Department of Precision and Microsystems Engineering

Improve the Optical Neural Network structure by using Neural Architecture Search for Visual Classification Tasks

Tianyang Hu

Report no : 2023.037
Coach : Dr. R.A. Norte, Dr. M.A. Bessa
Specialisation : MSD/ED
Type of report : MSc Thesis
Date : 16 June 2023

Improve the Optical Neural Network structure by using Neural Architecture Search for Visual Classification Tasks

to obtain the degree of Master of Science
at Delft university of Technology,
Faculty of Mechanical, Maritime and Materials Engineering
to be defended publicly on July 7th 2023.

by

Tianyang Hu

Delft University of Technology, Delft, Netherlands

Supervisors:

Dr. M.A. Bessa,	Brown University, School of Engineering
Dr. R.A. Norte,	TU Delft, 3mE Faculty, PME

Committee member:

Dr. A.M. Aragon,	TU Delft, 3mE Faculty, PME

Project Duration: July 2022 - July 2023

Keywords: Neural Architecture Search, Optical Neural Networks, Mach-Zehnder interferometer

CONTENTS

SUMMARY

This thesis explores the integration of Neural Architecture Search (NAS) with Optical Neural Networks (ONNs) to optimize the efficiency and performance of ONNs in traditional visual image classification tasks. The study introduces a new approach that applies NAS, a technique traditionally used to optimize the performance of Artificial Neural Networks (ANNs), to the field of ONNs.

The primary goal of the research is to apply NAS to ONNs, predicated on the shared similarities between ONN and ANN layers, despite ignoring the physical manufacturing limitations. The integration's feasibility is established both theoretically and through independent NAS and ONN studies. The findings indicate that by tuning the specific ONN structures, similar to ANNs, can significantly influence ONN performance, thereby validating the possibility of NAS and ONN integration.

In this study, we develop a cell-based search space explicitly designed for ONNs, inspired by the NASBench201, and integrated with the DARTS architecture search strategy. This combination demonstrated improved ONN performance, particularly in image classification tasks and significantly improves the efficiency of finding the optimal result.

Comparative performance analysis revealed that the ONN model generated using the proposed algorithm outperformed a simple ONN model by approximately 2%, proving the effectiveness of the NAS-ONN integration. The study also examined the properties of ONNs and compared them with ANNs, and explored the similarity of their optimal structure, both of which exhibited their unique characteristics and performance. Based on the findings we try to apply some ANN structures with superior performance into ONNs, and the results obtained confirm our main conclusions.

In conclusion, this research successfully enhances the performance of ONNs through the integration of NAS, establishing the potential of applying high-performance ANN structures to ONNs. The results set the foundation for future exploration in the optimization of ONNs, promising significant contributions to the field of next-generation artificial intelligence computing.

1

INTRODUCTION

THE objective of this study is to enhance the efficiency of Optical Neural Networks (ONN) using the Neural Architecture Search (NAS) technique. To achieve this goal, the prevailing method of searching Artificial Neural Networks (ANN) is adopted and employed for Optical Neural Networks. The primary focus is to optimize the performance of Optical Neural Networks by utilizing the Neural Architecture Search algorithm, which facilitates the exploration of the most suitable architecture for Optical Neural Networks. The proposed approach is expected to contribute to the current Optical Neural Networks implementation methods by improving their efficiency and accuracy.

1.1. PURPOSE, GOALS AND THEIR JUSTIFICATIIONS

Optical Neural Network is a potential candidate for next-generation artificial intelligence computing due to its impressive energy efficiency and computing speed.[1] Currently, most Optical Neural Network designs lack scalability. Previous designs are limited to benchtop setups or parts integration [2]. Therefore, one possible solution is first to regard ONN as ANN and use an automatize strategy to search for a optimal structure, then consider physical and manufacturing constraints to obtain a feasible optimal structure. Neural Architecture Search (NAS) automates the search for the best structure for constrained neural networks, so this work uses NAS to achieve our goal of finding the best ONN structure.

1.2. PROPOSED METHODS AND THEORETICAL BASICS

In this study, physical principles and restrictions are discussed and partially ignored during the whole progress. However, to facilitate the subsequent study of the introduction of physical constraints to adjust the automatic search. Hence, the search space should be efficient and introduce more information into the network. NASbench201 [3] can achieve mentioned goals. Provide extra diagnostic information to help us analyze. NASbench201 [3] is a search space inspired by cell-based algorithms. Compared to the

original search space used in DARTS [4] and other search spaces with similar functions (like NATS-bench [5], etc.), NASbench201 [3] has fewer operational candidates [4][3] and unique architectures [5], which can improve the efficiency of the computation. During the iteration, this work will modify our search space based on NASbench201, making the search space more suitable for our work while ensuring the above advantages. The search strategy is another important part of NAS. DARTS, a differentiable NAS strategy, has competitive performance and efficiency compared to the discrete strategies [4, 6]. It can be explained in detail without considering as a black box [6] since both DARTS and NASbench use directed acyclic graphs to represent each cell structure [3, 4, 6] to make it analyzable. The high analyzability and efficiency of DARTS enable iterable updates to be more effective and accurate. This work also adopts the same performance estimation strategy as the one used in DARTS [4] to ensure compatibility with the previous two parts.

As for the Optical Neural Network part, the simulation provided from [7] is taken into consideration. The author provides a library integrated with the simulation of several different types of Optical Neural Networks, including the simulation of the Mach-Zehnder Interferometer (MZI) type Optical Neural Network. Only MZI-type Optical Neural Networks will be discussed in this thesis report. Besides, using this library can allow switching to other types of ONN since similar APIs are used in the library for different ONN types of ONNs.

Based on these two parts, a Neural Architecture Search specific for searching Optical Neural Networks can be designed to discover the potential possibilities to extend the usage of NAS to find better performance ONN structure under a specific physical implementation by following the output from the NAS to change the stack of physical structures.

1.3. OUTLINE

This work initiates with a comprehensive literature review presented in Chapter 2, which provides an in-depth analysis of the essential concepts and features of various Optical Neural Networks (ONNs) and the fundamentals of different types of neural architecture search (NAS) for each of its components.

Chapter 3 introduces the original research conducted in this thesis. It starts with a novel proposed solution that combines both NAS and ONN techniques, with a preliminary conclusion based on the literature review. Subsequently, it presents several basic experiments on NAS (DARTS+NASBench201 on NASLib) and simulations of ONNs (MZI-ONN on Torch-ONN) separately to investigate their unique properties, followed by a thorough analysis of the feasibility of integrating these two aspects. This is followed by a section where NAS and ONN are combined. Initially, the theoretical foundation of this combination is presented, then a new search space specifically designed for ONNs is introduced along with its integration with the currently available search strategy.

Chapter 4 provides an in-depth discussion and a detailed analysis and comparison of the results for the combination of NAS and ONN from the previous methodology sec-

tion. The final chapter focuses on the conclusion and recommendations, presenting a comprehensive summary of the main findings and suggestions for various directions of future research. The study concludes with supplementary information and appendices to offer additional context and details for our conclusion.

2

LITERATURE REVIEW

RECENTLY, Optical neural networks (ONNs) have drawn the attention of researchers since ONNs provide a potential solution to resolve high energy costs in traditional artificial neural networks (ANNs) [1]. In this research, Neural Architecture Search (NAS), a state-of-the-art technique in the computer science field, will be applied to optimize currently available Optical Neural Networks (ONNs) structures to reach higher accuracy in image classification tasks. (such as CIFAR10/MNIST) First, optical neural networks will be introduced and some popular implementation will be introduced in the first section. Then in the second section, the neural architecture search will be introduced in general and some techniques/tricks that might be used will also be introduced.

2.1. OPTICAL NEURAL NETWORK

2.1.1. THE ADVANTAGES OF OPTICAL NEURAL NETWORKS

Optical neural networks provide a potential means to solve the energy-cost problem faced by deep learning.[1] ONN has several advantages, including a noticeably faster computational speed since electronic neural networks are restricted by clock rate (for GPUs mostly less than 3GHz [2]), compared to the ONN model described in [8] can reach 100 GHz photo-detection rate. And this type of computing is not constrained by sequential instruction execution and memory access limitations. [9] Also, ONNs have significantly lower energy consumption than traditional electronic ones. The author [8] thinks it only requires about 10mW energy to maintain one-phase modulator settings, which is the key component in MZI (Mach-Zehnder interferometer) type ONN. An estimation of the energy efficiency given in [8] for MZI-ONN shows the energy per FLOP (A key efficiency indicator) is 3 orders to 5 orders better than the conventional GPUs calculation. All these advantages benefit from the ultra-high bandwidth, low latency, and small energy consumption of the physical properties of optics devices.

2.1.2. SOME TYPICAL TYPES OF OPTICAL NEURAL NETWORKS

Since different ONNs have completely different implementations, here implementation of some types of ONNs will be introduced. The focus will mainly be on on-chip ONNs such as PDNN and MZI, but classic types like Diffractive Deep Neural Networks will also be discussed in addition to on-chip solutions. The on-chip design is preferred because the on-chip design is more suitable for the actual application scenario and more sustainable.

2.1.3. MACH-ZEHNDER INTERFEROMETER OPTICAL NEURAL NETWORK (MZI-ONN)

The author [8] uses the Mach-Zehnder interferometer (MZI) to create a triangular photonic circuit in order to realize the matrix and its multiplication by using optics.

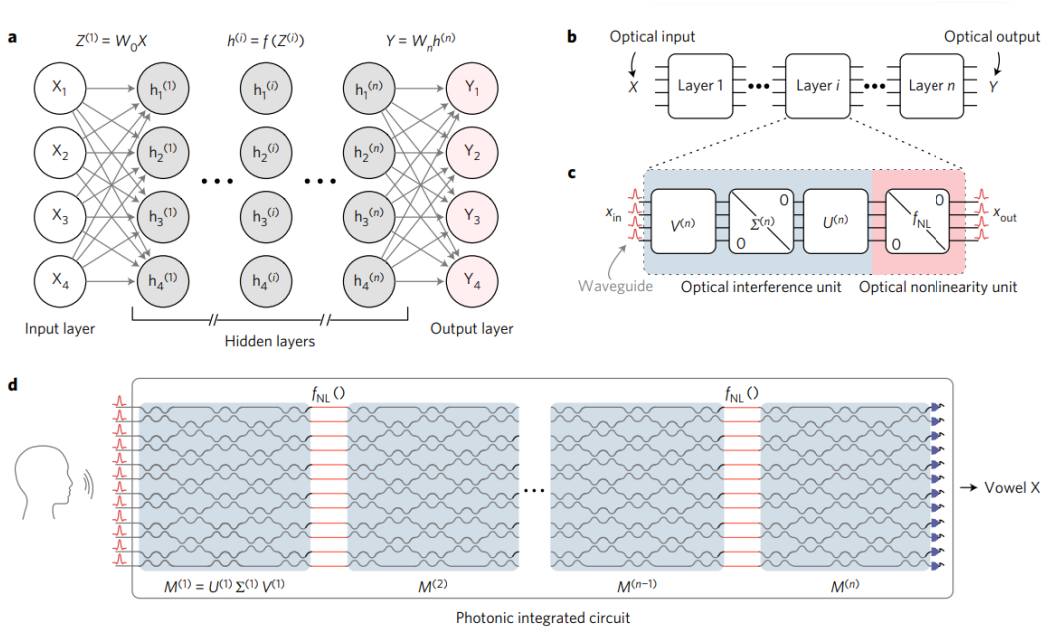


Figure 2.1: General architecture of the MZI-ONN. a, The general architecture of an artificial neural network consists of an input layer, several hidden layers, and an output layer. b, The general neural network is decomposed into individual layers. c, Each layer of the artificial neural network consists of optical interference and nonlinearity units. d, An all-optical, fully integrated neural network is proposed for a vowel classification task. [8]

The architecture of the Mach-Zehnder Interferometer Optical Neural Network (MZI-ONN) is composed of two elements: the optical input unit (OIU) and the optical nonlinearity unit (ONU), as shown in Figure 2.1c. The OIU is responsible for performing the optical matrix multiplication required by the MZI-ONN. On the other hand, the ONU utilizes common optical nonlinearities like saturable absorption and bistability to carry out the nonlinear activation function. The physical implementation and design of the

MZI-ONN can be found in [10], while [8] presents a detailed description of its application in vowel classification. Thermo-optic phase shifters within the OIU may be appropriately configured once the weight matrices in the Multilayer Perceptron (MLP) have been trained and deconstructed. This ONN is entirely passive as the weight matrices are set after training, which reduces energy consumption [11]. Additionally, an extended convolutional layer based on this technique is available in [7], which has also been employed in image classification tasks. The author in [12] also introduced an optical CNN model, which demonstrated the prospects for the application of optical CNN.

According to the author [8], there are several limitations associated with the proposed method. Firstly, the resolution of ONNs is constrained by practical non-idealities, which restricts their performance. Secondly, in the case of this particular ONN, signal preprocessing is necessary to encode the amplitude of optical pulses. However, this creates a trade-off between encoding error and photodetector noise. Lastly, further research [11] has indicated that the MZI-ONN may not be an efficient use of space since it relies on SVD-based techniques that require high photonic component utilization.

2.1.4. PHOTONIC DEEP NEURAL NETWORK (PDNN)

The author [2] proposed a novel approach to address the scalability issues of on-chip optical neural networks (ONNs) by introducing a photonic deep neural network (PDNN). The proposed PDNN consists of a SiGe photodetector integrated with optical PIN attenuators to construct the linear computation component (Fig.2.2d). Additionally, a transimpedance amplifier drive and a microring modulator were utilized to construct the nonlinear activation (RELU) component of the PDNN (Fig.2.2e). Together, these components were integrated to implement a photonic-electronic neuron, thus providing an innovative solution to the challenges of on-chip ONNs. [2]

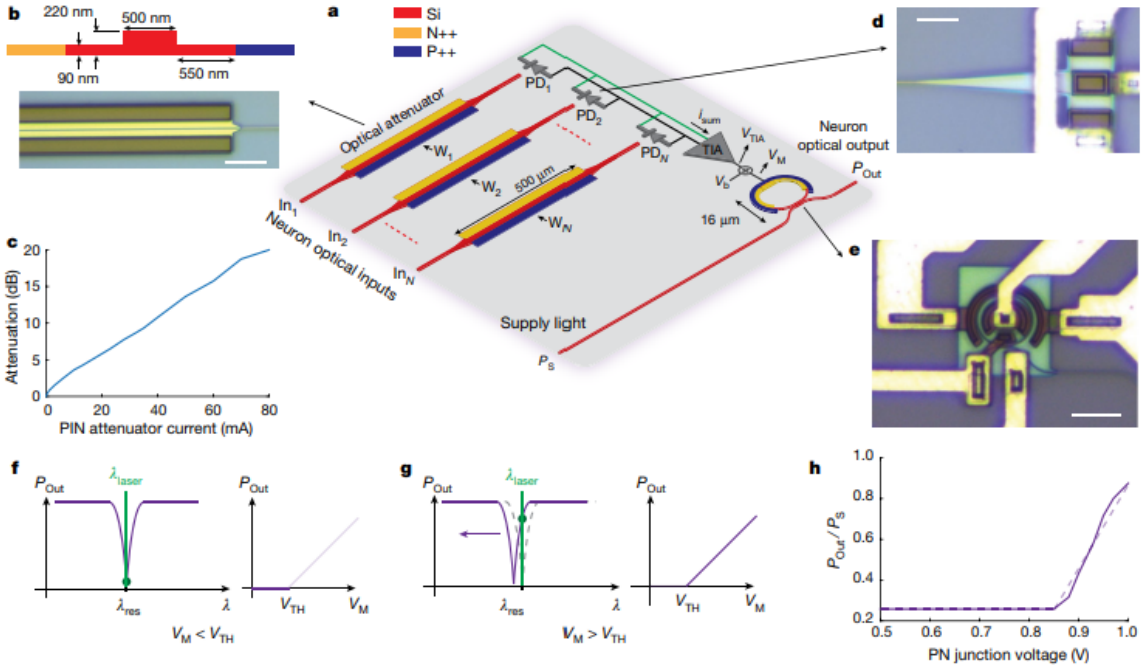


Figure 2.2: Photonic–electronic neuron implementation. a, This schematic shows an on-chip photonic-electronic neuron that has N optical inputs and one optical output. The neuron is implemented using various electro-optical devices. b, The PIN attenuator is created by doping P++ and N++ regions on either side of a nanophotonic waveguide. The cross-section and microphotograph show the resulting structure, with a scale bar of 20 μm. c, The relationship between the injected current and attenuation of the PIN attenuator. d, This is a microphotograph of a SiGe PD that was used after each PIN attenuator. The scale bar in the image represents 15 μm. e, This is a microphotograph of the MRM that was used to implement the ReLU activation function. The scale bar measures 15 μm. f, For the case that the micro-ring is aligned with the wavelength of the supply light, when the voltage across the PN junction (VM) is smaller than the turn-on voltage of the PN junction (VTH), the junction remains off. In this case, no carriers are injected into the junction and the micro-ring resonance remains unchanged, resulting in a low neuron output power. g, When VM is greater than VTH, the PN junction becomes active and injects carriers into the junction. This causes a change in the refractive index of the waveguide, which shifts the micro-ring resonance. As a result, there is an increase in neuron output power as VM (which corresponds to the weighted sum of neuron inputs) increases. The measured output power of MRM (normalized to supply light power) varies with voltage across micro-ring PN junction [2].

