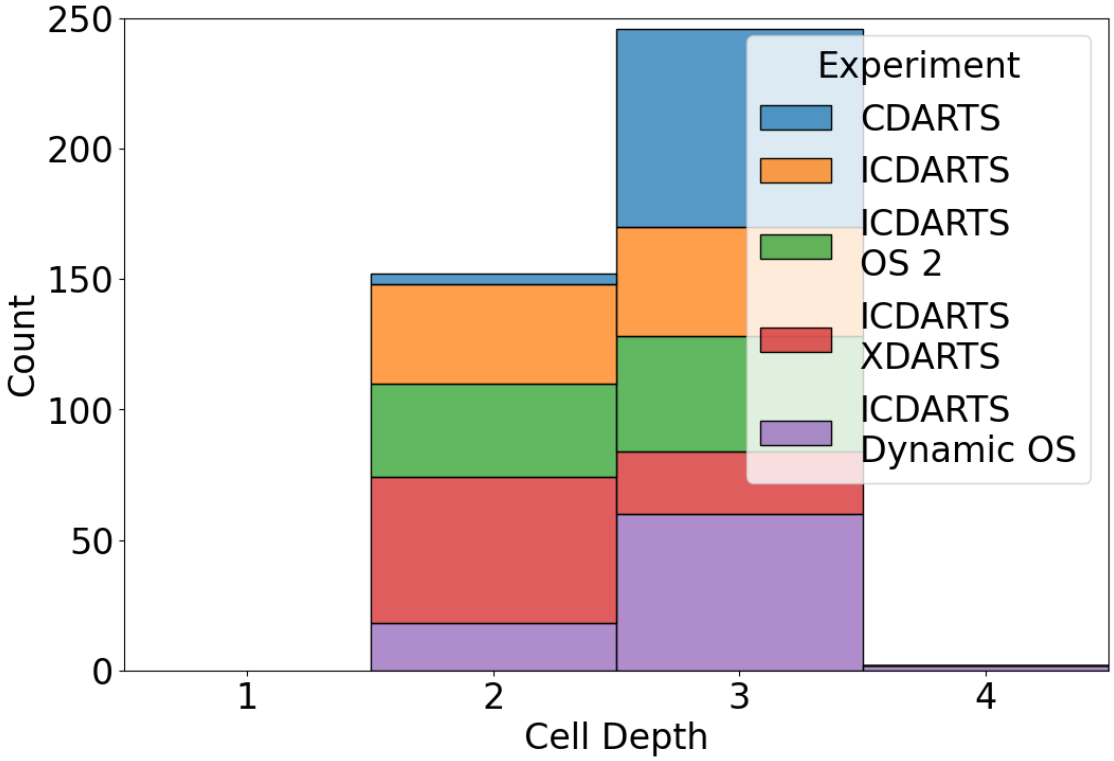


Figure 5.5: EM dataset per network cell depth frequencies of networks searched by CDARTS and the best configurations of ICDARTS.

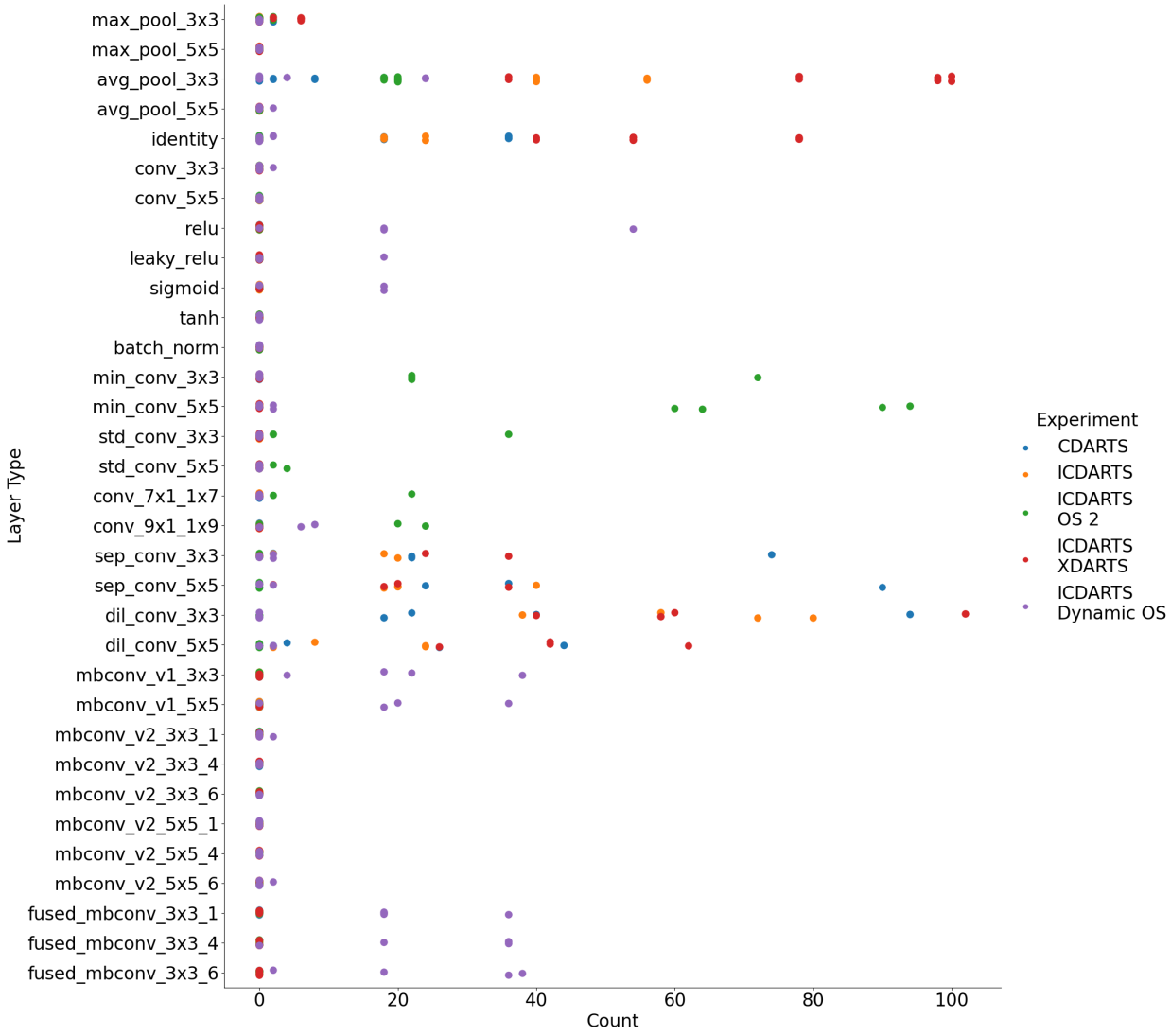
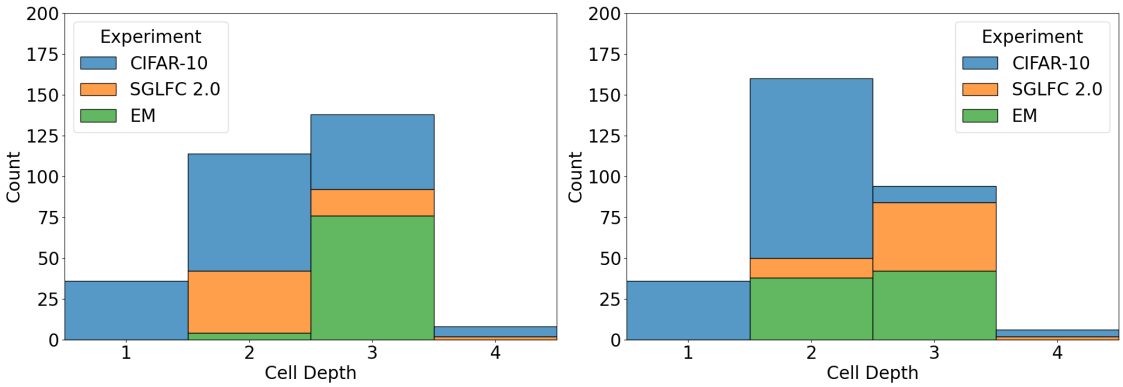
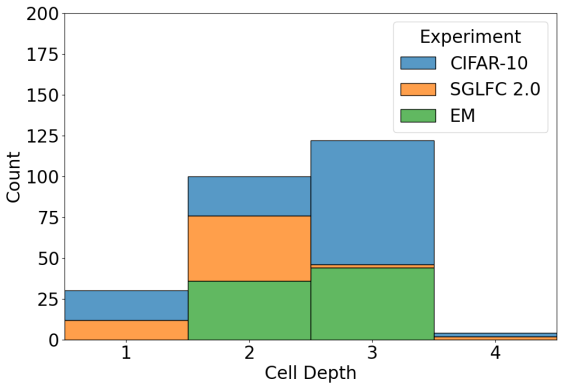


Figure 5.6: EM dataset per network operation frequencies of networks searched by CDARTS and the best configurations of ICDARTS.