With this basic structure of neuron the author can construct the PDNN classifier chip.

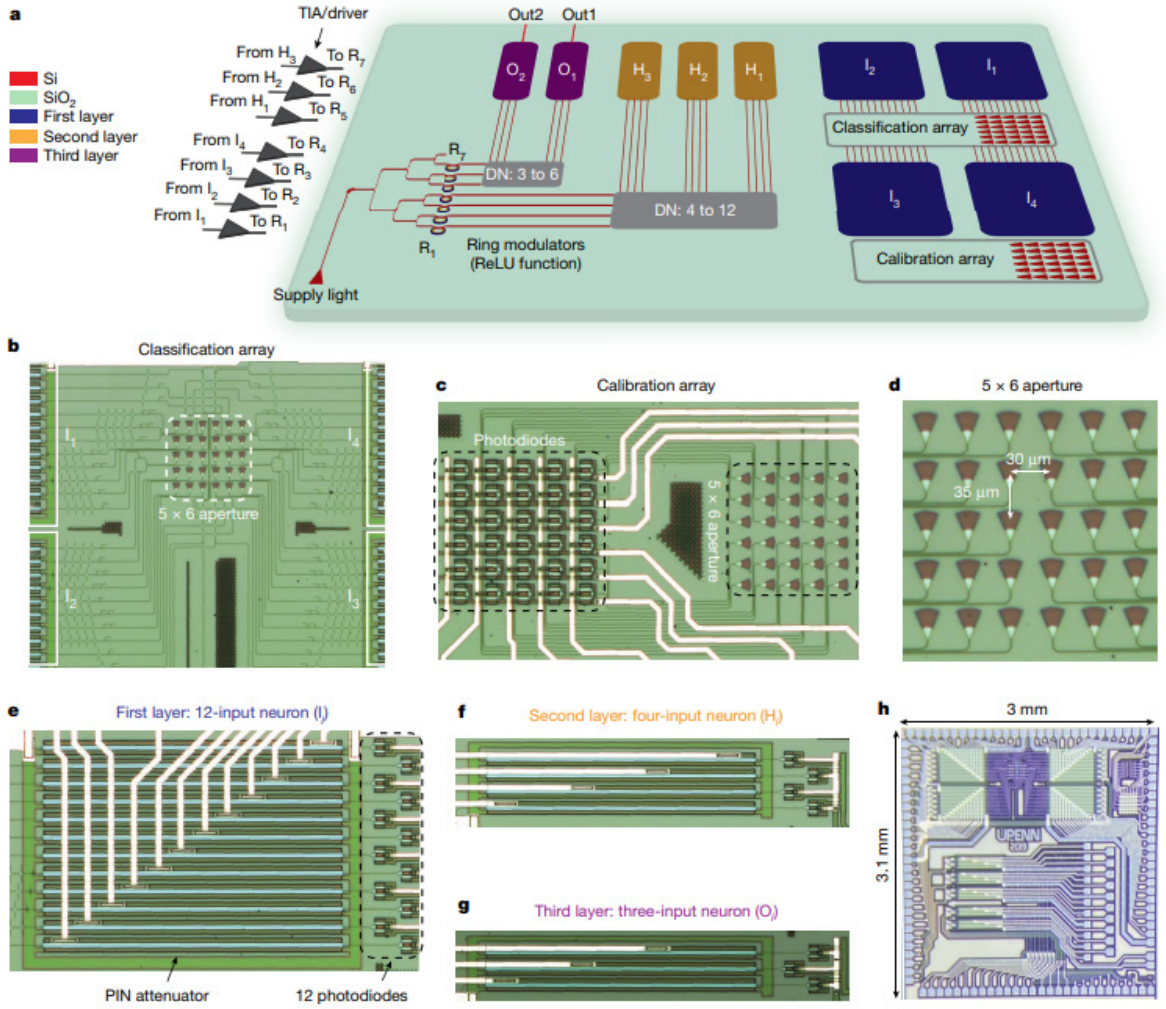


Figure 2.3: The implemented photonic classifier chip. a, The PDNN chip's top-level block diagram includes an input pixel array (b) and a calibration array (c), both consisting of two 5×6 arrays of grating couplers. d, The grating couplers are arranged in a 5×6 array, with each element having its own pitch. The input pixel array is used for classification and generates four sets of 12 optical signals that are directed to the first layer's neurons. The supply light is evenly distributed among the second and third layers' neurons, and seven MRMs are utilized to achieve the ReLU non-linear activation function. Seven off-chip TIAs drive on-chip modulators. The system produces two outputs that can be employed for up to four-class classification purposes. e–g, Presented the microphotographs of a single neuron in the first, second, and third layers. These images showcase the PIN attenuators and parallel PDs located after the attenuators in each layer. Additionally, a microphotograph of the photonic chip implemented in the AMF 180-nm SOI process is included, along with its distribution network (DN) [2].

The author designed a 5x6 input which is the one shown in Fig.2.3b and the classification array part in Fig.2.3a. The light enters the calibration array, depicted in Fig.2.3cd, after passing through the input layer. The first layer, shows in Fig.2.3e, comprises 12 inputs from the top, which pass the linear computation represented by the I1-I4 component shown in Fig.2.3a. The I1-I4 element corresponds to the four neurons in the first layer. The second and third layers are demonstrated in the same methodology as the first layer

in Fig.2.3fg. The network's progress from input to output involves the 5x6 input image, four overlapping sub-images (with 3x4 convolution), which are then reshaped into 12x1. The first layer has 4 neurons with a 12x1 input and 4x1 output, the second layer has 3 neurons with a 4x1 input and 3x1 output, and the third layer has 2 neurons with a 3x1 input and 2x1 output. The network is utilized for image classification tasks. [2]

2.1.5. DIFFRACTIVE DEEP NEURAL NETWORKS (D^2NN)

The author uses several transmissive and/or reflective layers. Each layer comprises of multiple artificial neurons that function as either transmitters or reflectors for the incoming waves. The connectivity between neurons within a layer is achieved through optical diffraction. Additionally, each unit in a given layer serves as a secondary source for waves. The amplitude and phase of the resultant wave are determined by the multiplication of the input wave with the corresponding complex-valued transmission or reflection coefficient at that particular point. [13]

The "bias" term of a neuron, which is a learnable parameter of the network, represents the transmission or reflection coefficient at that point. The iterative modification of this parameter during training allows the network to adapt and improve its performance. Furthermore, the author states that the phase and amplitude of each neuron as additional learnable parameters. In the author's scenario, coherent transmissive networks with phase-only modulation are employed, and each layer can be approximated as a thin optical element.[13]

The diffractive neural network operates by feeding training data into the input layer, and subsequently generating output through optical diffraction. Through this process, each layer of neurons within the network is trained iteratively to perform specific functions. To optimize the network's performance, an error back-propagation algorithm is employed, which adjusts the network's topology and neuron phase based on the discrepancy between the target and actual output. The algorithm aims to minimize this error and enhance the network's performance in predicting the output. [13]

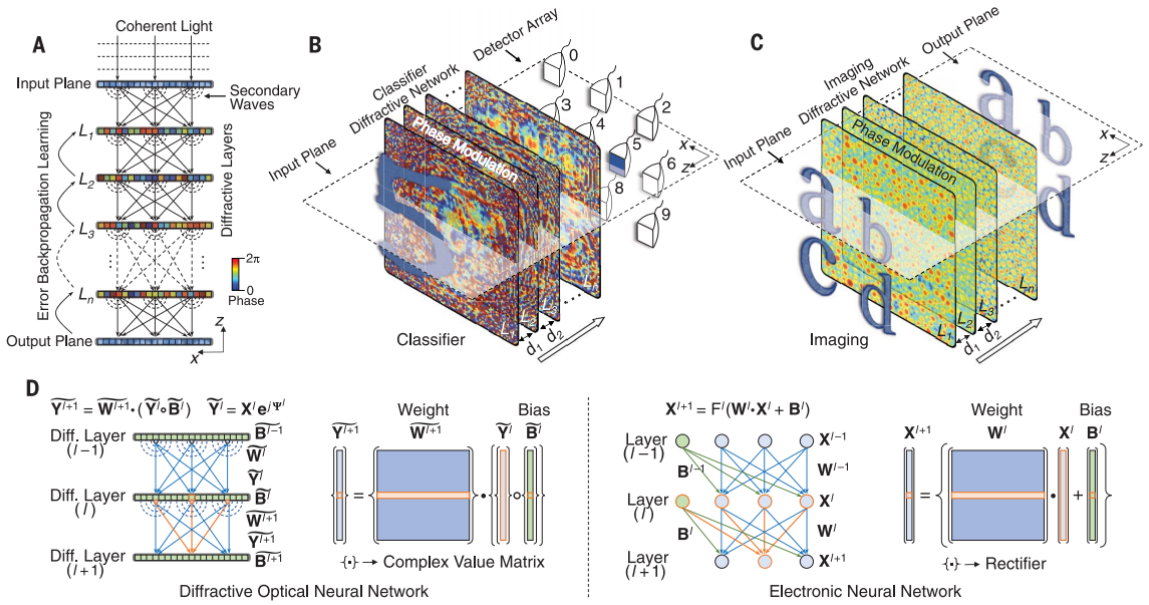


Figure 2.4: (A)A D^2NN is a network that consists of several layers, each with transmissive or reflective properties. Each point on a layer functions as a neuron and has a complex-valued transmission or reflection coefficient. By using deep learning techniques, the transmission or reflection coefficients of each layer can be trained to perform specific functions between the input and output planes of the network. Once this learning phase is complete, the design of the D^2NN becomes fixed. It can then be fabricated or 3D-printed and will perform its learned function at lightning-fast speeds equivalent to that of light. L in the graph represents each layer in this type of network design. (B and C) Various types of D^2NN have been trained and experimentally implemented, including (B) a classifier for handwritten digits and fashion products and (C) an imager that utilizes distance. (d for distance) (D) Comparison between a D2NN and a conventional neural network. The D^2NN is a neural network that operates on complex-valued inputs using coherent waves and includes multiplicative bias terms. The weights in this network are determined by free-space diffraction and control the interference of secondary waves that have been phase- or amplitude-modulated by previous layers. "o" denotes a Hadamard product operation. "Electronic neural network" refers to the conventional neural network virtually implemented in a computer. Y, optical field at a given layer; Ψ, phase of the optical field; X, amplitude of the optical field; F, nonlinear rectifier function [13]

2.2. NEURAL ARCHITECTURE SEARCH (NAS)

2.2.1. OVERVIEW

Deep learning has demonstrated powerful learning capabilities in many fields, such as image classification, object detection, and speech recognition. Much of this success is due to the continuous emergence of new neural network architectures like ResNet [14] and DenseNet [15]. However, designing high-performance neural network architectures requires a great deal of experience and iterations, which is extremely time-consuming. As for optical neural networks, since ONNs have significantly lower tunability than traditional ANNs, for example, after designing and manufacturing a designated structure, if the structure needs to be changed, it needs to be rebuilt from scratch. Therefore, Neural Architecture Search (NAS), a technique for automatically designing the architecture

for neural networks, can be introduced into this work to help us minimize this effect. In NAS, there are three main components: search space, search strategy, and performance estimation strategy. These three components are highly bounded, so we will explain them from the classification perspective on search strategies.

2

2.2.2. SEARCH STRATEGY BASED ON REINFORCEMENT LEARNING

The symbolic event in the design of neural network architectures from manual design to automatic design occurred in 2016. The author from [16] applies reinforcement learning to neural architecture search (NAS-RL) and outperforms manual-designed network architectures previously on image classification and language modeling tasks. The author uses RNNs as the controller to generate child networks, then train and evaluates to get the network performance as a reward for reinforcement learning. According to this reward, the optimal RNN and network structure can be obtained by means of gradient optimization. However, this approach has a fatal shortcoming: due to its discrete search strategy, it is very computationally intensive.

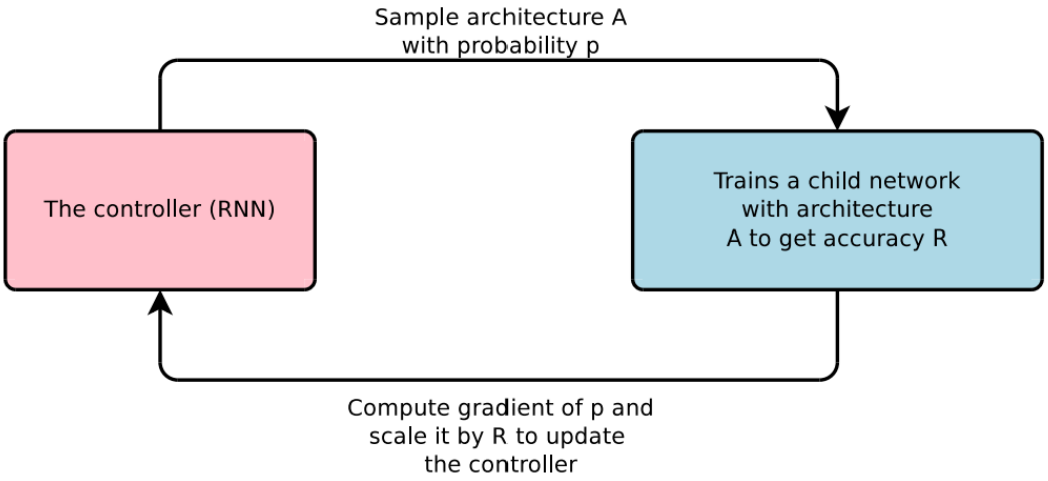


Figure 2.5: An overview of Neural Architecture search from [16]

Based on ideas from [16], motivated by skip connections [14] and repeat motifs [17], the author innovatively designed a new search space called NASNet search space [18]. NASNet [16] does not search the whole architecture. Instead, the author uses the idea of stacking convolution cells, which means the normal cell output is the same size as the input, and the reduction cell performs a down-sampling to output a reduced-resolution feature map. The controller only needs to learn two types of cells, and the final candidate architecture is obtained by stacking these cells. By using this approach, the size of the search space can be greatly reduced and can be better adapted to other datasets.

2.2.3. SEARCH STRATEGY BASED ON EVOLUTIONARY ALGORITHM

Genetic/Evolutionary Algorithm also plays an important role in NAS, the author [19] first applied Evolutionary Algorithms (EA) to NAS. The author evolves a population of mod-

els. Each individual is a trained architecture. The model's accuracy on the dataset is a measure of fitness. During each evolutionary step, a worker randomly selects two individuals and compares their fitness. The loser will be killed, and the winner will be a parent to perform mutating and join the population through training and evaluation, phasing out worse-performing architectures to get the best architecture. The experimental results also prove the effectiveness of EA in NAS. Afterward, Regularized Evolution proposed by [20] makes an improvement to the search strategy, adding the concept of "aging" to candidate architectures. The idea of "aging" is at the beginning, the whole population is placed in a queue, and when a new individual is added, removing the individual at the head of the queue, this has resulted in a younger population. Finally, it achieved a new state-of-the-art performance at that time.

2.2.4. SPEED UP RL AND EA-BASED NAS SEARCHING

Methods shown above require hundreds to thousands of GPU days to compute [6]. Although these early NAS approaches have achieved good results, it is highly time-consuming and computing-consuming, limiting the development of this field. Therefore, accelerating searching has become a very important direction. To accelerate searching, [21] proposed a new approach, first, reduce search space based on NASNet, second, a heuristic search strategy that searches the architectures from simple to complex, third, use a proxy function to predict model accuracy, so that algorithms do not need to train the complex model to saves time.

Efficient NAS (ENAS)

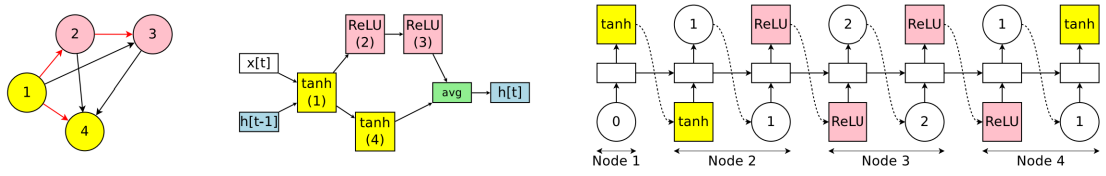


Figure 2.6: An example of a recurrent cell in ENAS with four computational nodes. Left: The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. Middle: The recurrent cell. Right: The outputs of the controller RNN result in the cell in the middle and the DAG on the left. Note that nodes 3 and 4 are never sampled by the RNN, so their results are averaged and are treated as the cell's output. [22]

To make the NAS algorithm more efficient, Efficient NAS [22] (ENAS) was introduced. The sampling process of ENAS is turned into the process of sampling a subgraph inside a Directed Acyclic Graph (DAG), where the operations (convolution, etc.) inside the sampled subgraph share weights. The weights of the subgraph in ENAS are inherited from DAG rather than reinitialized because weight-sharing greatly speeds up the NAS process, this strategy is gradually accepted in this field. However, in order to share weights, the number of nodes of candidate architectures must be the same, this limits the diversity of the generated models.