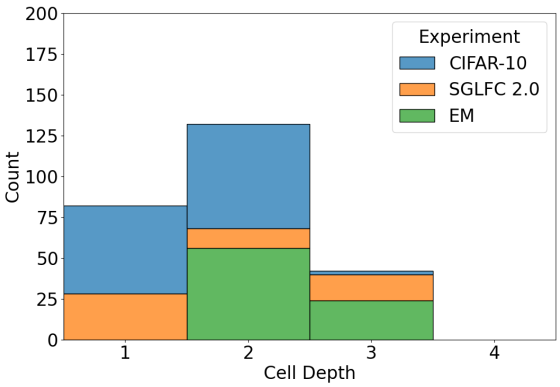


(a) CDARTS

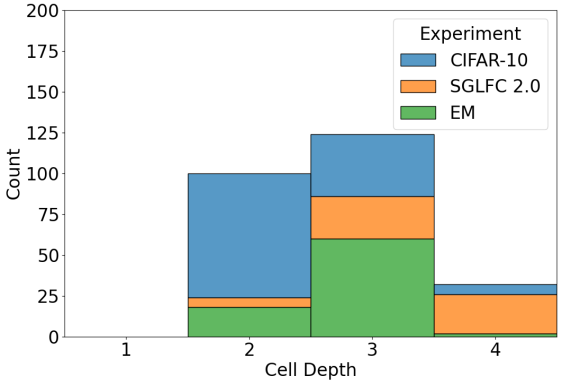
(b) ICDARTS



(c) ICDARTS Operation Space 2

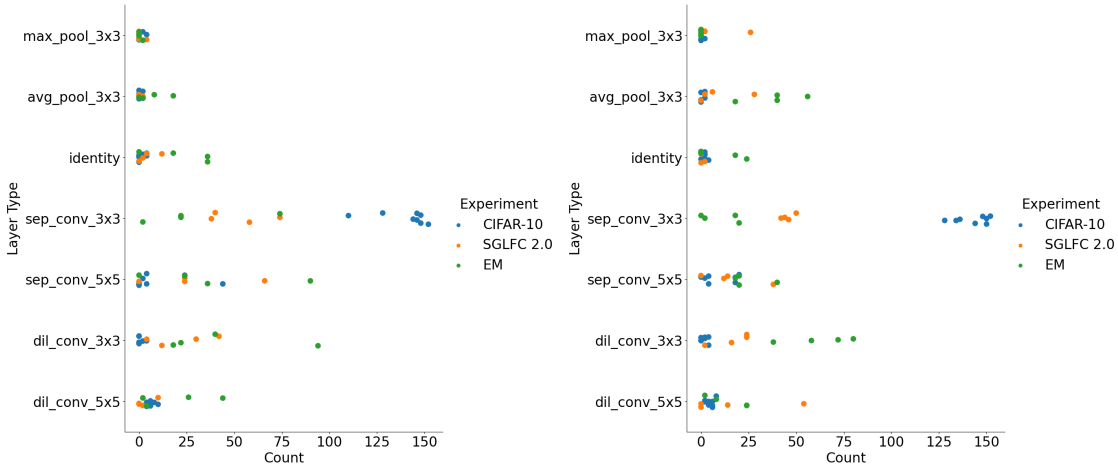


(d) ICDARTS XDARTS Cells $\delta = 0.05$



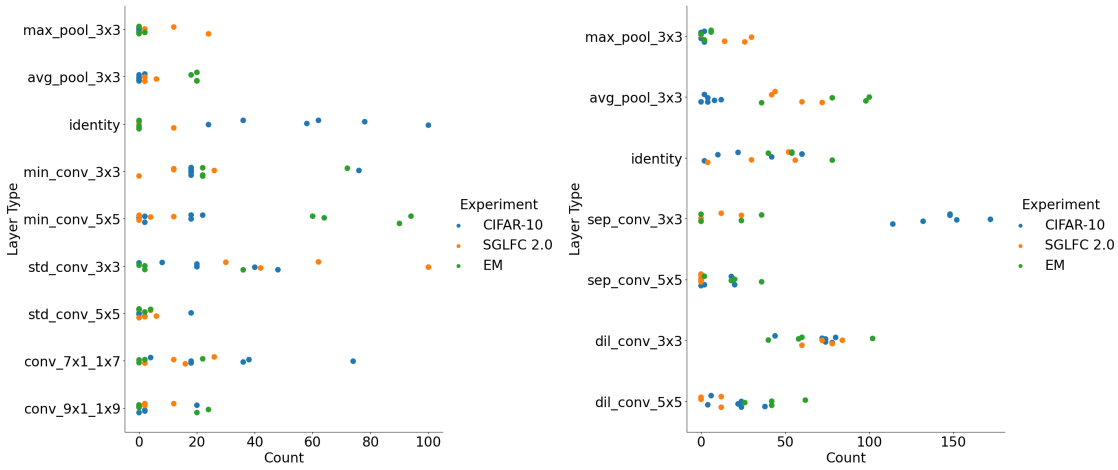
(e) ICDARTS Dynamic Operation Space

Figure 5.7: CIFAR-10, SGLFC 2.0, and EM datasets per network cell depth frequencies of networks searched by CDARTS and the best configurations of ICDARTS.



(a) CDARTS

(b) ICDARTS



(c) ICDARTS Operation Space 2

(d) ICDARTS XDARTS Cells $\delta = 0.05$

Figure 5.8: CIFAR-10, SGLFC 2.0, and EM datasets per network operation frequencies of networks searched by CDARTS and the best configurations of ICDARTS.

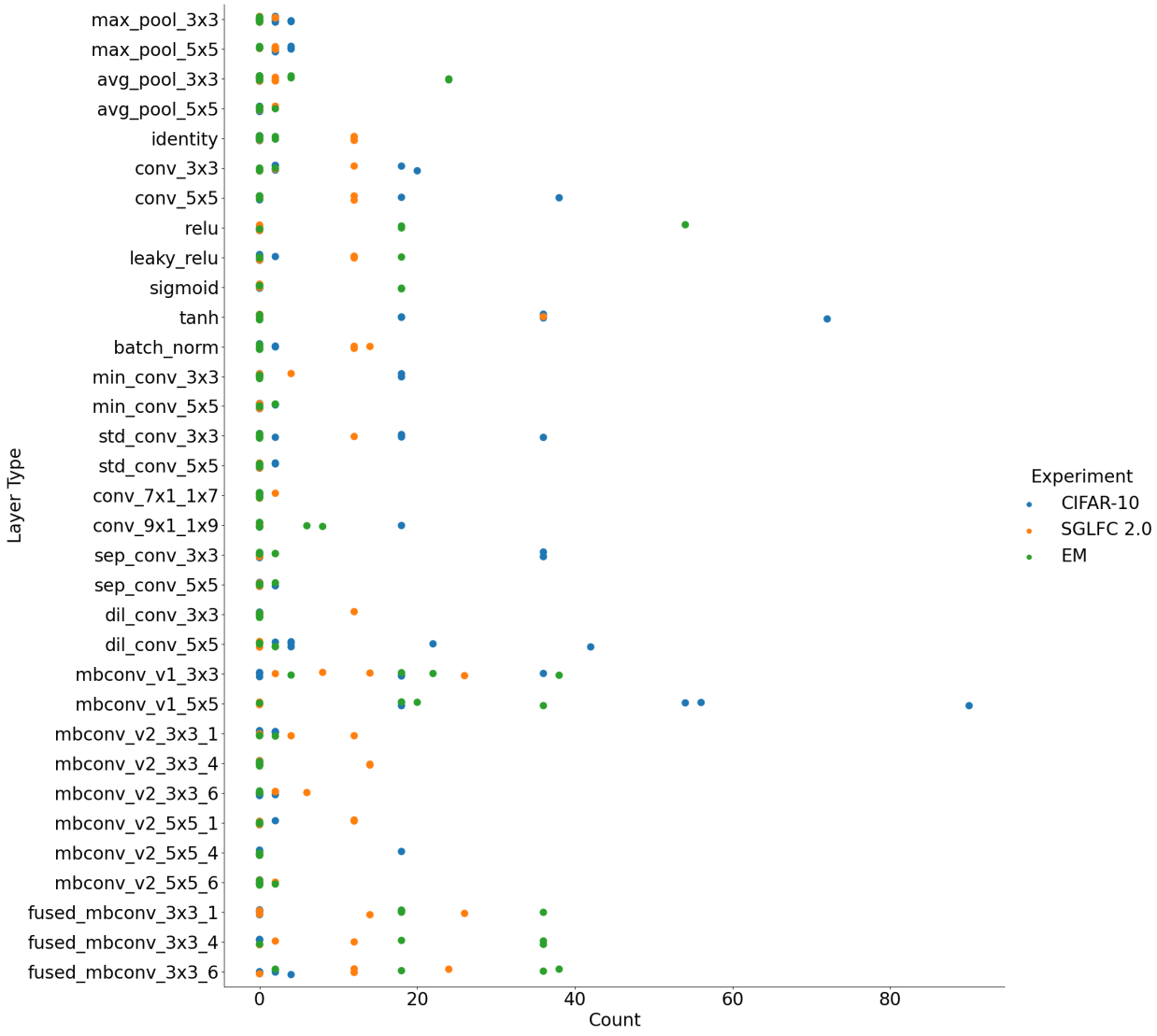
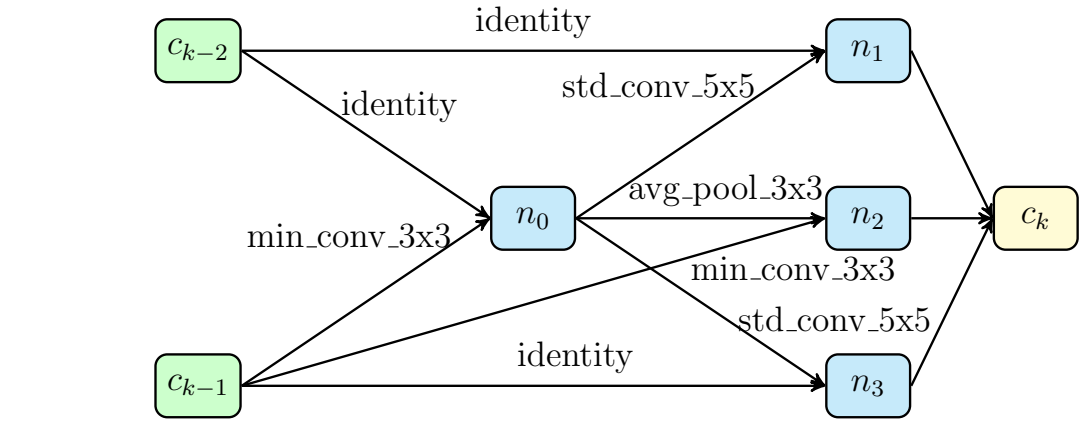
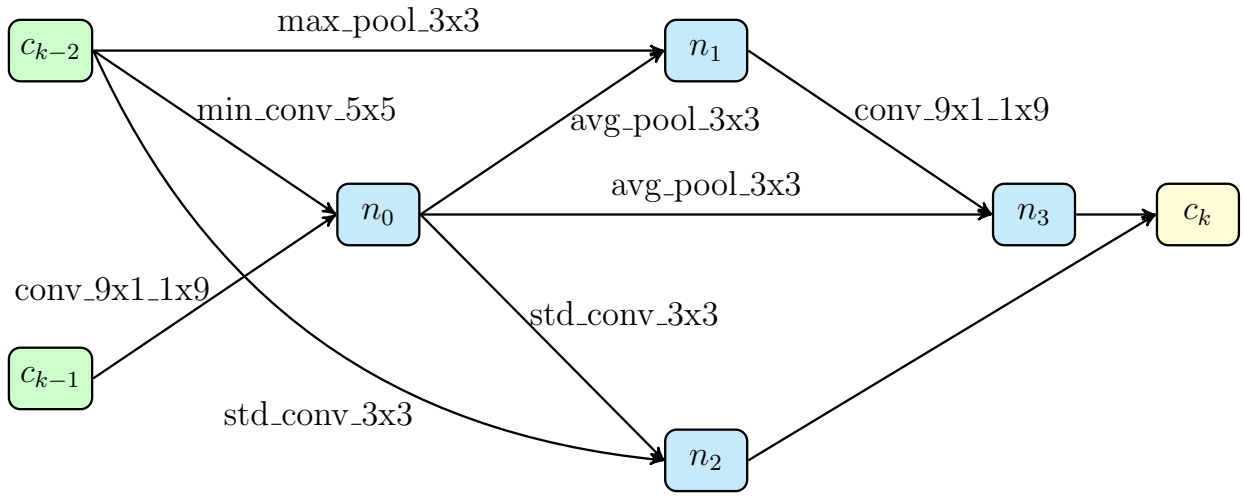


Figure 5.9: CIFAR-10, SGLFC 2.0, and EM datasets per network operation frequencies of networks searched by ICDARTS using a dynamic operation space.

Finally, the best network on the EM dataset, which was discovered using ICDARTS with XDARTS cells and a latency penalty coefficient of 0.05, obtained a final retraining accuracy of 97.15%. Its normal cells were composed of a mixture of diluted convolutions, separable convolutions, identity, and pooling operations. Meanwhile, its normal cells were composed of mostly dilated convolutions and pooling operations. Both cell structures have a depth of three.

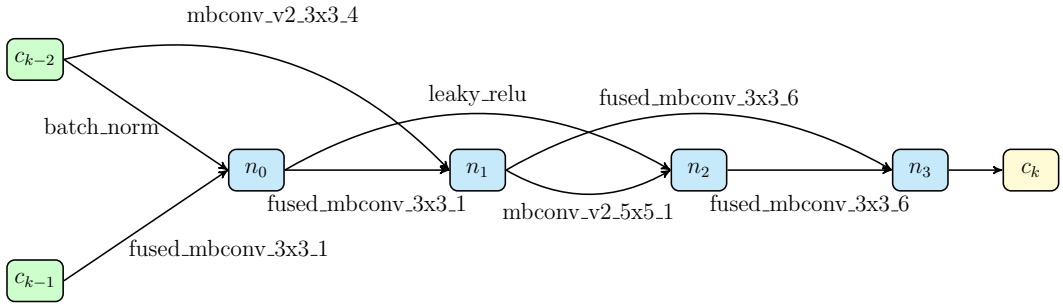


(a) Normal Cell

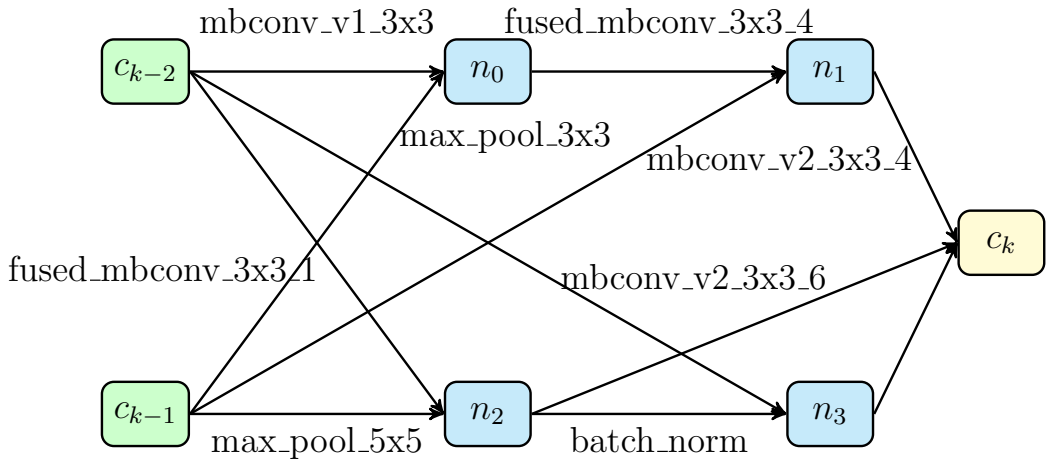


(b) Reduction Cell

Figure 5.10: Normal and Reduction Cells from Top-performing ICDARTS Network on CIFAR-10 Dataset: ICDARTS Operation Space 2