Neural Architecture Search for Mobile (MnasNet) & Once-for-all (OFA)

2

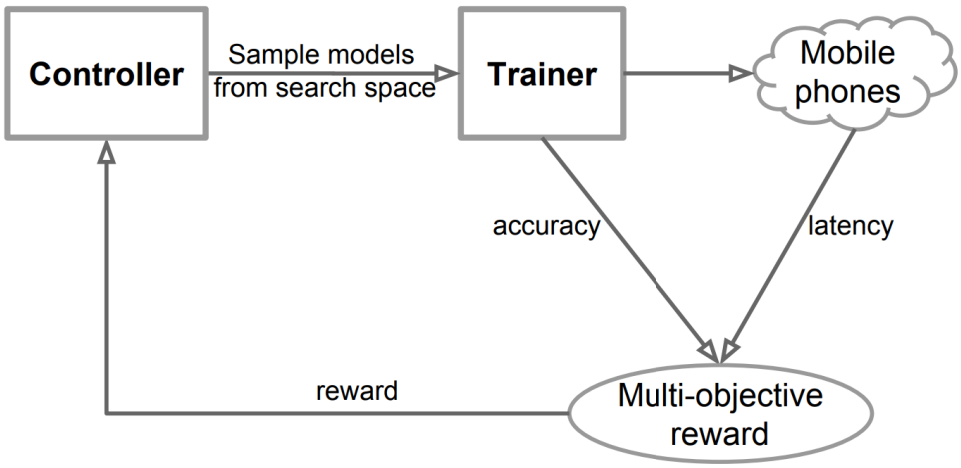
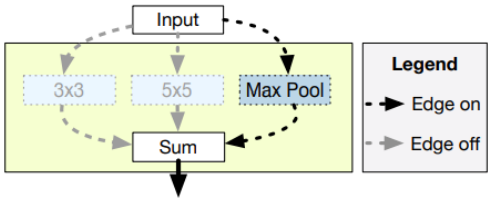


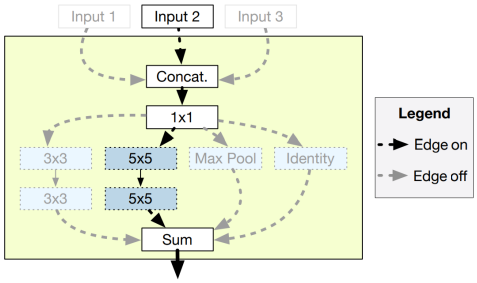
Figure 2.7: An overview of Platform-Aware Neural Architecture Search for Mobile from [23]

For most of the search strategies described in sections 2.2 and 2.4, algorithms are basically redesigning a new search space, although good performance has been achieved, it is not suitable for model deployment. The author [23] proposed a MnasNet designing convolutional neural networks for mobile devices, it incorporates model latency into the main objective so that the search can identify a model that achieves a good trade-off between accuracy and latency. The author [24] proposed a once-for-all (OFA) architecture search algorithm, which could handle multiple deployment scenarios. It decouples the model training stage and the neural architecture search stage, designing and training a once-for-all network that supports different architectural configurations. Depending on the actual deployment scenario, select proper sub-architectures from OFA so that there is no need for an additional training process.

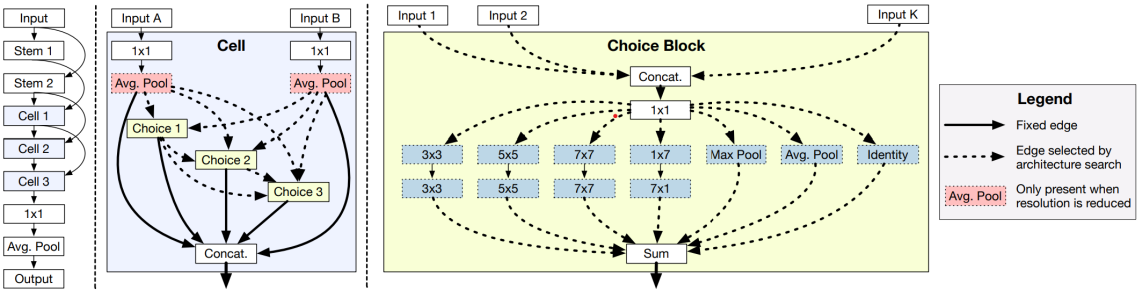
2.2.5. SEARCH STRATEGY - ONE-SHOT



(a) Example building block used in a one-shot model. The search space comprises three different operations, which are combined by the one-shot model to produce their outputs. During the evaluation, some of the operations may be removed or zeroed out.



(b) Example of a cell during one-shot model evaluation. While the one-shot model consists of four different operations, it is possible to emulate a cell structure that contains a max-pooling operation by eliminating the remaining operations from the network, without the need for weight retraining.



(c) Diagram of the one-shot architecture used in [25]. Solid lines indicate components that are present in every architecture, while dashed lines indicate optional components that are part of the search space.

Figure 2.8: An example of One-Shot Neural Architecture Search from [25].

The strategy based on weight sharing is still evolving, the most representative weight-sharing method is the One-Shot method, the whole search space is built as a supernet, and all the possible sub-architectures that be sampled with different architectures can share weights. Many researchers continuously propose new ideas based on this method. The author [26] proposed to use existing networks as a starting point to explore the search space by means of network transformation instead of from scratch. Conventional networks are time-consuming to validate, The author [27] proposed a training-assisted network called HyperNet, which could dynamically generate model weights for different structures. In the early stage of training, the generated weights are correlated to fully-trained weights, so that it is possible to sort a large number of structures by using only one epoch of training. The author [25] pre-trained a large-scale One-Shot model with all possible candidate operations. Then continuous dropout operations and measure the impact on the model. The experimental results show that the One-Shot model learns

which operations in the network are useful and relies on which operations so that it is possible to efficiently identify promising architectures without RL or hypernetworks.

Besides, the ability to sort models according to their capabilities is crucial to NAS, traditional methods like Learning Curve Extrapolation [28], [29] can achieve it, however, the cost is still high, through weights sharing, One-Shot models could reduce cost, but weights sharing cannot be sure that it really works. It is also unclear whether the models are selected for better performance because of those features or simplicity since they are over-trained. The author [30] proved that the One-Shot model's biased evaluation is due to inherent unfairness in the supernet training. they set a constraint called strict fairness, which ensures equal optimization opportunities for all choice blocks, the experimental results show a steady increase in the average accuracy curve with no oscillations.

Dfferentiable architecture search (DARTS) and its variants

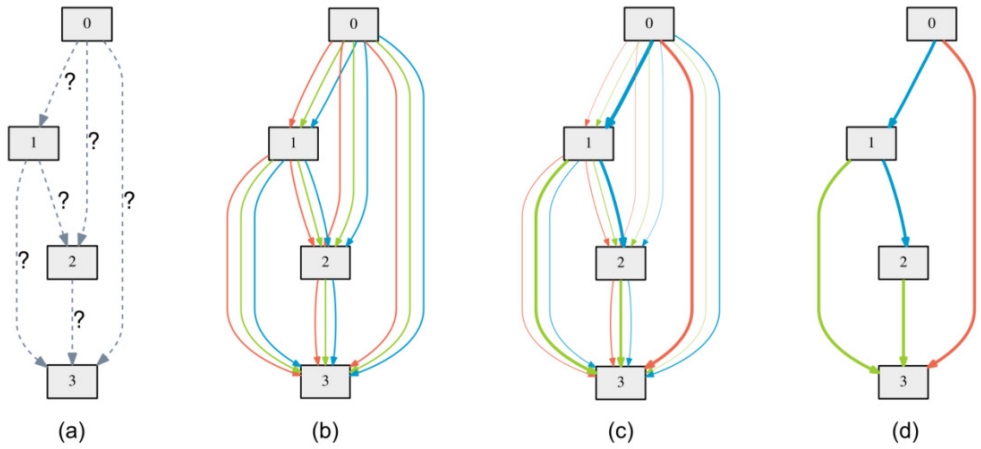


Figure 2.9: An overview of DARTS: (a) The operations performed on the edges are unknown at the beginning. (b) The search space is been continuously relaxed by placing a mixture of candidate operations on each edge. (c) The bilevel optimization problem is solved to jointly optimize the network weights and the mixing probabilities. (d) The final architecture is derived based on the probabilities learned from the optimization process. [4]

What makes NAS so inefficient is that they regard NAS as a black-box optimization problem in a discrete search strategy, in order to fix this problem, DARTS [4] was developed. Unlike previous approaches, which search in discrete search space, DARTS relaxes the search space to be continuous, while searching, each operation in the candidate operation set processes the feature map of each node and gets a weighted summation of all results, where node is a latent representation (e.g. a feature map in convolutional networks). For the architecture weights, repeat doing gradient descent on the loss of validation set to update architecture weights and doing gradient descent on the loss of training set to update operation weights. DARTS run fast since it does not need to evalu-

ate large amounts of network structures. However, it takes up lots of GPU memory when computing stack blocks on large-scale datasets. The author [31] proposed ProxylessNAS that directly optimizes the network architecture on target task and hardware, with which if you want the results of a dataset on certain hardware, search directly on that hardware, rather than search in CIFAR-10 then transfer to a large-scale dataset. The author [32] proposed P-DARTS that the depth of the search structure gradually grows during the training process, meanwhile, using search space approximation and regularization to solve heavy computational overheads and weaker search stability, this approach could complete searching in about seven hours on single GPU.

2.2.6. A SPECIAL TYPE OF SEARCH SPACE - BENCHMARK

NAS theory has evolved rapidly, and comparison between different search algorithms for NAS is either very different [33] For different scenarios or hardware, different strategies may have better performance case by case.

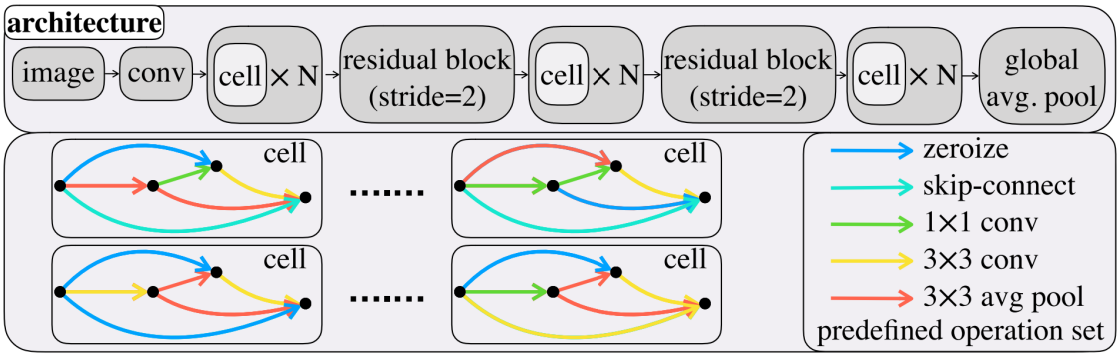


Figure 2.10: An overview of the architecture of NASBench201 - At the upper part of the diagram, the macro skeleton of each architecture candidate can be observed. On the bottom-left section, examples of neural cells comprising of four nodes are shown. It is notable that every cell is a directed acyclic graph (DAG), in which every edge is associated with an operation selected from a predetermined set of operations, as demonstrated in the bottom-right section. [3]

Since NAS is highly computationally intensive, which makes it difficult to reproduce experiments, the author [34] proposed a Benchmark named NASbench101, which is the first publicly available network architecture dataset for the research of NAS. NASbench101 built a compact cell-based search space [16][18][35], exploiting graph isomorphisms to 423k unique convolutional architectures, they trained and evaluated CIFAR-10 and compiled over 5 million trained models into the dataset, which allows researchers to evaluate the quality of a diverse range of models in milliseconds by querying the precomputed dataset. NASBench201 [3] is an extension of NASBench101, with its search space encompassing all potential cell structures generated by four nodes and five associated operations. The choice of four nodes is based on the minimum number needed to enable the search space to include basic residual block-like cells. Each node in the search space represents the sum of all feature maps transformed through the associated operations of the edges pointing to that particular node. The training logs and performance

for each candidate architecture are provided under the same settings for three datasets (CIFAR-10, CIFAR-100, and ImageNet) and based on this the author provides an extension named NATS-bench [5]. Since the search spaces for existing tabular Benchmark NASbench101 and NASbench201 are limited, the author [36] proposed NASbench301, which is the first surrogate Benchmark contains 10^{18} architectures and 60k architecture evaluations. These benchmarks can be regarded as well-designed search spaces and show great compatibility with various kinds of search strategies with a reasonable number of unique architectures.

3

Neural Architecture Search and Optical Neural Network

In this chapter, an in-depth exploration of a potential combination of two cutting-edge techniques - NAS and ONN is conducted. Each method is examined separately to assess the feasibility of effectively integrating NAS and ONN. NASLib [37, 38], a powerful and widely used Neural Architecture Search framework, is employed to study the fundamental features mentioned in the proposed method of NAS. Concurrently, a powerful ONN simulation library, PyTorch-ONN [7], is thoroughly investigated to determine the viability of combining these two components. By conducting separate evaluations of NAS and ONN, a comprehensive understanding of the properties of each component and the potential advantages of their integration can be obtained within the context of this scientific investigation. Based on the findings from the previous sections, a novel combination of NAS and ONN is introduced, including the theoretical basics and software implementation.

3.1. Choosing a proposed solution

3.1.1. Reasons to use NAS for ONN

Our target is to find an ONN structure with the best accuracy under a specific implementation that partially ignores the physical limitations and manufacturability. The basic unit to optimize for is the composition of the layer and neuron rather than the physical parameters of each optical device that make up the layer and neuron. It is hard to manually choose the optimal structure for the on-chip ONNs since the number of options for combinations is huge. Specifically, it is important to directly generate the neural network with the best structure for a specific task, for example, image classification. Even if we only consider limiting each parameter to a small range, the selections of parameters grow exponentially. This search volume cannot be done manually. Therefore, Neural Architecture Search (NAS) is a natural idea to find the best structure for the neural net-

work. Hence, a theoretical best structure for an ONN with specific implementation for the image classification task might be found.

3.1.2. PROPOSED SOLUTION AND REASONS

MZI-ONN has an intuitive implementation, and it inspired several different designs. Theoretically, this design is relatively easy to stack since the way to make ONN deeper and larger is to stack the basic structure. It has complete simulation code from [7] and [9]. Here the version from [7] will be used since it's much more efficient compared to the other one, and it has the simulation implementation for the convolution layer. The physical implementation of this simulation is comes from [8].

NASbench201 is a search space inspired by cell-based algorithms. It consists of a macro skeleton with four nodes, and each node has designated operation candidates, which provide a fixed suitable size search space. DARTS can learn high-performance architectural building blocks with complicated graph topologies. The search cell structure in NASBench201 is a directed acyclic graph, showing the strong compatibility between the chosen search space and the search strategy. Compared to the original search space used in DARTS [4] and other search spaces with similar functions (like NATS-bench [5], etc.), NASbench201 [3] has fewer operational candidates [4] and unique architectures [5], which can improve the efficiency of the computation. Due to computational resource limitations, this smaller and compatible search space is suitable for our ONN work. Moreover, NASbench201 can provide additional information, including architecture computational costs (number of parameters, FLOPs, and latency), fine-grained training and evaluation information (changes in loss and accuracy of every architecture after every training epoch) and parameters of optimized architecture to help us understand the progress better. It is also possible to add more output information, such as the structure change for each epoch during the search optimization progress. Thus, in this research, NASBench201 is an ideal search space choice for searching for the best ONN structure. [3][4]

After selecting the search space, the search strategy is one of the main considerations in NAS. DARTS, a differentiable NAS strategy, has competitive performance and efficiency. It's a gradient-based optimization combined with reduced search space which can compute our combined NAS and ONN problem more efficiently. Besides, the model learned under a certain dataset can transfer to another dataset, which shows high flexibility for our trained model. Moreover, this method can be adapted to different ONN methods, showing a potential extension of this work.[4]

Finally, we can use gradient descent methods to optimize the validation loss for evaluating the algorithm's performance, which is the same as the one used in DARTS.[4] An approximate gradient is used to accelerate this optimization progress. Then it starts to repeat the following two steps before it converges: gradient descent on validation loss to update architecture and gradient descent on training loss to update weights. The biggest advantage of this method is to use an approximate gradient method to accelerate this optimization progress. Hence, this work can compute our ONN more efficiently. [4]

In summary, this work chooses the NASbench201 as the search space, DARTS as the search strategy, and the gradient descent method from DARTS as the performance estimation strategy. These three parts are compatible with each other, and they can achieve high efficiency within the limitation of the ONNs. The NAS and ONN components require prior independent research to explore the feasibility of the combination.

3.2. NEURAL ARCHITECTURE SEARCH ON NASLIB

NASLib is a flexible and modular framework designed to study Neural Architecture Search (NAS). It offers a common codebase for the NAS, which includes high-level abstractions for designing and reusing search spaces, interfaces to benchmarks, and evaluation pipelines. The modular structure of the framework allows for effortless modification of individual components, such as defining a new search space while reusing an optimizer and evaluation pipeline or proposing a new optimizer with existing search spaces [37, 38]. And this is what this study intended to do in the next subsection.