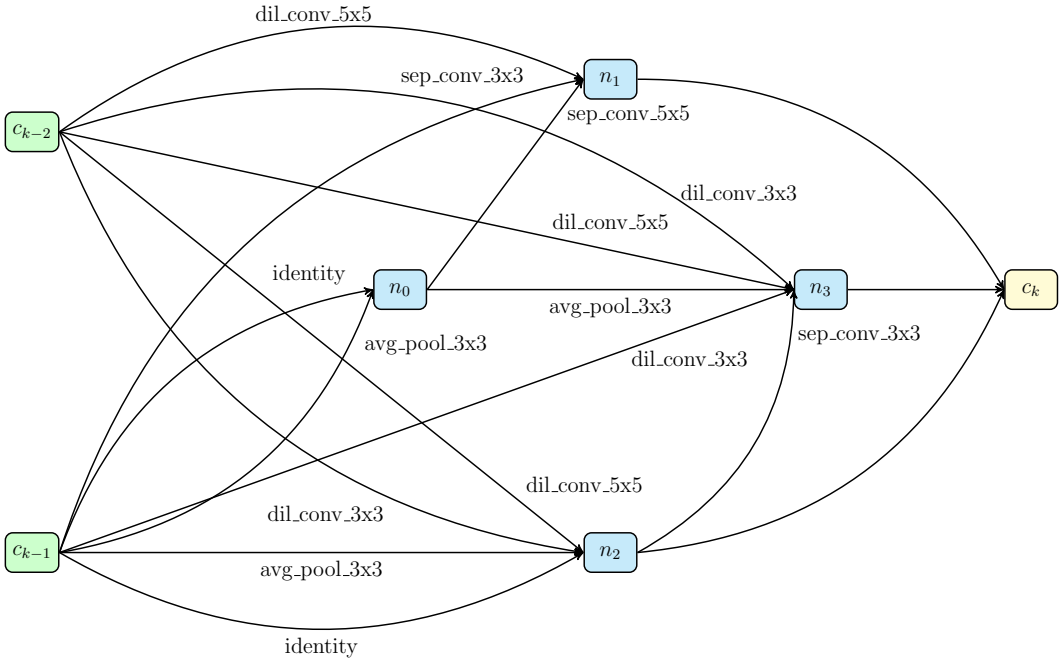


(a) Normal Cell

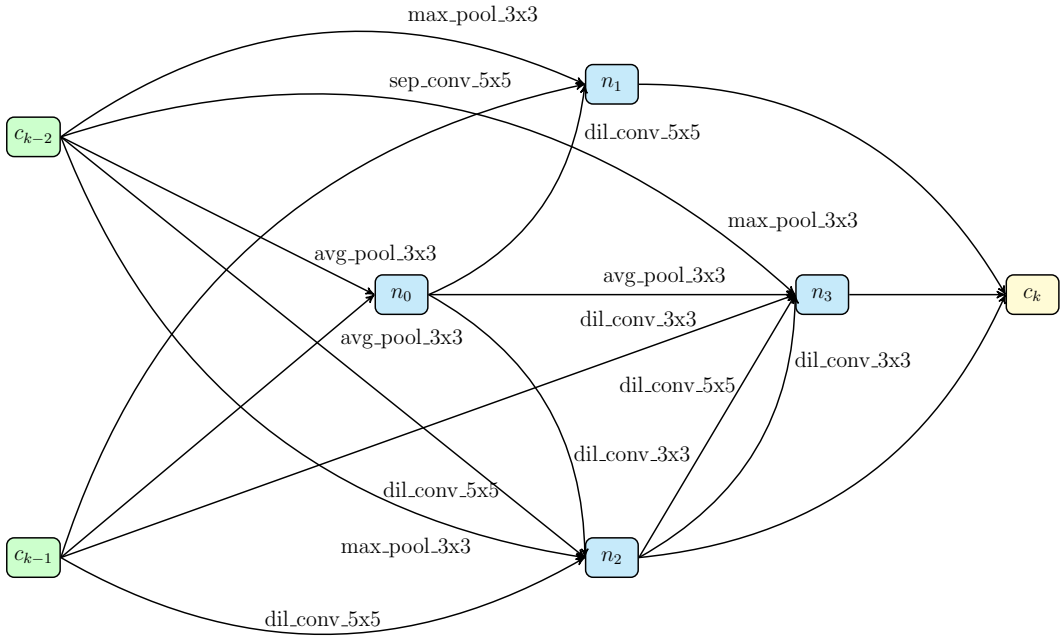


(b) Reduction Cell

Figure 5.11: Normal and Reduction Cells from Top-performing ICDARTS Network on SGLFC 2.0 Dataset: ICDARTS Dynamic Operation Space



(a) Normal Cell



(b) Reduction Cell

Figure 5.12: Normal and Reduction Cells from Top-performing ICDARTS Network on EM Dataset: ICDARTS with XDARTS Cells, $\delta = 0.05$

Chapter 6

Hierarchical NAS

This section proposes a hierarchical version of ICDARTS that optimizes network templates and cell structures. As previously discussed, CDARTS and ICDARTS discover networks within a modular search space by optimizing cell structures, which are then copied and stacked according to the specifications of a pre-determined template. The templates of the search and evaluation networks (shown in Figure 3.1) determine the placement of each cell structure in their respective architectures, how many times the cells are repeated, the number of channels at each depth, and the transformations applied at the beginning and end of each network. It also dictates that each cell accepts the outputs of the two previous cells as inputs and that the networks include auxiliary heads that combine logits output at different network depths. Although modular NAS methods tend to be more computationally efficient, their reliance on repeating cells and pre-determined templates limits the flexibility of their search spaces. Hierarchical NAS approaches address this criticism by optimizing network templates given a set of one or more cell structures that are pre-determined or searched in a previous step.

To incorporate hierarchical search into ICDARTS, an intermediate search phase is introduced following the initial search phase that optimizes the template of the final network given the set of cell structures optimized in the initial search phase. Following a similar approach to the initial search phase, the template search space represents a large DAG with nodes and edges corresponding to cells discovered by the initial search phase and prospective connections between cells, respectively. Figure 6.1 compares the original and intermediate

search phases and shows how both cyclicly optimize search and evaluation networks. The initial cell-level search phase first leverages ICDARTS (see Algorithm 1) to cyclically optimize the search network, S_{micro} and evaluation network, E_{micro}. The intermediate search phase then takes the cell structures discovered in the micro-level search phase and optimizes their connectivity and arrangement within a new evaluation network E_{macro}. Within E_{macro},a set of γ weights, analogous to the α weights of the original search network, S_{micro}, are learned within the search network graph S_{macro}. The γ, w_S, and w_E weights are updated according to the following loss functions:

$$w_E^* = \arg\min_{w_E} L_{val}^E(w_E, \bar{\gamma}) \tag{6.1}$$

$$\gamma^* = \arg\min_{\gamma} L_{val}^S(w_S^*, \gamma) + \lambda L_{val}^{S,E}(w_S^*, \gamma, w_E^*, \bar{\gamma}) \tag{6.2}$$

$$w_S^* = \arg\min_{w_S} L_{train}^S(w_S, \gamma) + \lambda L_{train}^{S,E}(w_S, \gamma, w_E, \bar{\gamma}) \tag{6.3}$$

$$w_E^* = \arg\min_{w_E} L_{train}^E(w_E, \bar{\gamma}) \tag{6.4}$$

At each iteration, the learned weights generate a new evaluation network E_{macro} until convergence. This process, shown in full in Algorithm 4, closely resembles that of the original ICDARTS algorithm.

This chapter proposes two different versions of hierarchical search for ICDARTS. The first version draws inspiration from Huang et al. (2016) and O'Neill et al. (2021) and has been designed to optimize the skip connections between the cells within the original network template. As stated above, in the original network template, each cell takes the output of the previous cell as well as a skip connection from the cell preceding the previous cell as inputs. This version of hierarchical search searches the set of all possible skip-connect inputs to a given cell to determine which is most optimal. In this case, the designs of the cell-level search phase and networks follow those of the original ICDARTS search phase closely. The search and evaluation networks of the intermediate search phase are of the

Algorithm 4 ICDARTS Macro-Level Search Phase

Input: Datasets *train* and *val*, search and evaluation iterations S_S and S_E, update iterations S_U, architecture hyperparameter γ, and weights w_S and w_E for search S and evaluation E networks
Output: Evaluation network E
Initialize γ randomly
Initialize w_S
for each search step $i \in [0, S_S]$ **do**
  if $iModS_U$ **then**
    Discretize γ to $\bar{\gamma}$ by selecting the top k
    Generate E with $\bar{\gamma}$
    for each evaluation step $j \in [0, S_E]$ **do**
      Calculate L_{val}^{E}
      Update w_E according to Eq. 6.1
    end for
  end if
  Calculate L_{val}^{S}, L_{val}^{E}, and $L_{val}^{S,E}$
  Update γ according to Eq. 6.2
  Calculate L_{train}^{S} and $L_{train}^{S,E}$
  Update w_S according to Eq. 6.3
  Calculate L_{train}^{E}
  Update w_E according to Eq. 6.4
end for

same depth. In the search network, S_{macro}, a set of γ weights are placed at each incoming skip connection to a cell and initialized so that the skip connections closest to a given node are weighted most heavily. To generate the evaluation network, E_{macro}, at each iteration, all incoming connections to each node except for the one with the highest γ value and that of the immediately preceding node are discarded. The search network, S_{macro}, and an example evaluation network corresponding to this implementation of the network level search phase are both pictured in Figure 6.2.

The second version of hierarchical search concentrates on optimizing the placement of cells in the final architecture. The initial search phase is modified to search four normal cell structures rather than one normal and one reduction cell structure. This change requires re-imagining the micro-level search and evaluation network templates to specify that the four normal cells alternate in both networks, as shown in Figure 6.3. The networks otherwise follow the templates of the original search phase. The search network of the intermediate

search phase is constructed so that one of each type of cell exists at each level. Each cell takes as input the γ weighted sum of the outputs of the cells from the two previous layers, as shown in the network on the left-hand side of Figure 6.4. All cell types except those corresponding to the single highest γ weight at each depth are discarded to generate the evaluation network. An example of an evaluation network generated by this version of the hierarchical search is shown on the right side of Figure 6.4.

The initial search phases of both versions of hierarchical ICDARTS used the exact parameters of the original ICDARTS search space. However, the macro-level search phases of both implementations used a learning rate of 3×10^{-3}. The second version also pre-trained the evaluation network for two epochs in each iteration rather than one to account for the significant structural differences between the search and evaluation networks.

The results of both configurations of hierarchical ICDARTS are shown in Tables 6.1 and 6.2, which list the average accuracies, standard deviations, and inference latencies of the initial and intermediate evaluation networks of each configuration of hierarchical ICDARTS. These results reveal that only the networks produced by the second configuration of ICDARTS outperformed the default networks produced by their initial search phases. Figures 6.5 and 6.6 show that this configuration of hierarchical ICDARTS preferred cells of roughly the same depth but consisting of a higher proportion of separable convolutions with a kernel size of 3, the operation of the default operation that has been most preferred on the CIFAR-10 benchmark. The average latencies of these networks were also nearly identical to those of the default networks produced by the initial search phase.

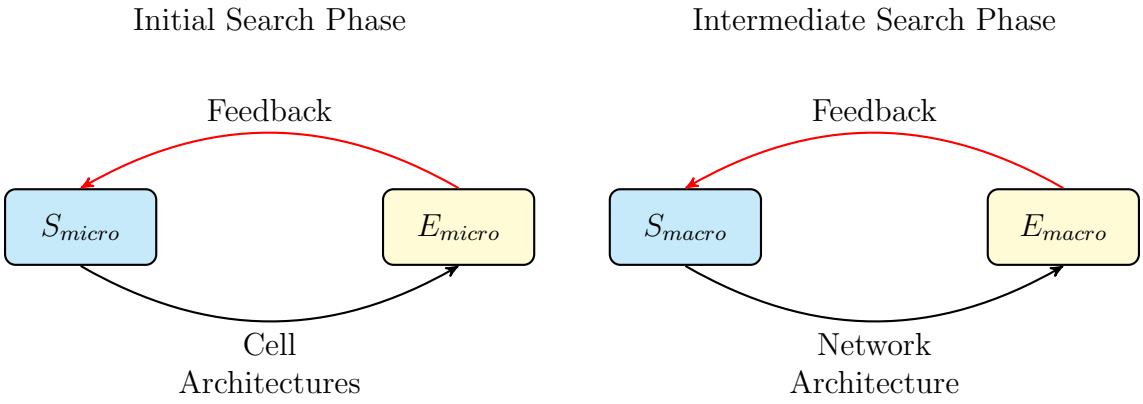


Figure 6.1: Hierarchical ICDARTS optimizes network architectures at both the cell and network-levels by introducing an intermediate search phase and search S_{macro} and evaluation E_{macro} networks. Cell structures in the micro level search phase are first optimized by leveraging ICDARTS to cyclically optimize the search S_{micro} and evaluation E_{micro} networks. Then, in the intermediate search phase, the template of the evaluation network is searched by using a modified version of ICDARTS (see Algorithm 4) to optimize connections within a large search graph S_{macro}, discretizing them at each iteration to form the evaluation network E_{macro}.

Table 6.1: ICDARTS Hierarchical Search Version 1 CIFAR-10 Retraining Test Accuracies and Inference Latencies

Network	**Retraining Accuracy (%)**	**Inference Latency (s/batch)**
Initial Evaluation	**96.81 (0.20)**	**0.11 (0.01)**
Intermediate Evaluation	96.60 (0.24)	0.12 (0.00)

Table 6.2: ICDARTS Hierarchical Search Version 2 CIFAR-10 Retraining Test Accuracies and Inference Latencies

Network	**Retraining Accuracy (%)**	**Inference Latency (s/batch)**
Initial Evaluation	95.85 (0.30)	0.13 (0.01)
Intermediate Evaluation	**95.92 (0.32)**	0.13 (0.01)

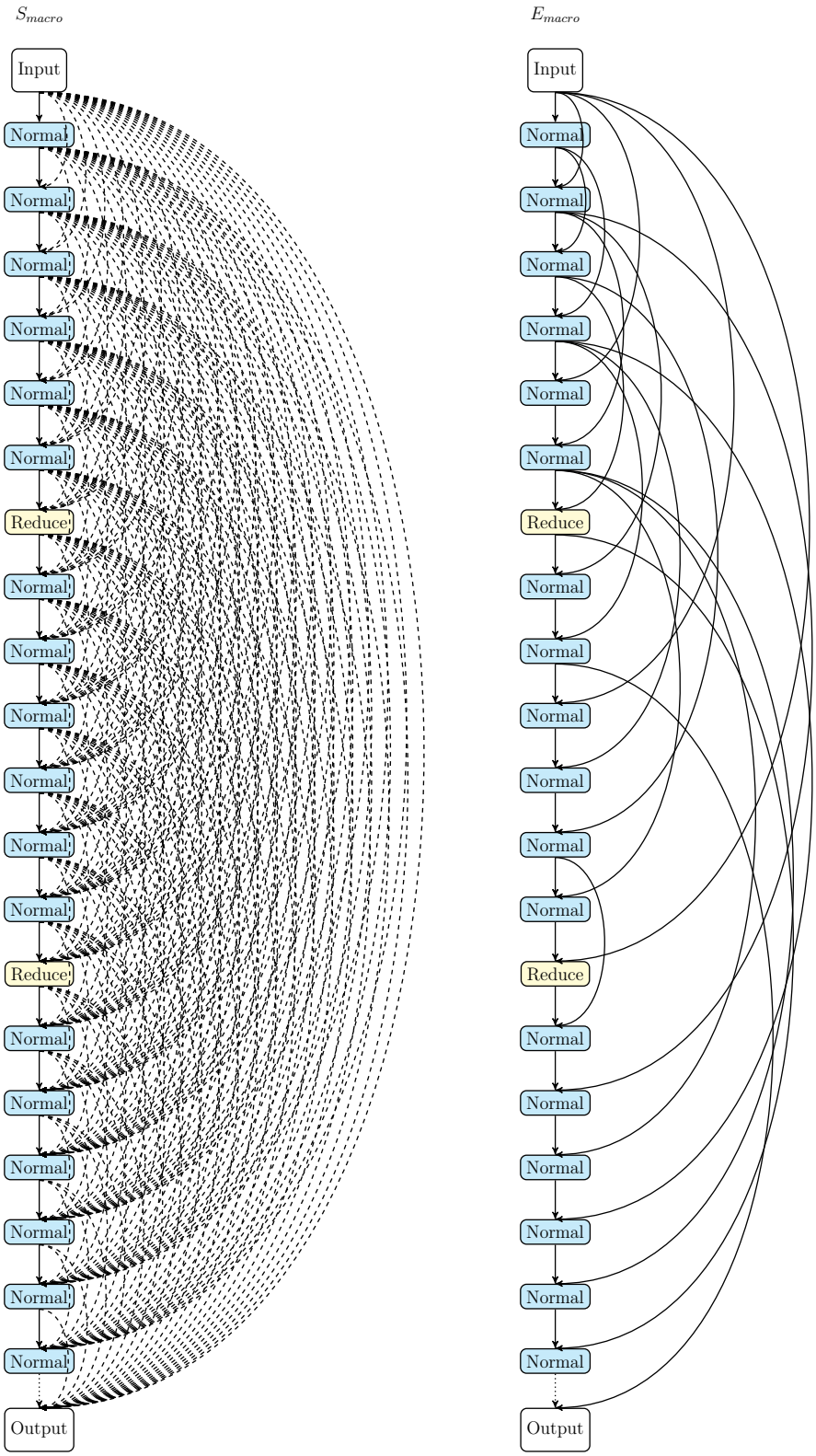


Figure 6.2: Hierarchical ICDARTS Version 1 Intermediate Search Phase Search Network, S_{macro} (left), and Example Evaluation Network, E_{macro} (right).

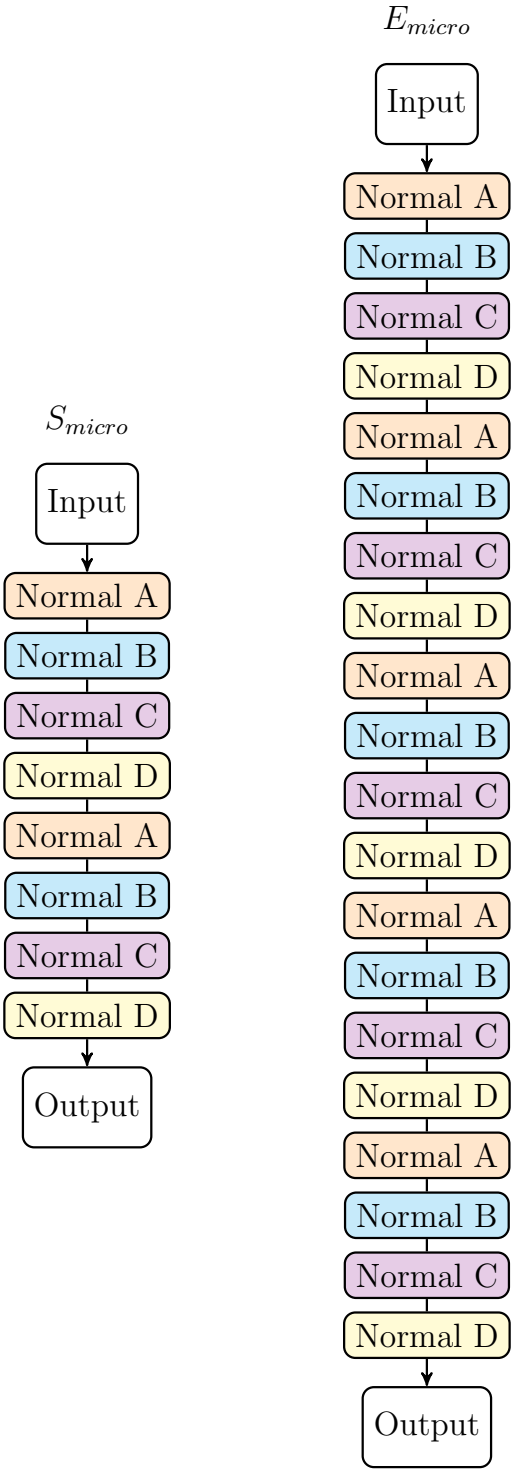


Figure 6.3: ICDARTS Hierarchical Search Version 2 Initial Search Phase Search Network, S_{micro}, (left) and Evaluation Network, E_{micro}, (right).