As discussed in previous sections, NASBench101 consists of over 423,000 unique architectures [34], NASBench301 has approximately 23,000 architectures [36], the original DARTS search space contains 10^{18} different architectures [4]. In comparison, NASBench201 has only 15,625 architectures (after removing isomorphic ones, there are 6,466 unique architectures) [3]. This makes NASBench201 is currently one of the smallest available cell-based search spaces [38] and a suitable choice under our conditions.

Previous studies on NASBench201 [3] in NASLib, working on CIFAR-10 [39], demonstrated that NASBench201 could achieve greater than 80 percent classification accuracy on ANN, regardless of the searching method employed [40]. In the original NASBench201 paper [3], the authors evaluated ten different searching algorithms, including DARTS, Reinforcement Learning-based, and Evolutionary Algorithm-based methods. Another study based on NASBench201 [40] utilized Bayesian optimization from [41–43] and more Evolutionary Algorithm from [44, 45] on NASBench201 for CIFAR-10 [39], taking about 10^6 seconds on a single GPU for both experiments to get a similar accuracy result. However, unlike the original paper [3] discuss the potential issues, a fair result is obtained in the setup for the following experiment based on the implementation on NASLib. The advantage of this setup is that DARTS can leverage the optimization of all shared parameters [3, 4], allowing the task to be completed with fewer epochs when adjusting the same number of parameters in the search space [3] comparing to some other searching methods. In this research, the combination of search strategies from DARTS and NASBench201 as the search space was selected, given their compatibility and promising search efficiency. An experiment with 50 epochs (to limit computational resource consumption) was conducted under these conditions, with the results detailed below.

Search Epoch	Top 1 Train Acc.	Top 5 Train Acc.	Top 1 Eval. Acc.	Top 5 Eval. Acc.
1	28.79	83.40	29.23	83.02
5	58.87	95.69	58.87	95.54
10	70.07	97.82	69.68	97.87
20	80.22	99.10	78.10	98.80
30	86.07	99.53	81.36	98.99
40	90.10	99.79	82.86	99.22
50	91.94	99.82	83.53	99.32

Table 3.1: Results of DARTS + NASBench201 on NASLib based on epoch

The result shows that the final classification accuracy is about 83.5 percent in the experiment shows above, which is within the range of the experiment reported in [40] and [3]. Thus, this setup can be considered as a reasonable approach. The structure we got at the end of the search is all the connections become identity. The author [3] also noted that the best architecture within CIFAR-10 [39] and NASBench201 is the ResNet [14].

The setting for the experiment above is 50 epochs, ReLU as the activation function, with the initial learning rate is 0.025, and the minimum learning rate is 0.001.

The table 3.1 presented above shows the Top 1 and Top 5 training and evaluation accuracies for the combined DARTS and NASBench201 model over epochs. It's evident that the accuracy of each term in the table gradually improves with a diminishing gap as the number of epochs increases, which indicates that the potential for accuracy improvement decreases as the number of epochs grows. The results clearly indicate that more than 50 epochs could still offer a noticeable performance improvement with each epoch, however, since the primary objective is to demonstrate the viability of this combination, it is already sufficient. In this experiment, most of the authors' recommendations for avoiding overfitting were adhered [3], including avoiding regularization for a specific operation and using the provided performance. Consequently, all these data substantiate that the chosen combination is appropriate for subsequent research in section 3.4.

3.3. SIMULATION OF OPTICAL NEURAL NETWORK ON TORCH-ONN

The Optical Neural Network simulation employed in this research is based on a library called Torch-ONN. Torch-ONN is a PyTorch-based simulation framework that integrates neuromorphic photonics, supporting training and inference for both coherent and incoherent Optical Neural Networks (ONNs) on GPU. With these efficient implementations and optimization, it can scale up to ONNs with millions of parameters. The biggest advantage to using this is that this powerful library can accelerate the computation progress using GPU acceleration with the massive number of functions and options that may be needed for this research and subsequent research.[7]

Compared to other types of ONNs, the MZI-type ONNs possess significant advantages in terms of physical structure complexity. Although it's not the most area-sufficient one compared to other types of ONNs such as [2] or [7], the MZI-based ONN system provides a very clear, highly scalable, and relatively simple physical structure for each layer. The author [8] also demonstrate the high compatibility of MZIs with other advanced photonic computational architectures. In addition, MZI-ONN has a large number of other all-optical devices design based on this ONN principle, such as all-optical parts as different activation functions. Hence, by using these clear and concise physical structures, there is a great opportunity for a deep neural network to be designed and assembled.

The baseline for the type of ONN chosen from the Torch-ONN is the MZI-ONN using the default setting in the demo. It should be noted that the default setting is a CNN model with two convolution layers and one linear layer. The datasets utilized in this part are MNIST [46] and CIFAR-10 [39].

The network selected and experiment designed are closely related to the specifications and operations with in the NAS search space designed below. To demonstrate which mode to use for ONN (exp. 1), whether structural changes can trigger noticeably performance changes (exp. 2), which dataset is more suitable for the search (exp. 3), and some extensive properties evaluation.

Dataset	Structure	Mode	Accuracy
MNIST	2 Conv(3*3)+1FC(classifier)	USV	99.23
CIFAR-10	2 Conv(3*3)+1FC(classifier)	USV	75.87
MNIST	3 Conv(3*3)+1FC(classifier)	USV	99.23
MNIST	3 Conv(6*6)+1FC(classifier)	USV	99.26
MNIST	3 Conv(3*3 3*3 4*4)+1FC(classifier)	USV	99.26
MNIST	3 Conv(5*5 5*5 4*4)+1FC(classifier)	USV	99.26
MNIST	3 Conv(5*5 5*5 4*4)+1FC(classifier)	Phase	70.18
MNIST	2 Conv(5*5 4*4)+1 Hidden(12)+1 FC(classifier)	Weight	99.40
MNIST	2 Conv(5*5 4*4)+1 Hidden(12)+1 FC(classifier)	USV	99.53

Table 3.2: All results of ONN experiments

The table presented above shows the results from Torch-ONN for four out of five experiments conducted to explore the fundamental properties of the ONN. It should be noted that all the data were generated under identical conditions, using the MZI-type ONN, 200 epochs, ReLU as the activation function, and a learning rate of 0.002, which is nearly the same as the default setting in the Torch-ONN demo [7]. The results in table 3.2 can be divided into four distinct experiments to derive the necessary conclusions.

3.3.1. EXPERIMENT 1 - COMPARE MODE OF MZI-ONN

Dataset	Structure	Mode	Accuracy
MNIST	3 Conv(5*5 5*5 4*4)+1 FC(classifier)	USV	99.26
MNIST	3 Conv(5*5 5*5 4*4)+1 FC(classifier)	Phase	70.18

Table 3.3: ONN experiments 1 - USV and phase

Dataset	Structure	Mode	Accuracy
MNIST	2 Conv(5*5 4*4)+1 Hidden(12)+1 FC(classifier)	Weight	99.40
MNIST	2 Conv(5*5 4*4)+1 Hidden(12)+1 FC(classifier)	USV	99.53

Table 3.4: ONN experiments 1 - USV and weight

As described and explained in [10] [8], there are three options for the mode of MZI-ONN, which include phase, weight, and USV (representing a combination of phase and weight). The name USV is derived from the matrix SVD (singular value decomposition) representation (U $\sum$ V), as the principle of MZI-ONN operation is based on matrix computation. A detailed theoretical explanation can be found in the Section 3.4. The conclusion that can be drawn is that both phase and weight contribute to the performance, with weight is being relatively more important than the phase.

3.3.2. EXPERIMENT 2 - CHANGE/ADD CONV/HIDDEN LAYER

Dataset	Structure	Mode	Accuracy
MNIST	2 Conv(3*3)+1FC(classifier)	USV	99.23
MNIST	3 Conv(3*3 3*3 4*4)+1FC(classifier)	USV	99.26
MNIST	3 Conv(5*5 5*5 4*4)+1FC(classifier)	USV	99.26
MNIST	2 Conv(5*5 4*4)+1Hidden(12)+1FC(classifier)	USV	99.53

Table 3.5: ONN experiments 2 - Change/Add Conv/Hidden layer

In this experiment, convolutional layers with 3 x 3, 4 x 4, and 5 x 5 filters and a hidden layer consisting of 12 nodes are tested on the MNIST dataset. This experiment modifies the network structure accordingly based on several different operations of the search space NASBench201 within its cells. A detailed explanation and introduction can be found in section 2.2.6 and 3.4. Hence, in this situation, it's obvious that altering the number and design of convolutional layers and adding hidden layers does have an impact on performance. This implies that the operations used in the search of the ANN can also be employed into the search of the ONNs. Among all candidates within this experiment, the best performance is achieved by the configuration with two convolutional layers using 5 x 5 and 4 x 4 filters, followed by a hidden layer with 12 nodes. This indicates that deeper neural networks exhibit superior performance in this particular experiment.

3.3.3. EXPERIMENT 3 - DATASET MNIST/CIFAR-10

Dataset	Structure	Mode	Accuracy
MNIST	2 Conv(3*3)+1FC(classifier)	USV	99.23
CIFAR-10	2 Conv(3*3)+1FC(classifier)	USV	75.87

Table 3.6: ONN experiments 3 - Dataset MNIST/CIFAR-10

Here the two datasets used in this thesis are tested and compared to demonstrate whether both dataset, especially the CIFAR-10 works well on our ONN settings. In comparison to MNIST, CIFAR-10 is a more suitable option for the Neural Architecture Search task, as it is difficult to observe a noticeable change in performance on MNIST based on the results from table 3.2 and table 3.6. Since a simple convolutional neural network on ONN can already achieve near-perfect results on MNIST, it becomes impractical to identify whether a performance improvement is significant or not when we combine NAS and ONN. Consequently, CIFAR-10 is a more appropriate choice for the following NAS for ONN tasks.

3.3.4. EXPERIMENT 4 - MZI-ONN/FFT-ONN

Dataset	Method	Accuracy	Time Consumption
MNIST	MZI [8]	99.53	77 min
MNIST	FFT [11]	97.89	20 min

Table 3.7: ONN experiments 4 - Compare MZI-ONN to FFT-ONN

The structure can be found on the main table 3.2 above. (2 Conv(5*5 4*4) + 1 Hidden(12) + 1 FC(classifier)) In comparison to the MZI-based ONN [8], the FFT-based ONN [7][11] shows a considerably faster computation speed in simulation but with lower accuracy on MNIST dataset. Given that the goal of this study is to achieve the highest possible performance and that there are significantly more all-optical devices design based on MZI-ONN studies, MZI-ONN[8] can be considered a more suitable choice than FFT-ONN[11] in this study.

3.3.5. EXPERIMENT 5 - ACTIVATION FUNCTION & LEARNING RATE

lr/activation	ReLU	Sigmoid	Tanh
0.001	99.23	99.16	99.14
0.002	98.97	99.2	99.21
0.01	99.15	99.28	99.10

Table 3.8: ONN Experiments 5 - classification Accuracy

lr/activation	ReLU	Sigmoid	Tanh
0.001	100	99.88	100
0.002	99.96	100	100
0.01	100	100	100

Table 3.9: ONN Experiments 5 - Training Accuracy

The two tables presented above contain a basic grid search for the MZI-ONN. As a simple parameter optimization, these tables demonstrate that adjusting the activation function does have an impact on the performance of ONN. Therefore, the choice of the nonlinear activation functions will be compared again in the subsequent section on combined NAS and ONN, and the results can be found in the next section.

3.3.6. CONCLUSION

Based on the experiments above, the MZI-ONN simulation framework [7] shows promising results for the goal that this research aims to achieve. The ONN parameters associated with the optimization target in the designated Neural Architecture Search indeed influence the ONN's performance, indicating that the combination of NAS and ONN has the potential to enhance ONN performance. By using the NASLib [37, 38] as a framework for NAS research and selecting a suitable search space, such as NASBench201, the integration of these components can lead to advancements in both the understanding and the performance of Optical Neural Networks. A search space adopted from NASBench201 that is suitable for Optical Neural Networks will be introduced in the next section.

3.4. COMBINED NEURAL ARCHITECTURE SEARCH AND OPTICAL NEURAL NETWORK

3.4.1. THE THEORETICAL BASIS FROM THE MATHEMATICAL AND PHYSICAL IMPLEMENTATION OF MZI-ONN BLOCKS THAT CAN BE SEARCHED BY NAS

In this section, an exploration of why the ONN linear and convolutional layer blocks can be searched by NAS from the perspective of mathematical and physical design is presented.

The connections between artificial neurons in photonic circuits are represented by a scalar synaptic weight, which is a primary memory element. Thus, the layout of interconnections can be depicted as a matrix-vector multiplication operation. In this operation, the input to each neuron is calculated as the dot product of the output from connected neurons, which is then attenuated by a weight vector. [47]

The fundamental idea for MZI-type convolutional block and linear block computation is based on matrix computation. The photonic matrix can be built by MZIs to implement the matrix multiplication. [8, 47, 48] Then the single value decomposition (SVD) can be

applied to the MZI photonics matrix M:

$$M = U\Sigma V \tag{3.1}$$

The MZI-ONN implementation can be divided into two parts, the linear part, which is the optical interference unit (OIU) that utilizes beamsplitters and phase shifters to parametrize the unitary matrices U and complex conjugate unitary matrix V. Optical attenuators are used to implement the real diagonal matrix Σ in the photonic integrated circuit. And the nonlinear part, which is mainly related to the activation. [8, 48]

Evaluation and selection of all-optical activation functions

There's an important part that was not discussed in the previous section is the role of the nonlinear optics components part, which typically serves as activation functions. In this study, two different nonlinear activation functions including ReLU and Sigmoid are incorporated into the combined NAS and ONN model. Prior research has demonstrated the existence of all-optical activation function implementations performed in MZI-ONN for ReLU [49] and Sigmoid [50].

The authors of [51] compared the two activation function listed above with several other activation functions specifically available for ONN with all-optical design in a 3-layer linear MZI-ONN for an image classification task. Among all nonlinear activation functions with all-optical implementations, the selected for this study exhibit the highest classification accuracy, which meets the goal of pursuing the best performance in this study.

MZI Linear layer Block

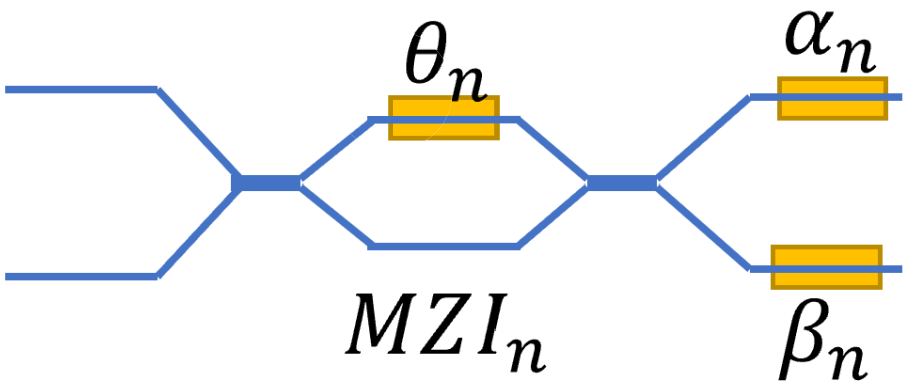


Figure 3.1: The structure of a single MZI

Figure 3.1 shows the structure of a single MZI, which is the most basic unit of OIU. The OIU can be formed into a triangular [52] or square [8, 10] shape using this structure to achieve its function. Despite the different shapes, their functional principles remain the same. Consequently, unitary matrix transformations can be achieved through architectures composed of beam splitters, phase shifters, and mirrors arranged according to specific rules [53]. The subsequent discussion focuses on the triangular structure.

A Mach-Zehnder interferometer(MZI) consists of two multimode interference couplers and two interference arms. The phase shifts on the internal phase shifters can be expressed as θ_n, as shown in Figure 3.1. The external phase shifters can be expressed as α_n and β_n, also shown in Figure 3.1. These phase shifters are thermally programmable [53]. Each MZI performs all rotations in the SU(2) Lie group rotation matrix:

$$U_{MZIn} = R(n) = \frac{1}{2}\begin{bmatrix} e^{i\alpha_n}(e^{i\theta_n}-1) & ie^{i\alpha_n}\left(e^{i\theta_n}+1\right) \\ ie^{i\beta_n}\left(e^{i\beta_n}+1\right) & e^{i\beta_n}\left(1-e^{i\theta_n}\right) \end{bmatrix} \tag{3.2}$$

Theoretically, an arbitrary n × n unitary transformation matrix SU(N) can be decomposed into the product of a series of SU(2) rotation submatrices. Using the 4 × 4 structure with transformation matrix SU(4) as an example:

$$U_2 = R_{1,1}U_1 = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & R(1) & \end{bmatrix} U_1 \tag{3.3}$$

$$U_3 = R_{2,1}R_{2,2}U_2 = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & R(3) & \end{bmatrix}\begin{bmatrix} 1 & & & \\ & R(2) & & \\ & & & 1 \end{bmatrix} U_2 \tag{3.4}$$

$$U_4 = R_{3,1}R_{3,2}R_{3,3}U_3 = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & R(6) & \end{bmatrix}\begin{bmatrix} 1 & & & \\ & R(5) & & \\ & & 1 & \end{bmatrix}\begin{bmatrix} R(4) & & & \\ & 1 & & \\ & & & 1 \end{bmatrix} U_3 \tag{3.5}$$

Therefore, a typical 4 × 4 unitary transformation matrix SU(4) can be expressed as follows:

$$SU(4) = R_{3,1}R_{3,2}R_{3,3}R_{2,1}R_{2,2}R_{1,1} \tag{3.6}$$

The above calculation can also be extended into an n × n unitary transformation matrix SU(N):

$$SU(N) = R_{N-1,1}R_{N-1,2}\cdots R_{N-1,N-1}\cdots R_{2,1}R_{2,2}R_{1,1} \tag{3.7}$$

The results indicate that when the angles are determined, the phase shifter can be adjusted to get the desired parameter, allowing the ONN layer to be effectively programmed to apply the weight matrix to the input signal amplitudes.