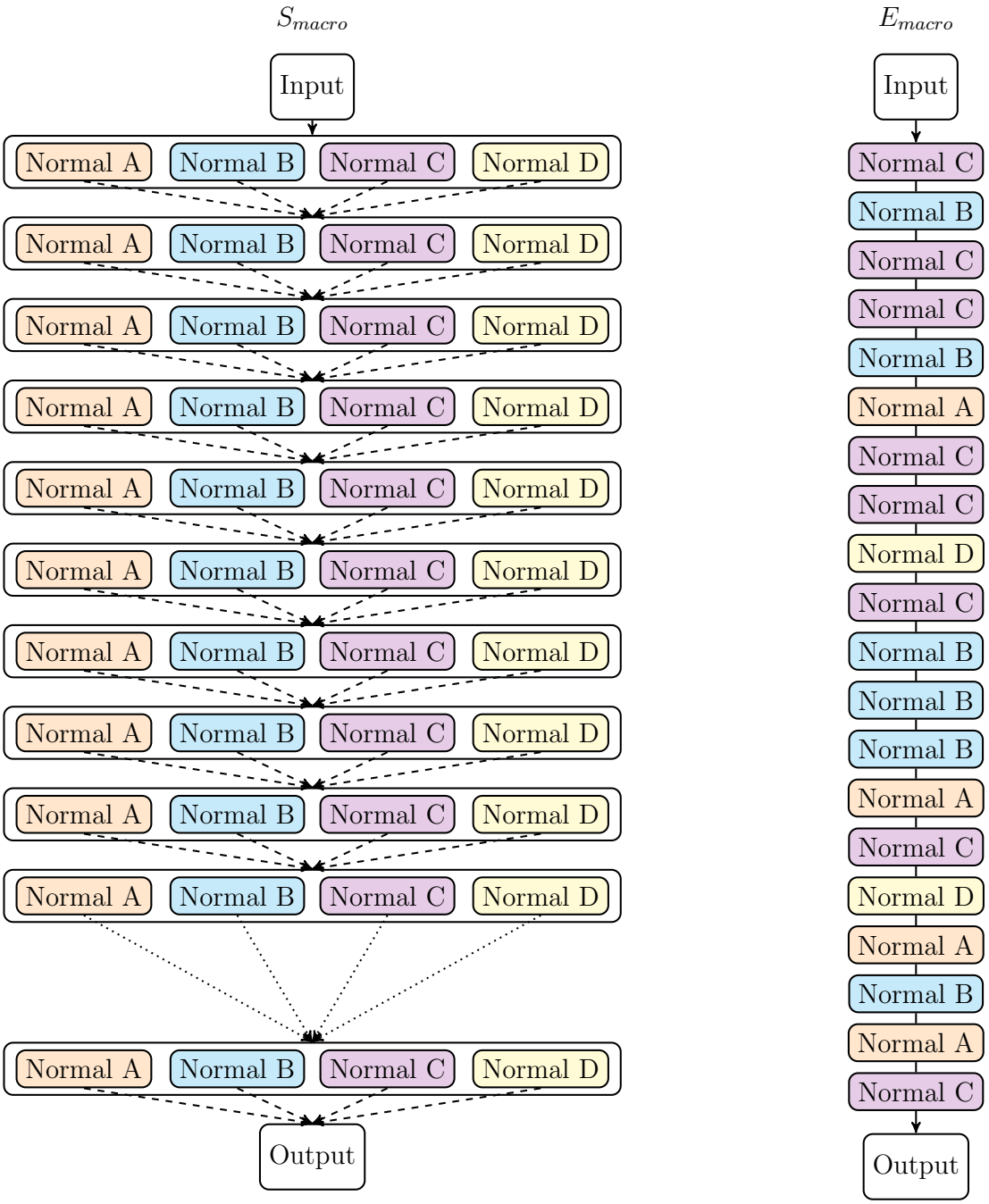


Figure 6.4: ICDARTS Hierarchical Search Version 2 Intermediate Search Phase Search Network, S_{macro}, (left) and Evaluation Network, E_{macro}, (right).

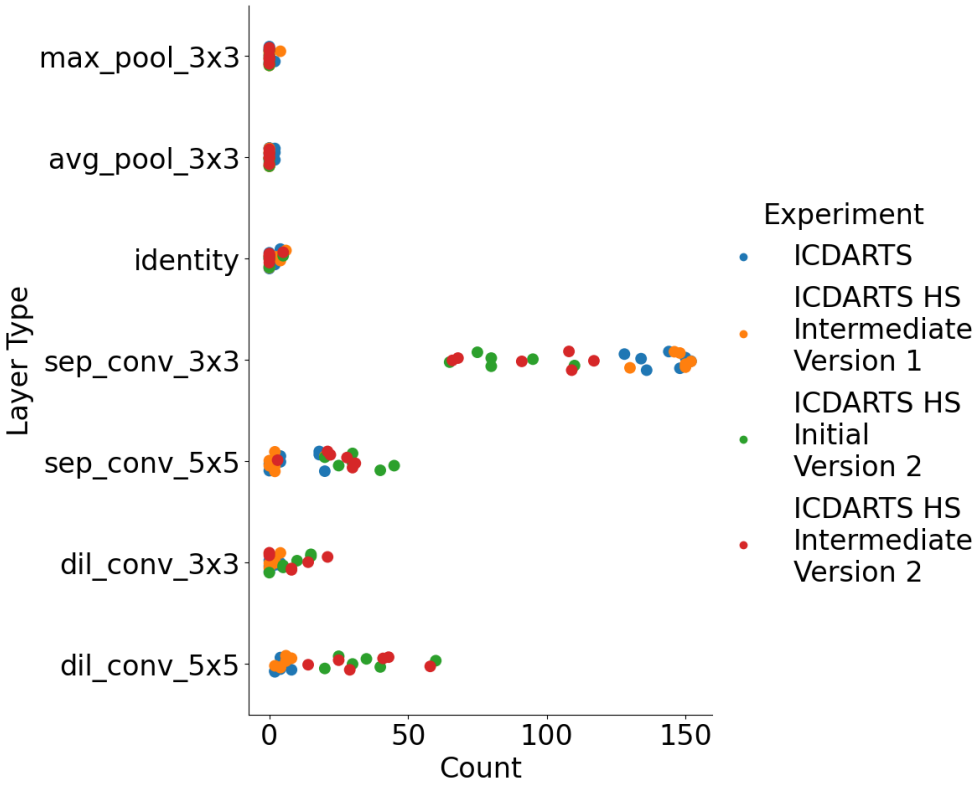


Figure 6.5: Hierarchical Search Per Network Layer Frequencies

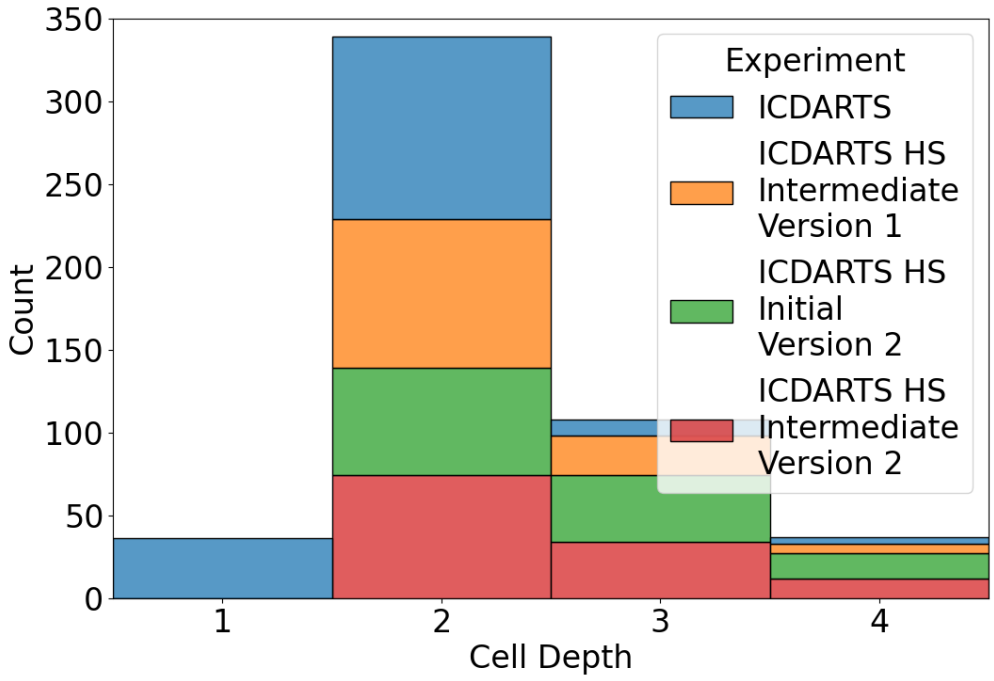


Figure 6.6: Hierarchical Search Per Network Cell Depths

Chapter 7

Summary and Future Work

This work proposed a set of improvements to the stability and performance of an existing gradient-based NAS method. The first half of this chapter summarizes each iteration of improvements, along with the results and conclusions drawn from the experimental results of each. The closing half of the chapter discusses steps that can be taken in the future to improve and expand upon this work, including extending it to alternative tasks and architectures.