Given that the linear layer of MZI-ONN and the linear block of ANN operate on the same principle, both utilizing matrix multiplication computations, and we have shown in the above proof how MZI mesh is used to represent the matrix, it is feasible to use NAS to search the linear layer block of MZI-ONN.

MZI Convolutional Layer Block

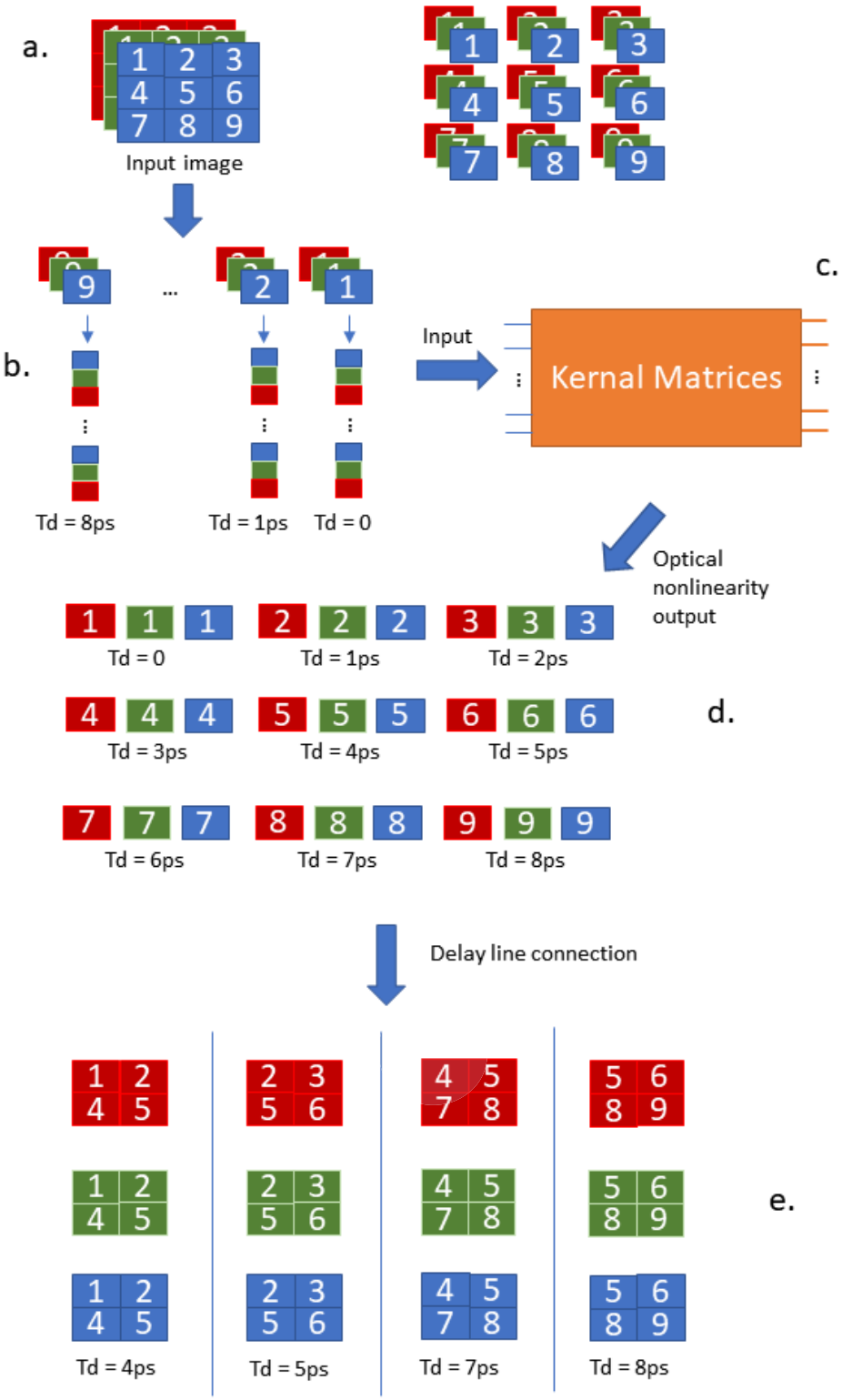


Figure 3.2: Convolution progress of MZI-ONN

The design of the optical MZI convolutional layer can be achieved by using optical delay lines with different lengths. In Figure 3.2(a), the input image pixels can be grouped into smaller patches (1 - 9 in Figure 3.2), matching the dimensions of the first layer kernels shown in Figure 3.2(c). Each patch can be reshaped into a single data column, as shown in Figure 3.2(b), which is then sequentially fed, patch by patch, into the Optical Interference Unit (OIU). In the OIU, as illustrated in Figure 3.2(c), data column signal propagation through the unit implements a dot product between the first layer kernels and the input patch vector. Then the output can be got as a time series of optical signals, with amplitudes proportional to the dot products between the patches and kernels. Optical nonlinearity is applied to each output port of the OIU. Each output port corresponds to a separate time series of dot products associated with a specific kernel, as shown in Figure 3.2(d). The optical delay lines are configured to enable a sequence of kernel dot products can be reshuffled, forming a new patch with dimensions matching those of the subsequent layer's kernels. Each delay line is connected to the original signal line through 3-dB splitters, which allows for data replication and subsequent delay for synchronization purposes. In the example shown in Figure 3.2, it is necessary to convert 9 small patches into 4 two-by-two patches as shown in Figure 3.2(e), which will appear as a time sequence input for the subsequent optical matrix multiplication. In Figure 3.2(d)(e), the output from the previous layer is split into four separate waveguides and subjected to varying delays. A one-time unit delay line is required for each two-by-two patch to ensure that the 4 signals arrive simultaneously in 3.2(e). The formation of new patches is illustrated in 3.2(d)(e), where at specific times Td, the four desired patches in 3.2(e) are created. This reshuffling process generates valid patches only at specific sampling times. (In this case is 4/5/7/8 ps) [48]

In comparison to the convolutional layer in ANN, the optical convolutional layer utilizing optical components can achieve the same target. The complex optical devices can successfully build an ONN convolutional layer to acquire the feature equivalent to the receptive field in ANN's convolutional layer (Figure 3.2(a)(e)). Subsequently, this field can be reshaped into a column, similar to the ANN's convolutional layer. (Figure 3.2(b)) The kernel matrices in the OIU can be considered as weights, while the nonlinear part can be treated as the activation function. By altering the connection of optical components and modifying the sampling times, it is possible to adjust padding and stride parameters. Given their similar functionality, convolutional layers implemented in MZI-type ONN can be compared to those in ANN. Since employing NAS to search ANN's convolutional layer block is feasible, it is also viable to use NAS to search MZI-ONN's convolutional layer block.

3.4.2. IMPLEMENTATION DETAILS

NASbench201 is a benchmark (and search space) for neural architecture search (NAS) that provides a unified benchmark for almost any up-to-date NAS algorithms. It has a fixed search space that includes all possible densely connected directed acyclic graphs (DAGs) with four nodes and five associated operation options. Each architecture consists of a predefined skeleton with a stack of the searched cell. MZI convolution is a technique to implement convolution operations in optical neural networks (ONNs) us-

ing Mach-Zehnder interferometers (MZIs), which are optical devices that can manipulate the phase and amplitude of light. MZI convolution can achieve high-speed, parallel, and energy-efficient optical computing with reduced footprint and power consumption.

In this thesis, we want to combine NASbench201 and MZI convolution for searching the structure with the best performance on visual tasks. Our proposed way to combine NASbench201 and MZI convolution is to use MZIs as the operation options in the NASbench201 search space. Each edge in the DAG can be associated with an MZI that performs a convolution operation with a certain kernel size and stride. The MZIs can be arranged in different grid structures, such as Fast Fourier Transform (FFT) grids, to realize parallel Fourier transforms and convolutions. The performance of each architecture can be evaluated using the provided training log and accuracy data on the desired dataset from NASbench201. The best architecture can be selected based on the evaluation results and used for further tasks or analysis. The following pseudo-code describes our process. A detailed flow chart that can be combined with DARTS and our search space can be found in Appendix B with Figure B.1.

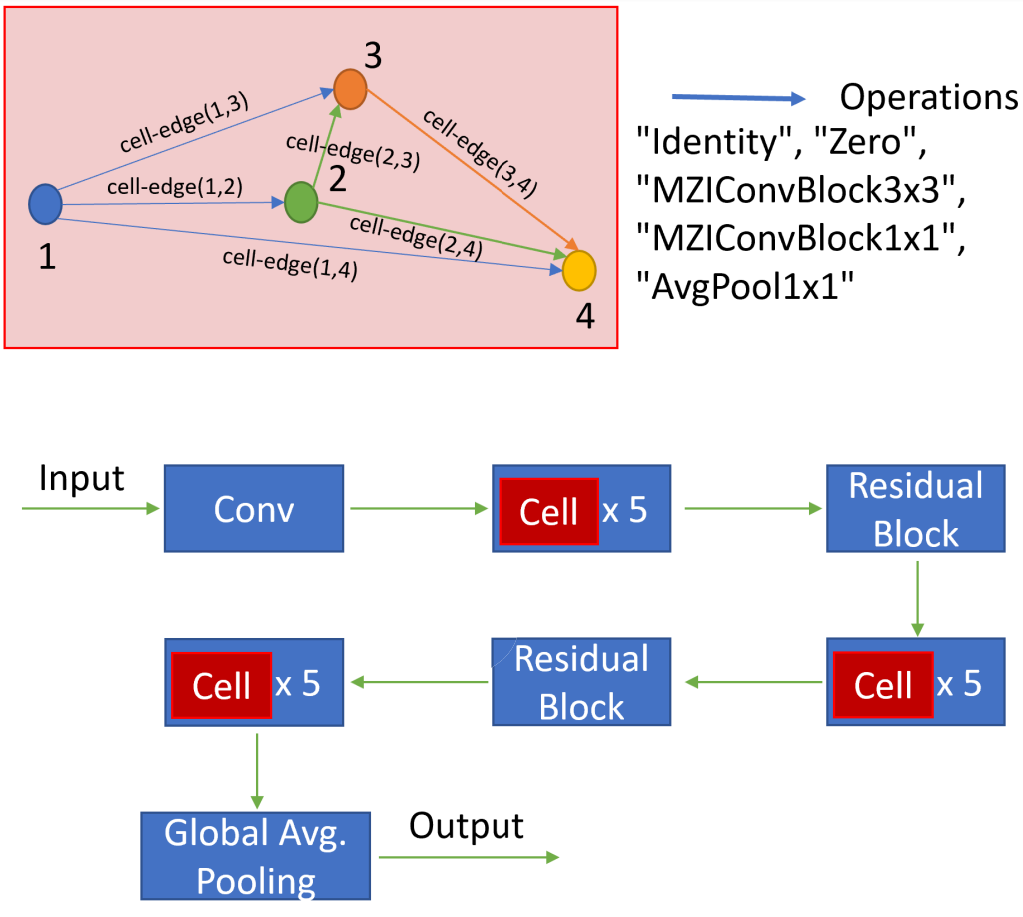


Figure 3.3: Design of ONN Search Space
Operations: Zero: Drop the edge Identity: Only pass parameters AvgPool 1x1: Average Pooling 1-by-1 Conv 3x3/1x1: 3-by-3/1-by-1 Convolution

Training Architectures

In this work, the batch size is set at 128, which is a fixed parameter. The batch size pertains to the number of training examples utilized in one iteration.

The learning rate, which controls the step size at each iteration while moving toward a minimum of a loss function, is initiated at 0.01. This deviates from the original Differentiable Architecture Search (DARTS) [4] method, which utilizes a starting learning rate of 0.025. This adjustment was made because using 0.025 in this context led to fluctuations in the reduction of the loss, and thus, the learning rate was decreased to 0.01 for stability. The final learning rate is kept above 0.001 to ensure that the model continues to learn and adjust its weights.

Stochastic Gradient Descent (SGD) is used as the optimizer with a momentum of 0.9 and weight decay of 0.0003. These are also fixed parameters.

In terms of training duration, the models were trained for 100 epochs, consistent with traditional DARTS, which is a fixed aspect of this approach.

Gradient clipping is also applied with a factor of 5. Gradient clipping [54] is a technique used to prevent exploding gradients in very deep networks. By capping the gradients to a threshold, we could ensure that the updates to the weights do not become too large, thus maintaining the stability of the learning process.

The initialization seed was set to 99, and although multiple seeds were tested, the variation in the results was less than 1%. This suggests that the model's performance is robust across different initializations.

In this work, no warm-up was used for the optimizer's scheduler, a decision based on the observation that a learning rate of 0.01 in combination with the scheduler already prevented any initial fluctuations in the loss.

The design of the cells was kept consistent with NASBench201, which is a fixed component of this architecture.

Time Consumption and Saving

In the ONN scenario presented here, there are 5 operations for each connection in the cell, resulting in $5^6 = 15625$ possible candidates within the search space. As demonstrated in section 3.3, training a specific simple optical neural network structure may take 1.5 to 2 hours to complete. In a not ideal situation, this could amount to approximately 30000 hours if each structure within the search space were trained sequentially to identify the best ONN structure. By employing NAS, the process is significantly more efficient, taking just over a day to complete and thus greatly improving the speed of selection.

Algorithm 3.1 NASbench201 for ONN

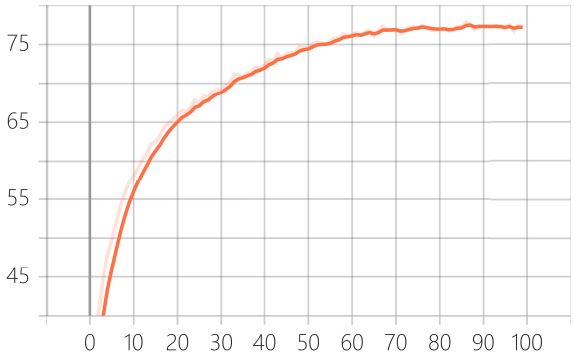
Define a search space $\mathscr{S}$ that consists of 4 nodes and 5 operations: Identity, Zero, MZIConvBlock3x3, MZIConvBlock1x1, "AvgPool1x1"
Initialize a NAS algorithm $\mathscr{A}$ with a budget B
Initialize an empty set of sampled architectures $\mathscr{A}_s$
while $B > 0$ **do**
    Sample an architecture a from $\mathscr{S}$ using $\mathscr{A}$
    if $a \notin \mathscr{A}_s$ **then**
        Add a to $\mathscr{A}_s$
        Retrieve the training log and test accuracy of a on the desired dataset from NASbench201
        Update $\mathscr{A}$ with the performance of a
        Update B with the cost of evaluating a
    end if
end while
Select the best architecture a^* from $\mathscr{A}_s$ based on the test accuracy
Return a^*

4

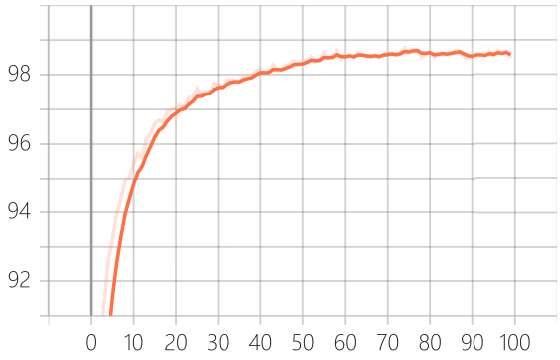
Discussion

In this chapter, an in-depth discussion and a detailed analysis and comparison of the results for the combination of NAS and ONN from the methodology introduced in the previous sections will be provided.

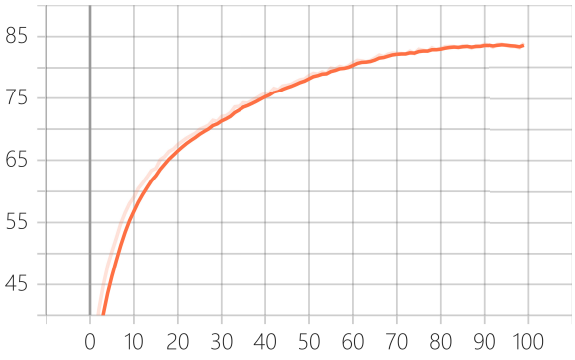
4.1. Result and analysis for NAS searching MZI-ONN



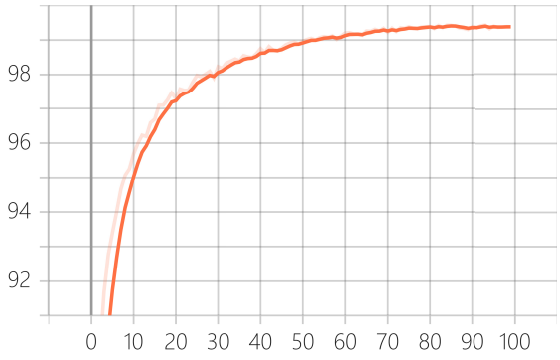
(a) Eval. Accuracy (Top 1) - Accuracy vs. Epochs



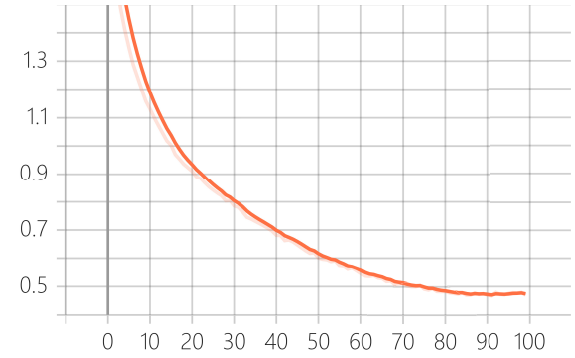
(b) Eval. Accuracy (Top 5) - Accuracy vs. Epochs



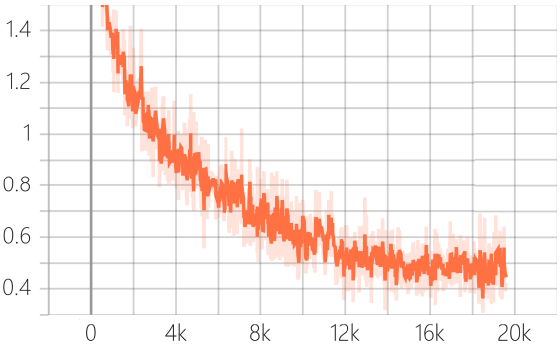
(c) Train Accuracy (Top 1) - Accuracy vs. Epochs



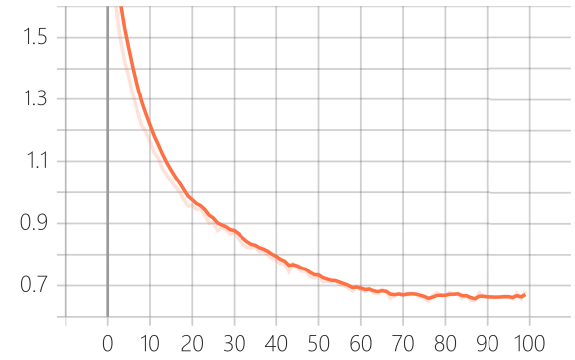
(d) Train Accuracy (Top 5) - Accuracy vs. Epochs



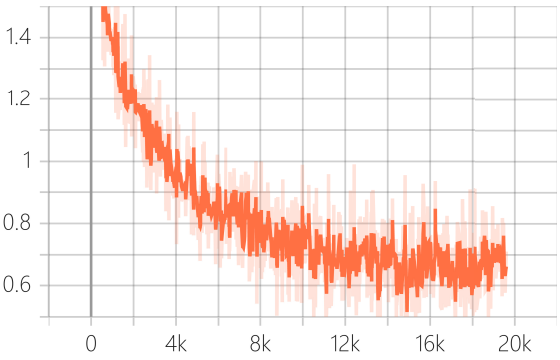
(e) Train Loss - Loss vs. Searching Epochs



(f) Train Loss - Loss vs. Total Steps of Training

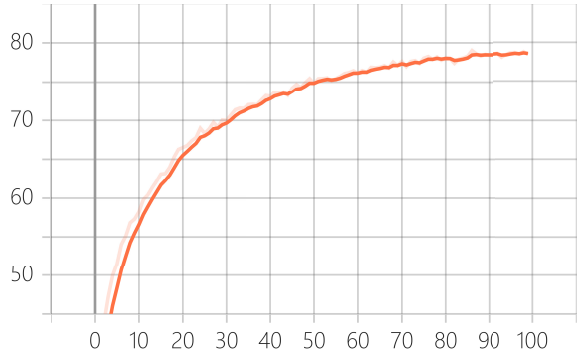


(g) Eval. Loss - Loss vs. Searching Epochs

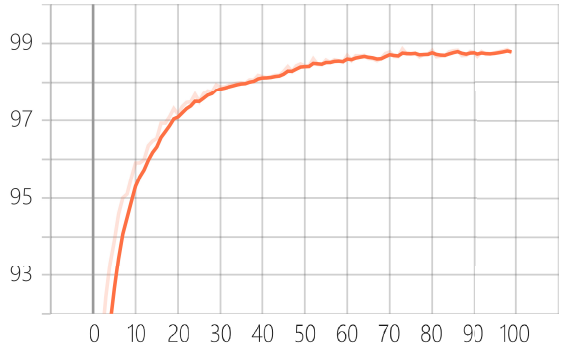


(h) Eval. Loss - Loss vs Total Steps of Training

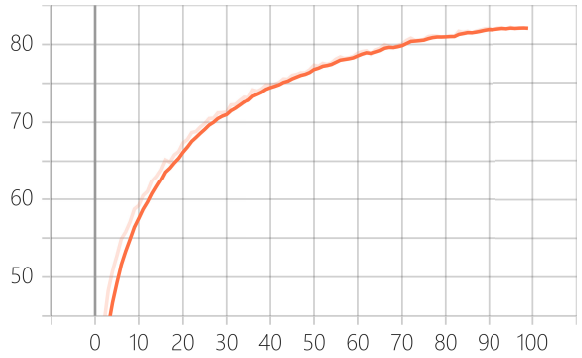
Figure 4.1: Result for Combined NAS and ONN with activation function ReLU (Light Orange: Raw Data, Deep Orange: Moving Average)



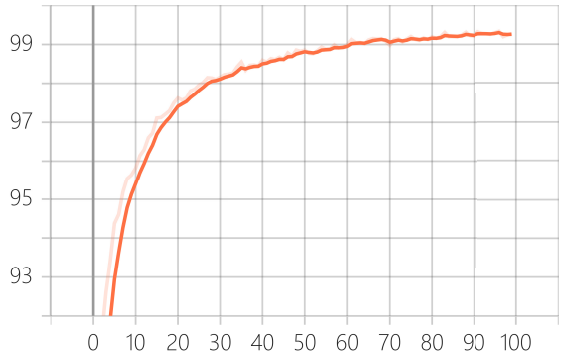
(a) Eval. Accuracy (Top 1) - Accuracy vs. Epochs



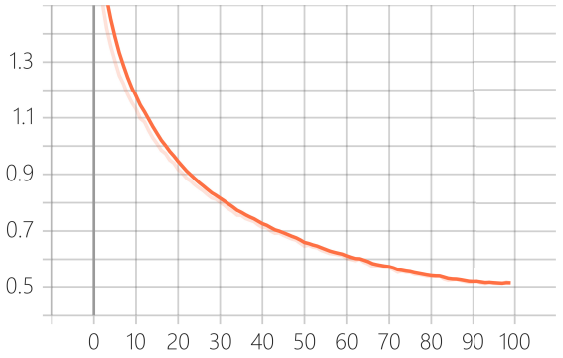
(b) Eval. Accuracy (Top 5) - Accuracy vs. Epochs



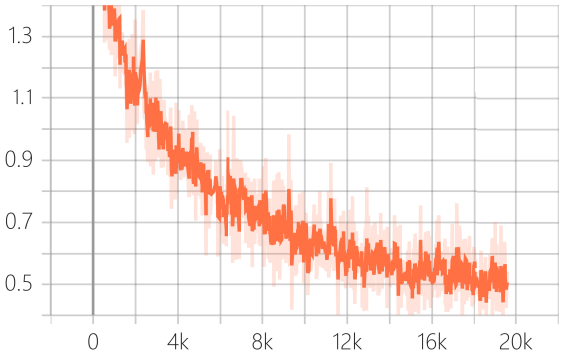
(c) Train Accuracy (Top 1) - Accuracy vs. Epochs



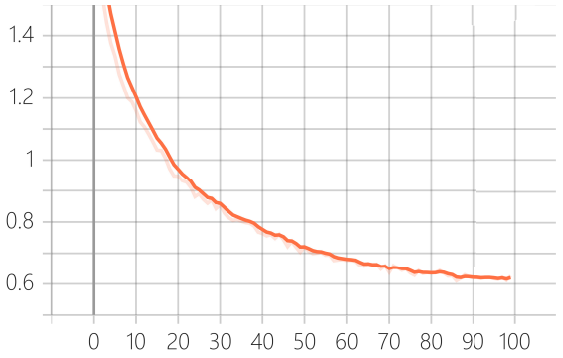
(d) Train Accuracy (Top 5) - Accuracy vs. Epochs



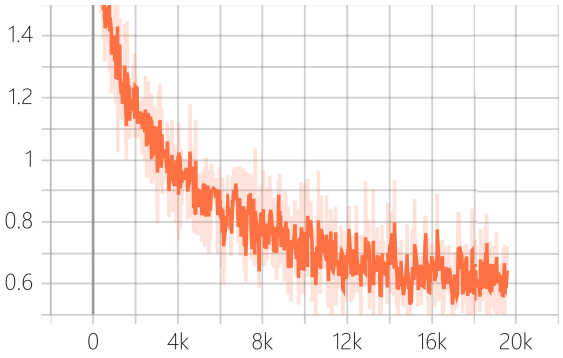
(e) Train Loss - Loss vs. Searching Epochs



(f) Train Loss - Loss vs. Total Steps of Training



(g) Eval. Loss - Loss vs. Searching Epochs



(h) Eval. Loss - Loss vs. Total Steps of Training

Figure 4.2: Result for Combined NAS and ONN with activation function Sigmoid (Light Orange: Raw Data, Deep Orange: Moving Average)

Here we show the results for the combined NAS and ONN with the activation functions mentioned in the previous part, including ReLU and Sigmoid.

	Train Acc.(Top 1)	Train Acc. (Top 5)	Eval. Acc. (Top 1)	Eval. Acc. (Top 5)
ReLU	83.91	99.38	77.14	98.46
Sigmoid	82.01	99.27	78.45	98.74

Table 4.1: Accuracy for NAS + ONN

There is no significant performance difference between the two chosen activation functions, both have noticeable improvements over our previous CIFAR-10 baseline.

The structure differs a lot between the ReLU one and the Sigmoid one.

Final Structure ReLU
(cell-edge(1,2)): Identity()
(cell-edge(1,3)): Identity()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): Identity()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): Identity()

Final Structure Sigmoid
(cell-edge(1,2)): SigmoidConvBN()
(cell-edge(1,3)): Identity()
(cell-edge(1,4)): SigmoidConvBN()
(cell-edge(2,3)): SigmoidConvBN()
(cell-edge(2,4)): SigmoidConvBN()
(cell-edge(3,4)): SigmoidConvBN()

Figure 4.3: Final structure for ReLU and Sigmoid

The ONN employing ReLU as the activation function achieves nearly the same performance as the one using Sigmoid with a considerably simpler cell structure design. From this perspective, the ReLU-based activation function in the nonlinear part of the MZI type ONN can be considered a more accessible approach in terms of manufacturing.

4.2. COMPARING THE RESULTS

In this section, the result from combined NAS and ONN will be compared to the results for each part separately to evaluate the performance between our new structure with the ANN and some simple ONN shows in the previous parts and evaluate the performance and properties of our NASBench201 for ONN search space. The result and analysis of structure optimization during the search progress can be found in Appendix A.

4.2.1. COMBINED NAS AND MZI-ONN & MZI-ONN ONLY

Method	Mode	Activation Function	Accuracy
NAS + ONN	USV	ReLU	77.14
ONN	USV	ReLU	75.87

Table 4.2: Accuracy Comparison between NAS + ONN and ONN only

A noticeable performance improvement can be noted for our combined NAS and ONN architecture compared to a simple CNN with two convolutional layers in ONN. Demonstrate that we have met and proven our primary goal that NAS can improve the performance of existing ONN architectures.

4.2.2. COMBINED NAS AND MZI-ONN & NAS AND ANN IN RELU

Method	Search Epoch	Activation Function	Accuracy
NAS + ONN	100 (Complete)	ReLU	77.14
NAS + ONN	50	ReLU	74.62
NAS + ANN	50 (Complete)	ReLU	83.53

Table 4.3: Accuracy Comparison between NAS + ONN and NAS + ANN

The results presented above demonstrate that the experiments conducted using the combination of NAS and ONN yield the same cell structure as those performed with the combination of NAS and ANN. However, the convergence in the ONN case is slower than that in the ANN case. And the difference in performance and accuracy may be attributed to the distinct network features of each system. Detailed cell structures can be found in Appendix A.

4.2.3. COMBINED NAS AND MZI-ONN & ONN RESNET FOR CIFAR

	Type	Accuracy
NAS + ONN	ONN	77.14
ResNet-20	ONN	85.81
ResNet-32	ONN	86.23
ResNet-44	ONN	86.65
ResNet-56	ONN	87.45

Table 4.4: Compare ONN ResNet with NAS + ONN Result

In this section, the results obtained from combining NAS and ONN are compared with those of some well-known CNN architectures in ANN implemented into ONN. As the structure of the combined NAS and ONN approach is significantly simpler than that of ResNet-20, this outcome aligns with expectations. A more comprehensive analysis of the ONN ResNet is provided in Appendix C.

5

Conclusion

This chapter focuses on the conclusion and recommendations, presenting a comprehensive summary of the main findings and suggestions for various directions of future research.

5.1. Conclusion

In this study, the primary goal is to integrate Neural Architecture Search (NAS) with Optical Neural Networks (ONNs). A minor simplification is predefined, which is to ignore the physical manufacturing limitation. Moreover, the feasibility of this integration is discussed from a theoretical perspective to demonstrate the similarities between ONN and ANN layers. An extra experiment to prove this can be found in Appendix C. This similarity suggests that ONNs can be searched by NAS similar to ANNs, with each main component possessing its own optical solutions.

Independent research on NAS (on CIFAR-10) and ONN (primarily on MNIST, with a small portion on CIFAR-10) reveals that tuning specific structures in the ONN, corresponding to the targets optimally tuned by the desired NAS, can have a positive or negative impact on ONN performance. Therefore, employing NAS instead of manually tuning the same targets can yield equivalent results. These studies provide a robust theoretical and practical foundation for the combination of NAS and ONN.

The results for NAS combined with ONN demonstrate that this combination achieves the predefined goal of enhancing ONN performance in visual image classification tasks. In this thesis, a cell-based search space explicitly designed for ONN is developed, based on the popular small cell-based search space, NASBench201 [3], and combined with a widely-used differentiable architecture search strategy, DARTS [4], to obtain the desired results.

Compared to the CNN with 2 convolution layers, the larger ONN model searched by our algorithm exhibits a performance improvement of approximately 2%, which demonstrates the effectiveness of our method for combining NAS and ONN. The evaluation of the activation functions has been discussed, with ReLU-based and Sigmoid-based results displaying their own characteristics. Sigmoid-based ONN and ReLU-based ONN exhibits comparable performance, while the ReLU-based ONN has a considerably simpler structure.

As the structure of the ONN cell evolves during the optimization process, the similarity between ONN and ANN structures suggests the possibility of transferring high-performance structures from ANNs to ONNs in order to achieve good results. We approved this finding in Appendix C.

5.2. LIMITATION AND FUTURE RESEARCH

5.2.1. PHYSICAL MANUFACTURE LIMITATION AND NAS SEARCHING

In this thesis, the physical manufacturing limitations are not considered during the search process. To account for these limitations, various approaches can be explored. For instance, different foundries may have distinct design specifications, which could include varying criteria for waveguides. As the ONN structure necessitates a substantial number of waveguides, splitters, crossings, etc., even minor changes may significantly impact manufacturability. It remains unknown how NAS can be adapted based on these specific physical theories and external constraints.

Manufacturing accuracy is also known to substantially influence performance, particularly when individual layer blocks are cascaded, leading to deeper neural networks. In this context, optimizing cumulative errors in manufacturing is far more critical than structural optimization and it's important for us to balance the relationship between the depth of the optical neural network and the cumulative error.

5.2.2. TRANSFER TO DIFFERENT DATASET/TASKS

Additional datasets can be used for image classification tasks, including CIFAR-100, ImageNet, Fashion-MNIST, etc. ONNs can also be applied to other tasks; for example, the authors of [8] used ONN for vowel recognition, demonstrating its potential for various applications, similar to ANN.

5.2.3. OTHER ONN IMPLEMENTATION

The ONN type employed in this thesis is based on MZI [8], which is one of the most fundamental ONN implementations available. In comparison, FFT-ONN [11] offers a more chip area-efficient solution with slightly lower performance. Moreover, recently, advanced MZI-ONN techniques have been developed, such as hardware error correction on MZI by [55] to minimize errors. These advanced ONN techniques could potentially improve image classification performance.