7.1 Summary

Chapter 3 introduced a set of algorithmic changes to the CDARTS algorithm. The algorithm was first modified to ensure that the evaluation network of the algorithm's search phase was a better proxy for the network evaluated during the retraining phase by reformulating the network's loss function to no longer rely on a term unavailable during the retraining phase. The soft-target cross-entropy loss term, which quantified the distance between the predictions of the search and evaluation networks, was also shifted to the loss function used for updating the search network's weights. This change allowed for the flow of performance-based feedback from the evaluation network to the search network. The final change ensured that the search phase networks were trained on the same data to eliminate bias and support the objective of the networks producing the same outputs. In addition to those algorithmic improvements, different configurations of its operation set were tested to evaluate the effect of the *zero*

operation that was included in the operation set of the CDARTS search network but not of its evaluation network. Experiments showed that the algorithmic changes to CDARTS did result in improvements to the algorithm's stability and consistency and that the *zero* operation could be removed entirely from the operation set to achieve optimal generalization and stability results when combined with the algorithmic changes. Ablation studies further supported the effectiveness of the algorithmic changes and provided valuable information regarding the importance of elements of the templates of the search and evaluation networks.

Chapter 4 presented the next phase of improvements to the CDARTS algorithm: expansions to its search phase. After testing the impact of changing the number of nodes per cell and not finding a significant improvement by changing this variable alone, changes were made to ICDARTS' cell discretization protocol that expanded the space of discoverable cells. Although this approach significantly improved the performance of the networks it discovered, it was computationally expensive. To address this drawback, a multi-objective version of ICDARTS incorporating a penalty term that estimates the latency of the evaluation network given the learned connection weights and a set of normalized expected latencies corresponding to each operation choice into its original loss function was developed. Experiments assessing the effectiveness of the new multi-objective loss function weighted with different coefficients found that the penalty term could successfully reduce the inference latency of ICDARTS networks. Additional experiments evaluated the impact of alternative operation spaces, ranging from spaces consisting of the most elementary operations to more complex state-of-the-art mobile convolution blocks, on the performance of ICDARTS networks. The results revealed that an operation space composed of simpler operations included in earlier NAS operation spaces, such as NASNet, produced networks with the best overall performance. In order to combine all the operation sets into one large operation space and efficiently apply ICDARTS to this master operation space, a novel approach for incorporating dynamic operation spaces into ICDARTS was implemented. The experiments on this approach revealed that it produced high-performing and consistent networks.

Chapter 5 shows the results of applying the best overall improvements to ICDARTS to two scientific datasets: the Strong Gravitational Lens Finding Challenge (SGLFC) 2.0 and

a dataset of electron microscopy images. The results on SGLFC and EM verify that the proposed methods produce similar outcomes to those produced by CIFAR-10.

Finally, Chapter 6 proposes an expansion of ICDARTS that allows for its template to be optimized in addition to its cell structures. This expansion, referred to in this work as hierarchical ICDARTS, adds an intermediate search phase between the original search and retraining phase, in which the template of the evaluation network is optimized given the cell structures optimized in the initial search phase. Two configurations of this template search phase were implemented. The first configuration optimized the skip connections between each cell, and the second optimized the choice of cell at each layer evaluation network. The results showed that only the second configuration produced networks with improved performances over the default evaluation networks produced by its initial search phase.

7.2 Future Work

The next steps of this work will involve combining all of the improvements to ICDARTS into one and evaluating the results, improving and expanding upon the hierarchical versions of ICDARTS, and adapting ICDARTS to global search spaces and tasks beyond image classification. As discussed at the close of Chapter 6, the proposed hierarchical ICDARTS solution requires additional exploration, particularly the approach that searches for optimal skip connections. The hierarchical ICDARTS approach that showed more promise optimized the choice of cell type at each depth and should be further evaluated on different and larger operation spaces and combined with the additional improvements presented in Chapter 5. Improvements that enable the optimization of additional parameters, including the placement of reduce cells, channel sizes at each layer, and network depth, remain to be developed. The ultimate goal will be to combine the intermediate search networks of each configuration into one master operation space that enables the optimization of all components of the evaluation network template.

Additionally, a global, proxyless version of ICDARTS could be implemented using the methods presented in this work. Efficient, proxyless global NAS algorithms such as MobileNet Howard et al. (2017); Sandler et al. (2018) and the algorithms of the DNAS

family Wu et al. (2018); Wan et al. (2020); Wu et al. (2021) are beneficial, despite their high compute costs, since they both ensure a high degree of flexibility in the space of architectures that can be searched and can ensure that the architecture discovered in the search phase transfers well to the target task and hardware. A proxyless, global version of ICDARTS could draw inspiration from existing literature and search candidate architectures from within a layer-wise super-network similar to the ones searched in DNAS and DMaskingNAS. Skip-connects could also be introduced into this search space to introduce functionality absent in previous literature, including FBNet and FBNetV2, and a latency penalty term could be similarly incorporated into this algorithm for optimal efficiency.

Furthermore, the improvements to ICDARTS introduced in this work will be augmented to be applied to problems beyond image classification, including semantic segmentation and NLP tasks. To adapt ICDARTS to semantic segmentation tasks, search spaces that reflect those of NAS-UNet Weng et al. (2019), Auto-DeepLab Liu et al. (2019a), SqueezeNAS Shaw et al. (2019), and FBNetV5 Wu et al. (2021) should be evaluated. Successful application of this approach to semantic segmentation will likely require adapting the method to search for separate upsampling and downsampling cells and skip connections between them as in Weng et al. (2019).

The NAS methods presented in this work only accommodate convolutional neural networks. However, a growing field of literature has emerged in which attention Transformer architectures, sometimes hybridized with convolutional networks, have succeeded in NLP and vision tasks. To adapt the contributions of this work to accommodate NLP tasks such as sequence translation, ICDARTS could be adapted to search for the Transformer model's encoder and decoder cells. Existing NLP Transformer NAS implementations optimize architectures at varying grains of complexity Wang et al. (2020a); Xu et al. (2021); So et al. (2019). Fine-grained search spaces that optimize encoders and decoders could take inspiration from the Evolved Transformer So et al. (2019). In addition to the layer options already part of ICDARTS, the search space could be expanded to include multi-head attention operations, gated linear units, and attention encoder layers. The coarse-grained versions of the search space could be similar to those of HAT and NASBERT Wang et al.

(2020a); Xu et al. (2021), in which block hyperparameters such as embedding dimensions, number of multi-head attention heads, MLP ratios, and Q-K-V dimensions are optimized.

Motivated by the observation that vision Transformer models currently rank high among state-of-the-art deep learning models for vision tasks, the NLP transformer search space findings could also be extended to search for vision Transformer architectures. The vision Transformer NAS approaches could resemble those used in the NLP transformer case and draw additional inspiration from the vision transformer search protocols described in Chen et al. (2021a). Search spaces that combine convolutions with Transformer models using methods similar to those described in Dai et al. (2021); Li et al. (2021a); Gong et al. (2022) might also be explored.