5.2.4. OTHER LEARNING ARCHITECTURE

The Transformer [56] is a highly influential deep learning model. In the vision classification task discussed in this paper, the Vision Transformer exhibits notably better performance than ResNet, which has the best performance in NASBench201 on CIFAR-10 [3, 57]. Recently, the Optical Transformer [58] has been introduced in the Optical Neural Network field, offering the possibility to utilize this impressive architecture and combine it with superior optical performance to accomplish the same task. It is important to note that the NAS searching algorithm developed in this thesis can only be used on CNN and is not compatible with the Transformer.

REFERENCES

[1] T. Wang, S.-Y. Ma, L. G. Wright, T. Onodera, B. C. Richard, and P. L. McMahon, *An optical neural network using less than 1 photon per multiplication,* Nature Communications **13**, 1 (2022).

[2] F. Ashtiani, A. J. Geers, and F. Aflatouni, *An on-chip photonic deep neural network for image classification,* Nature , 1 (2022).

[3] X. Dong and Y. Yang, *Nas-bench-201: Extending the scope of reproducible neural architecture search,* arXiv preprint arXiv:2001.00326 (2020).

[4] H. Liu, K. Simonyan, and Y. Yang, *Darts: Differentiable architecture search,* arXiv preprint arXiv:1806.09055 (2018).

[5] X. Dong, L. Liu, K. Musial, and B. Gabrys, *Nats-bench: Benchmarking nas algorithms for architecture topology and size,* IEEE transactions on pattern analysis and machine intelligence (2021).

[6] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, *A comprehensive survey of neural architecture search: Challenges and solutions,* ACM Computing Surveys (CSUR) **54**, 1 (2021).

[7] J. Gu, H. Zhu, C. Feng, Z. Jiang, R. Chen, and D. Pan, *L2ight: Enabling on-chip learning for optical neural networks via efficient in-situ subspace optimization,* Advances in Neural Information Processing Systems **34**, 8649 (2021).

[8] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones, M. Hochberg, X. Sun, S. Zhao, H. Larochelle, D. Englund, *et al.*, *Deep learning with coherent nanophotonic circuits,* Nature photonics **11**, 441 (2017).

[9] I. A. Williamson, T. W. Hughes, M. Minkov, B. Bartlett, S. Pai, and S. Fan, *Reprogrammable electro-optic nonlinear activation functions for optical neural networks,* IEEE Journal of Selected Topics in Quantum Electronics **26**, 1 (2019).

[10] W. R. Clements, P. C. Humphreys, B. J. Metcalf, W. S. Kolthammer, and I. A. Walmsley, *Optimal design for universal multiport interferometers,* Optica **3**, 1460 (2016).

[11] J. Gu, Z. Zhao, C. Feng, M. Liu, R. T. Chen, and D. Z. Pan, *Towards area-efficient optical neural networks: an fft-based architecture,* in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, 2020) pp. 476–481.

[12] Y. Qu, H. Zhu, Y. Shen, J. Zhang, C. Tao, P. Ghosh, and M. Qiu, *Inverse design of an integrated-nanophotonics optical neural network,* Science Bulletin **65**, 1177 (2020).

[13] X. Lin, Y. Rivenson, N. T. Yardimci, M. Veli, Y. Luo, M. Jarrahi, and A. Ozcan, *All-optical machine learning using diffractive deep neural networks,* Science **361**, 1004 (2018).

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

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017) pp. 4700–4708.

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

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016) pp. 2818–2826.

[18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, *Learning transferable architectures for scalable image recognition,* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018) pp. 8697–8710.

[19] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, *Large-scale evolution of image classifiers,* in *International Conference on Machine Learning* (PMLR, 2017) pp. 2902–2911.

[20] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, *Regularized evolution for image classifier architecture search,* in *Proceedings of the aaai conference on artificial intelligence*, Vol. 33 (2019) pp. 4780–4789.

[21] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, *Progressive neural architecture search,* in *Proceedings of the European conference on computer vision (ECCV)* (2018) pp. 19–34.

[22] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, *Efficient neural architecture search via parameters sharing,* in *International conference on machine learning* (PMLR, 2018) pp. 4095–4104.

[23] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, *Mnasnet: Platform-aware neural architecture search for mobile,* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019) pp. 2820–2828.

[24] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, *Once-for-all: Train one network and specialize it for efficient deployment,* arXiv preprint arXiv:1908.09791 (2019).

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

[26] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, *Efficient architecture search by network transformation,* in *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32 (2018).

[27] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, *Smash: one-shot model architecture search through hypernetworks,* arXiv preprint arXiv:1708.05344 (2017).

[28] B. Baker, O. Gupta, R. Raskar, and N. Naik, *Accelerating neural architecture search using performance prediction,* arXiv preprint arXiv:1705.10823 (2017).

[29] T. Domhan, J. T. Springenberg, and F. Hutter, *Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves,* in *Twenty-fourth international joint conference on artificial intelligence* (2015).

[30] X. Chu, B. Zhang, and R. Xu, *Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search,* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021) pp. 12239–12248.

[31] H. Cai, L. Zhu, and S. Han, *Proxylessnas: Direct neural architecture search on target task and hardware,* arXiv preprint arXiv:1812.00332 (2018).

[32] X. Chen, L. Xie, J. Wu, and Q. Tian, *Progressive differentiable architecture search: Bridging the depth gap between search and evaluation,* in *Proceedings of the IEEE/CVF international conference on computer vision* (2019) pp. 1294–1303.

[33] A. Yang, P. M. Esperança, and F. M. Carlucci, *Nas evaluation is frustratingly hard,* arXiv preprint arXiv:1912.12522 (2019).

[34] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, *Nas-bench-101: Towards reproducible neural architecture search,* in *International Conference on Machine Learning* (PMLR, 2019) pp. 7105–7114.

[35] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, *Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation,* in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2019) pp. 82–92.

[36] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter, *Nas-bench-301 and the case for surrogate benchmarks for neural architecture search,* arXiv preprint arXiv:2008.09777 (2020).

[37] M. Ruchte, A. Zela, J. Siems, J. Grabocka, and F. Hutter, *Naslib: A modular and flexible neural architecture search library,* https://github.com/automl/NASLib (2020).

[38] Y. Mehta, C. White, A. Zela, A. Krishnakumar, G. Zabergja, S. Moradian, M. Safari, K. Yu, and F. Hutter, *Nas-bench-suite: Nas evaluation is (now) surprisingly easy,* in *International Conference on Learning Representations* (2022).

[39] A. Krizhevsky, G. Hinton, *et al.*, *Learning multiple layers of features from tiny images*, (2009).

[40] C. White, A. Zela, R. Ru, Y. Liu, and F. Hutter, *How powerful are performance predictors in neural architecture search?* Advances in Neural Information Processing Systems **34** (2021).

[41] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, *Neural architecture search with bayesian optimisation and optimal transport*, Advances in neural information processing systems **31** (2018).

[42] L. Ma, J. Cui, and B. Yang, *Deep neural architecture search with deep graph bayesian optimization*, in *IEEE/WIC/ACM International Conference on Web Intelligence* (2019) pp. 500–507.

[43] H. Shi, R. Pi, H. Xu, Z. Li, J. Kwok, and T. Zhang, *Bridging the gap between sample-based and one-shot neural architecture search with bonas*, Advances in Neural Information Processing Systems **33**, 1808 (2020).

[44] Y. Sun, X. Sun, Y. Fang, G. G. Yen, and Y. Liu, *A novel training protocol for performance predictors of evolutionary neural architecture search algorithms*, IEEE Transactions on Evolutionary Computation **25**, 524 (2021).

[45] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, and J. Liang, *Npenas: Neural predictor guided evolution for neural architecture search*, IEEE Transactions on Neural Networks and Learning Systems (2022).

[46] L. Deng, *The mnist database of handwritten digit images for machine learning research [best of the web]*, IEEE signal processing magazine **29**, 141 (2012).

[47] B. J. Shastri, A. N. Tait, T. Ferreira de Lima, W. H. Pernice, H. Bhaskaran, C. D. Wright, and P. R. Prucnal, *Photonics for artificial intelligence and neuromorphic computing*, Nature Photonics **15**, 102 (2021).

[48] H. Bagherian, S. Skirlo, Y. Shen, H. Meng, V. Ceperic, and M. Soljacic, *On-chip optical convolutional neural networks*, arXiv preprint arXiv:1808.03303 (2018).

[49] G. H. Li, R. Sekine, R. Nehra, R. M. Gray, L. Ledezma, Q. Guo, and A. Marandi, *All-optical ultrafast relu function for energy-efficient nanophotonic deep learning*, Nanophotonics (2022).

[50] G. Mourgias-Alexandris, A. Tsakyridis, N. Passalis, A. Tefas, K. Vyrsokinos, and N. Pleros, *An all-optical neuron with sigmoid activation function*, Optics express **27**, 9620 (2019).

[51] M. Miscuglio, A. Mehrabian, Z. Hu, S. I. Azzam, J. George, A. V. Kildishev, M. Pelton, and V. J. Sorger, *All-optical nonlinear activation function for photonic neural networks*, Optical Materials Express **8**, 3851 (2018).

[52] M. Reck, A. Zeilinger, H. J. Bernstein, and P. Bertani, *Experimental realization of any discrete unitary operator,* Physical review letters **73**, 58 (1994).

[53] J. Cheng, H. Zhou, and J. Dong, *Photonic matrix computing: from fundamentals to applications,* Nanomaterials **11**, 1683 (2021).

[54] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks,* in *International conference on machine learning* (Pmlr, 2013) pp. 1310–1318.

[55] R. Hamerly, S. Bandyopadhyay, and D. Englund, *Asymptotically fault-tolerant programmable photonics,* Nature Communications **13**, 6831 (2022).

[56] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need,* Advances in neural information processing systems **30** (2017).

[57] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, *An image is worth 16x16 words: Transformers for image recognition at scale,* arXiv preprint arXiv:2010.11929 (2020).

[58] M. G. Anderson, S.-Y. Ma, T. Wang, L. G. Wright, and P. L. McMahon, *Optical transformers,* arXiv preprint arXiv:2302.10360 (2023).

[59] Y. Idelbayev, *Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch,* https://github.com/akamaster/pytorch_resnet_cifar10.

A

Appendix A
ONN cell structure searching optimization process using NAS

This section will discover the optimization process of combining the NAS and ONN. Here show the structural changes for ONN during the NAS searching optimization progress. Below shows the result from the ReLU activation function experiment in section 4. (NAS for ONN) The algorithm and process can be found in section 3.4.

Epoch 0:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): ReLUConvBN()
(cell-edge(3,4)): ReLUConvBN()

Epoch 1-2:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): ReLUConvBN()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Zero
(cell-edge(3,4)): ReLUConvBN()

Epoch 3-4:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): Zero
(cell-edge(1,4)): ReLUConvBN()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Zero
(cell-edge(3,4)): ReLUConvBN()

Epoch 5:
(cell-edge(1,2)): Zero
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): ReLUConvBN()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Zero
(cell-edge(3,4)): ReLUConvBN()

Epoch 6:
(cell-edge(1,2)): Zero
(cell-edge(1,3)): Zero
(cell-edge(1,4)): ReLUConvBN()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Zero
(cell-edge(3,4)): ReLUConvBN()

Epoch 7:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): Zero
(cell-edge(1,4)): ReLUConvBN()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Zero
(cell-edge(3,4)): ReLUConvBN()

Epoch 8:
(cell-edge(1,2)): Zero
(cell-edge(1,3)): Zero
(cell-edge(1,4)): ReLUConvBN()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Zero
(cell-edge(3,4)): ReLUConvBN()

Epoch 9-23:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): Zero
(cell-edge(1,4)): ReLUConvBN()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Zero
(cell-edge(3,4)): ReLUConvBN()

Epoch 24-28:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): Zero
(cell-edge(1,4)): ReLUConvBN()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): ReLUConvBN()

Epoch 29:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): Zero
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): ReLUConvBN()

Epoch 30-64:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): ReLUConvBN()

Epoch 65:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): Identity()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): ReLUConvBN()

Epoch 66:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): ReLUConvBN()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): ReLUConvBN()

Epoch 67-69:
(cell-edge(1,2)): ReLUConvBN()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): Identity()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): ReLUConvBN()

Epoch 70:
(cell-edge(1,2)): Identity()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): Identity()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): ReLUConvBN()

Epoch 71:
(cell-edge(1,2)): Identity()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): Identity()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): Identity()

Epoch 72:
(cell-edge(1,2)): Identity()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): Identity()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): ReLUConvBN()

Epoch 73-82:
(cell-edge(1,2)): Identity()
(cell-edge(1,3)): ReLUConvBN()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): Identity()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): Identity()

```
Epoch 83-99: (Finish)
(cell-edge(1,2)): Identity()
(cell-edge(1,3)): Identity()
(cell-edge(1,4)): Identity()
(cell-edge(2,3)): Identity()
(cell-edge(2,4)): Identity()
(cell-edge(3,4)): Identity()
```

Figure A.1: Structure of all ONN Cells during the searching progress

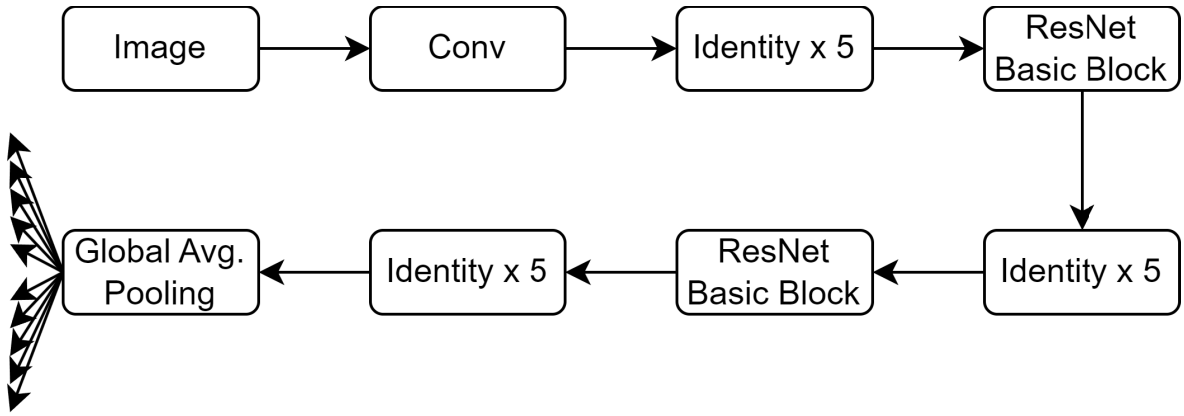


Figure A.2: Structure of the network with the searched final ONN Cell

It is quite straightforward that the structure of the ONN cell becomes increasingly simpler during the search process. Beginning with five MZI convolution connections and one identity connection, it eventually converges to identity connections for all connections. This means that the cell passes the value from the previous node to the next node without performing any additional operations. For the first 24 epochs, the search process seems to favor zero connections over convolution connections. As the epoch number increases, identity connections appear as a better choice than both zero and convolution connections. Identity connections replace all zero connections first (at 30 epochs) and then all convolution connections (at 83 epochs), until all connections in the cells become identity connections. The overall trend for the structure of the cells is to simplify gradually until the simplest structure configuration is reached.

During the search progress, this 'step-like' improvement in structure is quite similar to the findings presented in [3] and [40]. Furthermore, the structure of the ONN final search result is identical to the optimal structure for ANN-based NASBench201 on the CIFAR10 dataset, implying a potential architectural property similarity between ANN and ONN. Consequently, it can be hypothesized that structures yielding relatively better performance in ANN may also achieve comparable performance improvements under specific ONN implementations. An experiment to prove this can be found in the Appendix C.

B

Appendix B
Flow chart of Neural Architecture Search for ONN

B

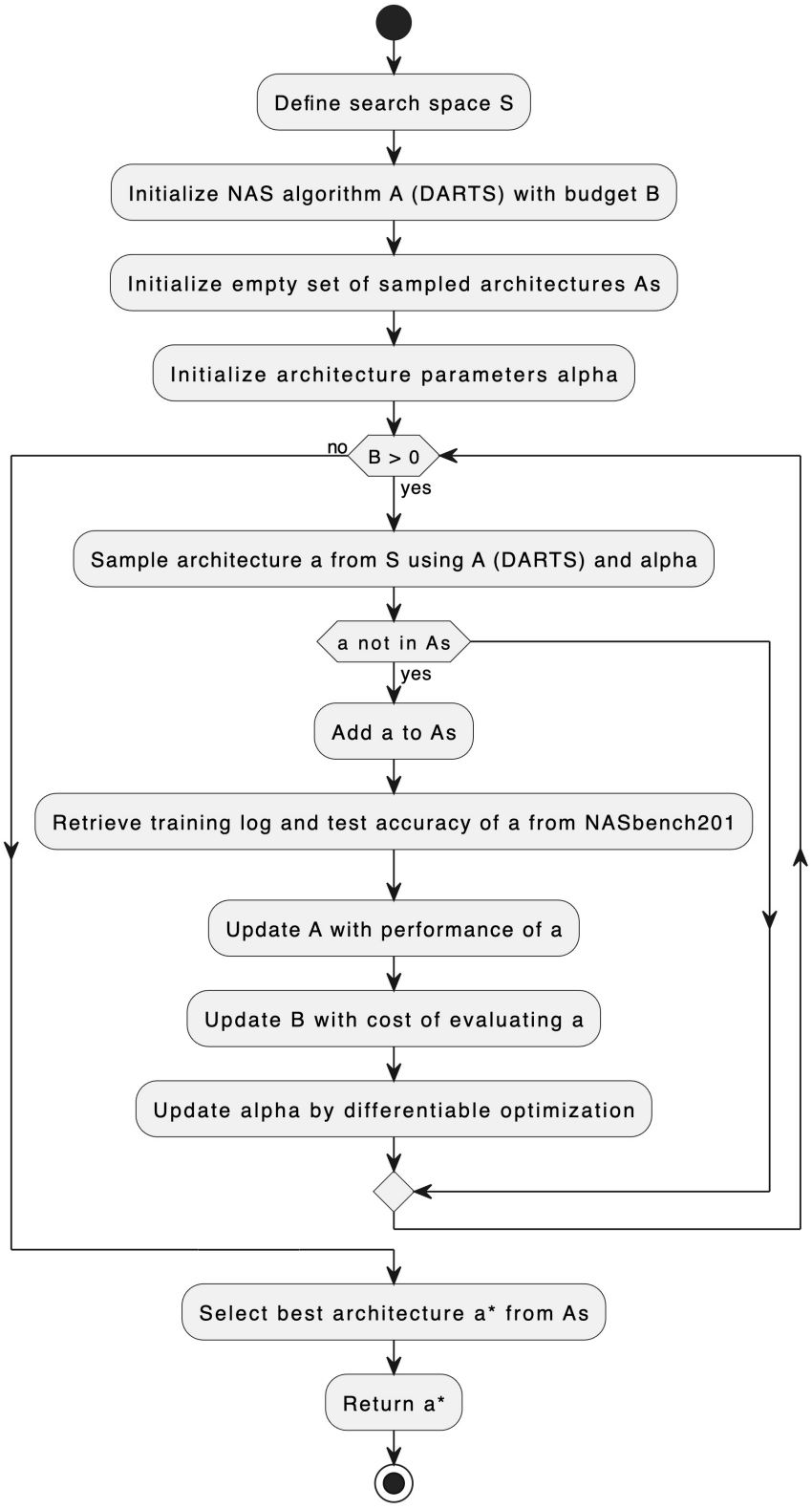
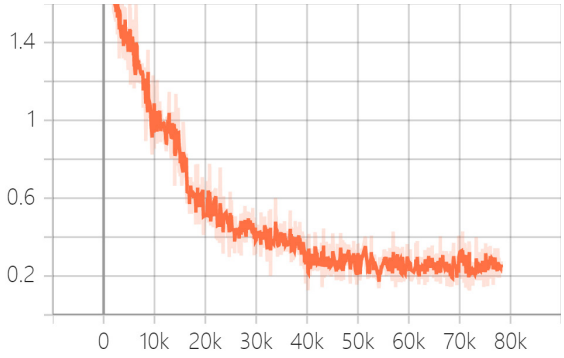


Figure B.1: Flow chart of the combination of DARTS and NASBench 201 for ONN

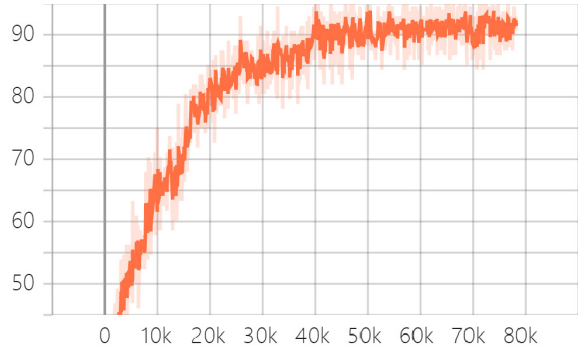
C

APPENDIX C
WELL-KNOWN CNN IMPLEMENTED ON ONN WITH RESNET AS AN EXAMPLE

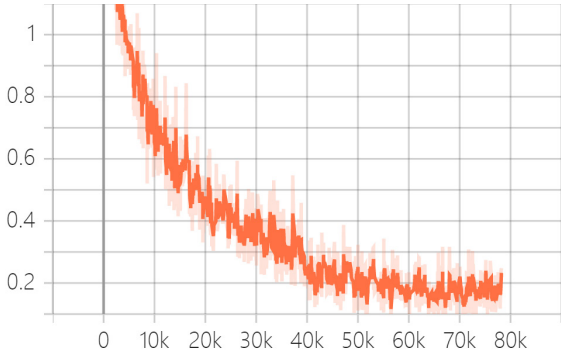
In the main sections, we obtained a finding by comparing NAS search results on ANN and ONN, that there are some similarities in the performance, shows a possibility of transferring high-performance structures from ANNs to ONNs in order to achieve good performance. In this appendix, a ResNet for ONN is created based on the original ResNet for CIFAR from [14] to compare the relation of performance between ONN and ANN. Here 4 ResNet for CIFAR-10 will be discussed including ResNet-20/32/44/56 based on impementation from [59].



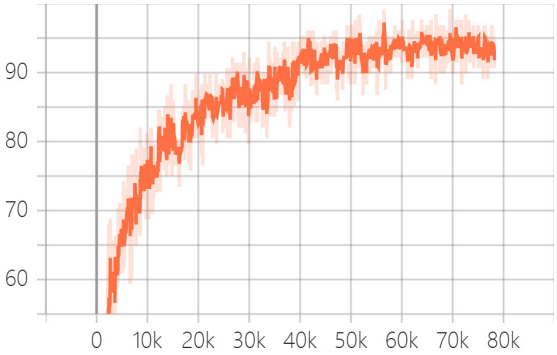
(a) Training Loss ONN ResNet-20
Loss vs. Mini-Batch



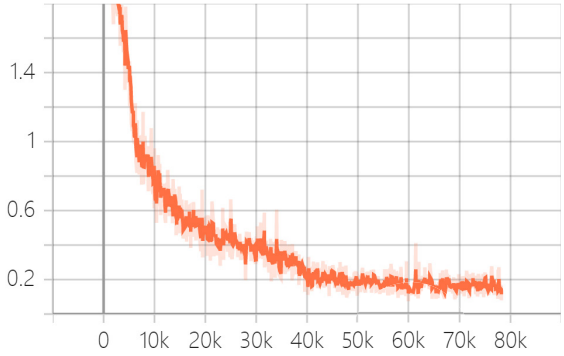
(b) Training Accuracy ONN ResNet-20
Loss vs. Mini-Batch



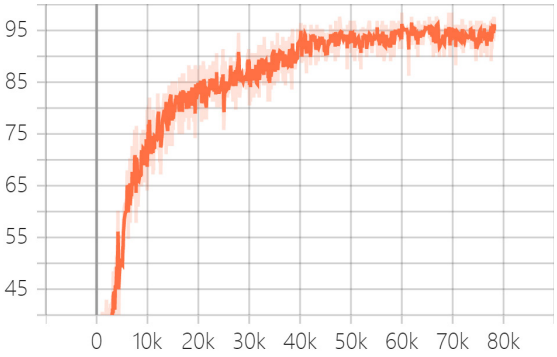
(c) Training Loss ONN ResNet-32
Loss vs. Mini-Batch



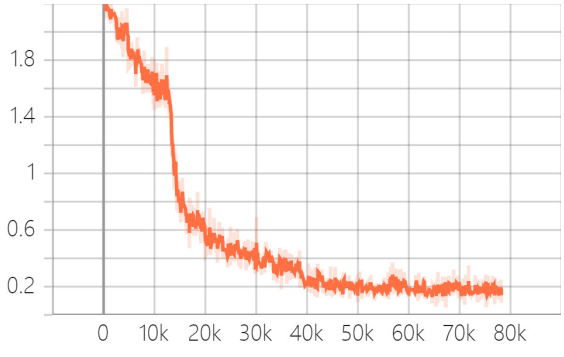
(d) Training Accuracy ONN ResNet-32
Loss vs. Mini-Batch



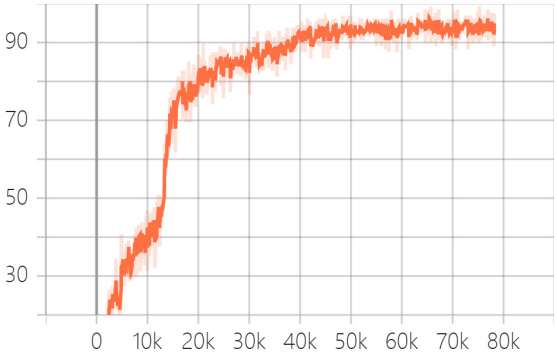
(e) Training Loss ONN ResNet-44
Loss vs. Mini-Batch



(f) Training Accuracy ONN ResNet-44
Loss vs. Mini-Batch

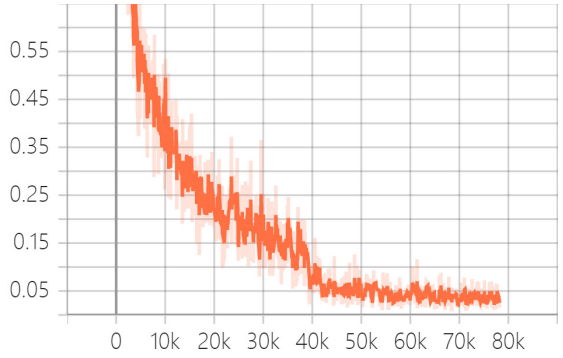


(g) Training Loss ONN ResNet-56
Loss vs. Mini-Batch

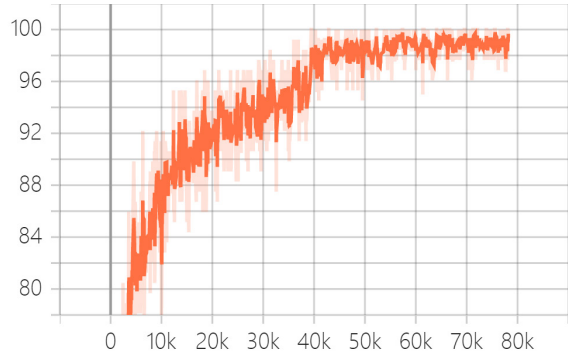


(h) Training Accuracy ONN ResNet-56
Loss vs. Mini-Batch

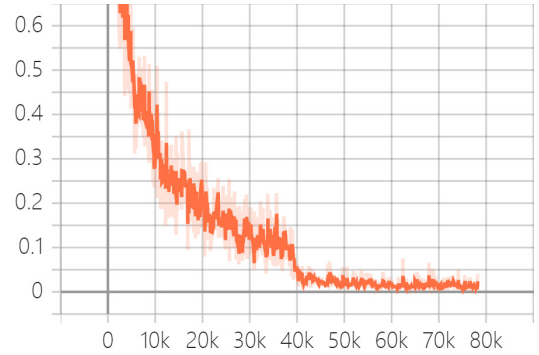
Figure C.1: Result for ONN ResNet on CIFAR-10
(Light Orange: Raw Data, Deep Orange: Moving Average)



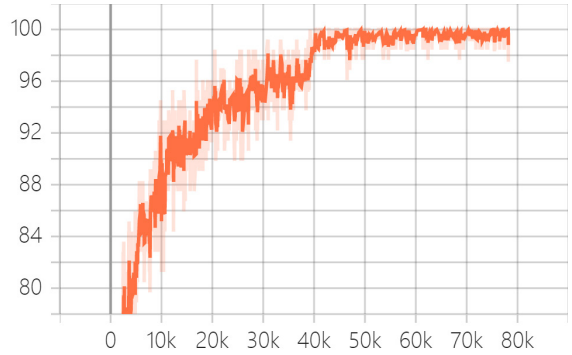
(a) Training Loss ResNet-20
Loss vs. Mini-Batch



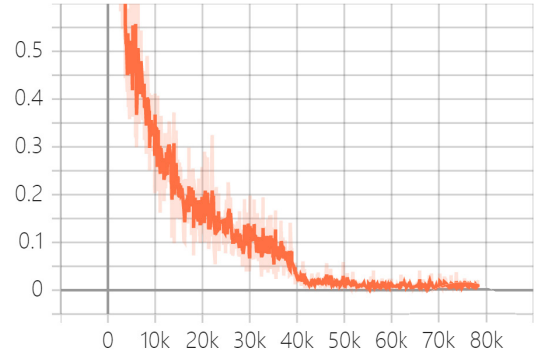
(b) Training Accuracy ResNet-20
Loss vs. Mini-Batch



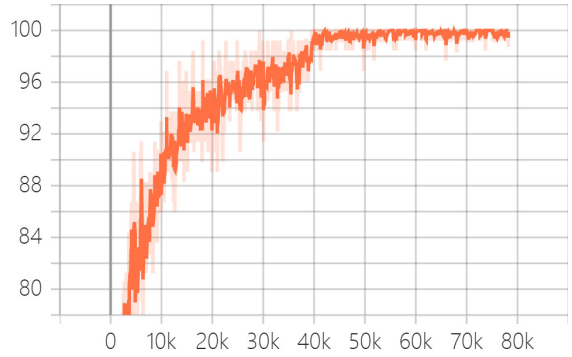
(c) Training Loss ResNet-32
Loss vs. Mini-Batch



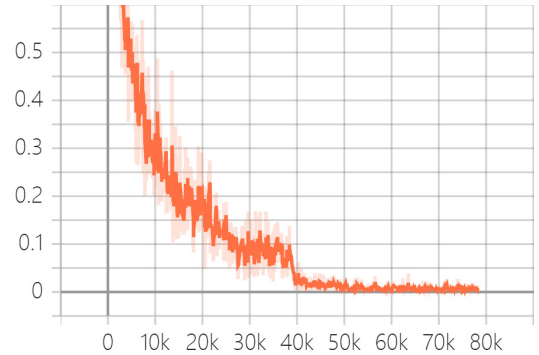
(d) Training Accuracy ResNet-32
Loss vs. Mini-Batch



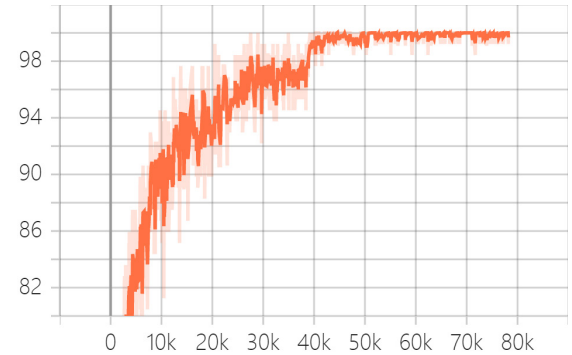
(e) Training Loss ResNet-44
Loss vs. Mini-Batch



(f) Training Accuracy ResNet-44
Loss vs. Mini-Batch



(g) Training Loss ResNet-56
Loss vs. Mini-Batch



(h) Training Accuracy ResNet-56
Loss vs. Mini-Batch

Figure C.2: Result for ANN ResNet on CIFAR-10
(Light Orange: Raw Data, Deep Orange: Moving Average)

	# Layers	Type	Accuracy	Time Consumption
ResNet	20	ANN	89.34	22 min
ResNet	32	ANN	89.76	30 min
ResNet	44	ANN	90.18	41 min
ResNet	56	ANN	90.45	50 min
ResNet	20	ONN	85.81	30 min
ResNet	32	ONN	86.23	46 min
ResNet	44	ONN	86.65	61 min
ResNet	56	ONN	87.45	70 min

Table C.1: ResNet Accuracy on ONN and ANN

A comparison is made between ANN and ONN ResNet structures with 20, 32, 44, and 56-layer as proposed by [14]. For ANNs, Figure C.2 and Table C.1 above demonstrate that with increasing network depth, deeper networks yield lower training errors and improved accuracy. The convergence in the ONN case is slower than that in the ANN case and the loss for ONN is bigger than ANN. Compared to the ANNs, our ONN ResNet shows the almost idential properties to those in ANN ResNet. As expected, there's an accuracy gap between the ANN and the ONN experiment. And the difference in performance and accuracy can be attributed to the unique network characteristics of each system. Also ONN models takes longer time to compute than ANN's. Hence our main conclusion is proved.

Training Architectures

For both ONN and ANN cases, parameters are set into identical. In this work, the batch size is set at 128, which is a fixed parameter. The batch size pertains to the number of training examples utilized in one iteration.

In terms of training duration, the models were trained for 200 epochs, consistent with traditional DARTS, which is a fixed aspect of this approach.

The learning rate, which controls the step size at each iteration while moving toward a minimum of a loss function, is initiated at 0.01 and drops to one-tenth of its previous learning rate at 100 and 150 epochs respectively. The final learning rate is 0.0001. In our case, we find that the initial learning rate of 0.1 from ResNet [14] is too large for ONN to start converging. So we set the initial learning rate to 0.01 for both ANN and ONN cases.

Stochastic Gradient Descent (SGD) is used as the optimizer with a momentum of 0.9 and weight decay of 0.0001. These are also fixed parameters.