Bibliography

Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. In *IEEE transactions on neural networks, 5(1)*, pages 54–65. IEEE. 10

Baker, B., Gupta, O., Naik, N., and Raskar, R. (2016). Designing neural network architectures using reinforcement learning. 11

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. (2018). Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR. 12

Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. (2017). Efficient architecture search by network transformation. 12, 13

Cai, H., Zhu, L., and Han, S. (2019). ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*. 14, 17

Chen, M., Peng, H., Fu, J., and Ling, H. (2021a). Autoformer: Searching transformers for visual recognition. 95

Chen, X., Xie, L., Wu, J., and Tian, Q. (2019). Progressive darts: Bridging the optimization gap for nas in the wild. 7, 14, 15

Chen, X., Xie, L., Wu, J., and Tian, Q. (2021b). Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129(3):638–655. 30

Chu, X., Zhang, B., Ma, H., Xu, R., and Li, Q. (2019). Fast, accurate and lightweight super-resolution with neural architecture search. 20

Dai, Z., Liu, H., Le, Q. V., and Tan, M. (2021). Coatnet: Marrying convolution and attention for all data sizes. 95

DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. 36

Dong, J.-D., Cheng, A.-C., Juan, D.-C., Wei, W., and Sun, M. (2018). Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. 12

Dong, X. and Yang, Y. (2019). Searching for a robust neural architecture in four gpu hours. 15, 16

Elsken, T., Metzen, J. H., and Hutter, F. (2018). Efficient multi-objective neural architecture search via lamarckian evolution. In *International Conference on Learning Representations*. 10

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017. 10, 11, 14

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. 6

Gong, C., Wang, D., Li, M., Chen, X., Yan, Z., Tian, Y., qiang liu, and Chandra, V. (2022). NASVit: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training. In *International Conference on Learning Representations*. 95

He, K., Gkioxari, G., Dollar, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 6

Herron, E., Young, S., and Rose, D. (2022a). Icdarts: Improving the stability of cyclic darts. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. 23

Herron, E., Young, S. R., and Rose, D. (2022b). Icdarts: Improving the stability of cyclic darts. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1055–1062. IEEE. 30

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. 93

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q. (2016). Densely connected convolutional networks. 20, 83

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In Coello, C. A. C., editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg. Springer Berlin Heidelberg. 11

Jiang, Y., Hu, C., Xiao, T., Zhang, C., and Zhu, J. (2019). Improved differentiable architecture search for language modeling and named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3585–3590, Hong Kong, China. Association for Computational Linguistics. 15, 16, 51

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR (Poster)*. 32

Li, C., Tang, T., Wang, G., Peng, J., Wang, B., Liang, X., and Chang, X. (2021a). Bossnas: Exploring hybrid cnn-transformers with block-wisely self-supervised neural architecture search. 21, 95

Li, Y., Wen, Z., Wang, Y., and Xu, C. (2021b). One-shot graph neural architecture search with dynamic search space. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):8510–8517. 48

Liu, C., Chen, L.-C., Schroff, F., Adam, H., Hua, W., Yuille, A., and Fei-Fei, L. (2019a). Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. 21, 94

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. (2018a). Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34. 25, 26, 36

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search. 20

Liu, H., Simonyan, K., and Yang, Y. (2018b). Darts: Differentiable architecture search. 7, 12

Liu, H., Simonyan, K., and Yang, Y. (2019b). DARTS: Differentiable architecture search. In *International Conference on Learning Representations*. 14, 25, 26, 30, 32, 36

Lucchi, A., Li, Y., and Fua, P. (2013). Learning for structured prediction using approximate subgradient descent with working sets. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1987–1994. 70

Metcalf, R. B. (2019). 66

Metcalf, R. B., Meneghetti, M., Avestruz, C., Bellagamba, F., Bom, C. R., Bertin, E., Cabanac, R., Courbin, F., Davies, A., Decenciè re, E., Flamary, R., Gavazzi, R., Geiger, M., Hartley, P., Huertas-Company, M., Jackson, N., Jacobs, C., Jullo, E., Kneib, J.-P., Koopmans, L. V. E., Lanusse, F., Li, C.-L., Ma, Q., Makler, M., Li, N., Lightman, M., Petrillo, C. E., Serjeant, S., Schäfer, C., Sonnenfeld, A., Tagore, A., Tortora, C., Tuccillo, D., Valentín, M. B., Velasco-Forero, S., Kleijn, G. A. V., and Vernardos, G. (2019). The strong gravitational lens finding challenge. *Astronomy &amp Astrophysics*, 625:A119. 64

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384. 7, 10

Negrinho, R. and Gordon, G. (2017). Deeparchitect: Automatically designing and training deep architectures. 11

Nekrasov, V., Chen, H., Shen, C., and Reid, I. (2019). Fast neural architecture search of compact semantic segmentation models via auxiliary cells. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9126–9135. 6

O'Neill, D., Xue, B., and Zhang, M. (2021). Evolutionary neural architecture search for high-dimensional skip-connection structures on densenet style networks. *IEEE Transactions on Evolutionary Computation*, 25(6):1118–1132. 21, 83

Pan, J., Shapiro, J., Wohlwend, J., Han, K. J., Lei, T., and Ma, T. (2020). Asapp-asr: Multistream cnn and self-attentive sru for sota speech recognition. 6

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR. 7, 12, 13, 36

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2018). Regularized evolution for image classifier architecture search. 10

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789. 10, 11, 12, 14, 26

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q., and Kurakin, A. (2017). Large-scale evolution of image classifiers. 10

Redmon, J. and Farhadi, A. (2017). Yolo9000: Better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 6

Ren, P., Xiao, Y., Chang, X., Huang, P.-Y., Li, Z., Chen, X., and Wang, X. (2020). A comprehensive survey of neural architecture search: Challenges and solutions. 11

Saito, S. and Shirakawa, S. (2019). Controlling model complexity in probabilistic model-based dynamic optimization of neural network structures. In *Lecture Notes in Computer Science*, pages 393–405. Springer International Publishing. 18

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520. 18, 46, 93

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175. 11

Shaw, A. E., Hunter, D., Iandola, F. N., and Sidhu, S. (2019). Squeezenas: Fast neural architecture search for faster semantic segmentation. *CoRR*, abs/1908.01748. 48, 94

Shin*, R., Packer*, C., and Song, D. (2018). Differentiable neural network architecture search. 13

So, D. R., Liang, C., and Le, Q. V. (2019). The evolved transformer. 94

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212. 10

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127. 10

Suganuma, M., Shirakawa, S., and Nagao, T. (2017). A genetic programming approach to designing convolutional neural network architectures. 10

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828. 18, 54

Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR. 46

Tan, M. and Le, Q. (2021). Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, pages 10096–10106. PMLR. 6, 46

Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., Vajda, P., and Gonzalez, J. E. (2020). Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. 19, 94

Wang, H., Wu, Z., Liu, Z., Cai, H., Zhu, L., Gan, C., and Han, S. (2020a). Hat: Hardware-aware transformers for efficient natural language processing. 94

Wang, N., Gao, Y., Chen, H., Wang, P., Tian, Z., Shen, C., and Zhang, Y. (2020b). Nas-fcos: Fast neural architecture search for object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 6

Wang, Y., Yang, Y., Chen, Y., Bai, J., Zhang, C., Su, G., Kou, X., Tong, Y., Yang, M., and Zhou, L. (2019). Textnas: A neural architecture search space tailored for text representation. 17

Weng, Y., Zhou, T., Li, Y., and Qiu, X. (2019). Nas-unet: Neural architecture search for medical image segmentation. *IEEE Access*, 7:44247–44257. 94

Wistuba, M., Rawat, A., and Pedapati, T. (2019). A survey on neural architecture search. 9, 11

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. (2018). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. 18, 54, 94

Wu, B., Li, C., Zhang, H., Dai, X., Zhang, P., Yu, M., Wang, J., Lin, Y., and Vajda, P. (2021). Fbnetv5: Neural architecture search for multiple tasks in one run. 94

Xie, L. and Yuille, A. (2017a). Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388. 10

Xie, L. and Yuille, A. (2017b). Genetic cnn. 11

Xie, S., Zheng, H., Liu, C., and Lin, L. (2019). SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*. 15

Xu, J., Tan, X., Luo, R., Song, K., Li, J., Qin, T., and Liu, T.-Y. (2021). NAS-BERT. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery &amp Data Mining*. ACM. 94, 95

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. (2020). Pc-darts: Partial channel connections for memory-efficient architecture search. 25

Yan, Z., Li, X., Li, M., Zuo, W., and Shan, S. (2018). Shift-net: Image inpainting via deep feature rearrangement. 18

Yu, H. and Peng, H. (2020). Cyclic differentiable architecture search. 26, 28, 30, 32, 36

Yu, H., Peng, H., Huang, Y., Fu, J., Du, H., Wang, L., and Ling, H. (2020). Cyclic differentiable architecture search. 7, 14, 15

Zhang, H., Li, Y., Chen, H., and Shen, C. (2019). Memory-efficient hierarchical neural architecture search for image denoising. 20

Zhong, Z., Yang, Z., Deng, B., Yan, J., Wu, W., Shao, J., and Liu, C.-L. (2018). Blockqnn: Efficient block-wise neural network architecture generation. 12

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. 7, 10

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. 10

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2017). Learning transferable architectures for scalable image recognition. 2, 7, 11, 12, 46

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710. 14, 25, 26, 36

Vita

Emily Herron holds a Ph.D. in Data Science and Engineering from the University of Tennessee, Knoxville. Her dissertation research involved developing stable, high-performing, generalized differentiable neural architecture search methods, and her Ph.D. advisor was Dr. Steven Young. While completing her Ph.D., Herron served as a Bredesen Center Fellow and Graduate Research Assistant in the Computing and Computational Science Directorate's Learning Systems group, in which she contributed to research on evolutionary algorithms, neural network ensembles, text mining, and NLP models and co-authored a successful proposal for 20,000 Summit hours awarded by the Oak Ridge Leadership Computing Facility's Director's Discretionary Program. Herron is a member of ACM and IEEE and has served as a reviewer for the International Conference on Machine Learning (ICML) and as a peer mentor. She also has a B.S. in Computational Science from Mercer University.