Differentiable Neural Architecture Search through Hypernetworks for Image Recognition

ENG4702: Final Year Project - Final Report

Author: Mah Yuen Yee (30213673)
Supervisor: Dr. Mohamed Hisham Jaward, Dr Liang Shiuan-Ni
Date of Submission: 26 May 2023
Project type: Research

Executive Summary

Deep learning has shown exceptional performance in solving computer vision tasks. However, traditional deep learning neural networks require researchers to manually design the architectures. The manual architecture design process is time-consuming and error-prone. Therefore, neural architecture search (NAS), which automates the architecture design process, has grown in importance. Differentiable neural architecture search (DAS) is a major field in NAS that employs gradient-based strategies to search for the optimal architectures. DARTS (Differentiable Architecture Search) is the first DAS model that shows state-of-the-art performance in finding optimal architectures by relaxing the discrete architecture search space to be continuous. DARTS maintains its pioneering status in the field of DAS even in the present-day, as most state-of-the-art DAS models developed are based on DARTS.

A DAS model typically frames the architecture search problem as a bilevel optimization problem (BLO). BLO involves updating architecture parameters (hyperparameters) using the gradients of the validation loss with respect to the hyperparameters (hypergradients). Hypergradient computation is expensive, and DARTS circumvented the issue by using a one-step unrolling scheme to approximate hypergradients, resulting in approximation error.

Therefore, this project aims to improve DARTS performance in finding optimal architectures by developing a new DARTS-based model called hyperDARTS. HyperDARTS should be able to find architectures with better image recognition capability. HyperDARTS changes the strategy employed by DARTS for architecture search to remove the hypergradient approximation scheme. HyperDARTS uses hypernetworks as a search strategy. A hypernetwork is a neural network that generates weights for another neural network. Through this method, the validation loss becomes a function of hypernetworks, which are functions of the hyperparameters. Therefore, the hypergradients can now be computed inexpensively using the popular algorithm: backpropagation.

This project has developed two hyperDARTS models: hyperDARTS-1 and hyperDARTS-2. The hypernetworks in hyperDARTS-1 and hyperDARTS-2 are linear regression and a 2-layer multilayer perceptron (MLP) with a sigmoid activation function, respectively. Two image recognition datasets (CIFAR-10 and CIFAR-100) are used in this project to evaluate the image recognition capability of architectures discovered by the DAS models.

The DARTS-discovered architecture achieved testing accuracies of 97.24% and 82.30% on CIFAR-10 and CIFAR-100, respectively. The HyperDARTS-1-discovered architecture achieved testing accuracies of 97.40% and 83.38% on CIFAR-10 and CIFAR-100, respectively. The HyperDARTS-2-discovered architecture achieved testing accuracies of 97.46% and 83.61% on CIFAR-10 and CIFAR-100, respectively.

Overall, the architectures discovered by both hyperDARTS models achieved better image recognition-based task performance than the DARTS-discovered architecture, demonstrating the superiority of using hypernetworks as a search strategy over the unrolling scheme for a DAS model.

Contents

1 Introduction

Deep learning has demonstrated exceptional performance in solving perceptual tasks such as machine translation [1–3], image recognition [4–6], and object detection [7–9], as illustrated in Fig. 1. This success is primarily attributed to the strong learning capabilities of deep learning methods, particularly their ability to automatically extract features from unstructured data.

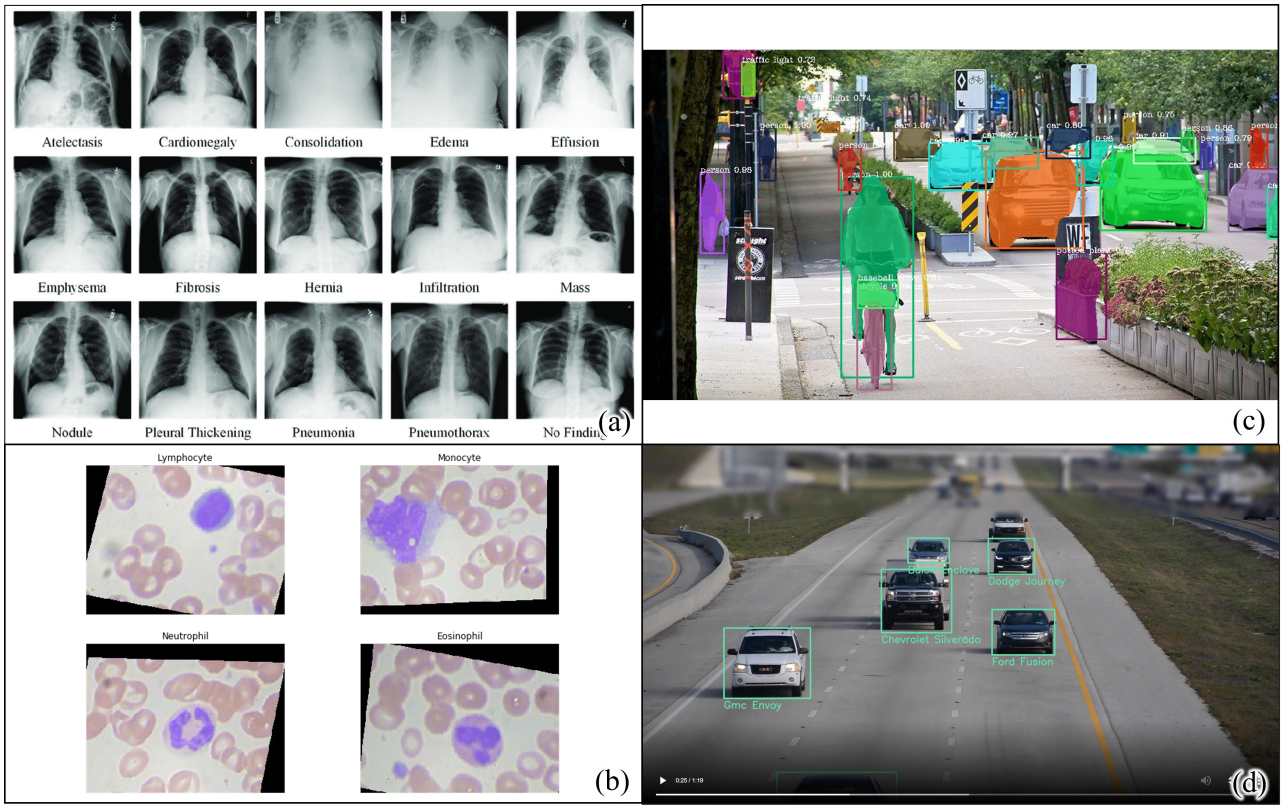


Fig. 1. Some examples of deep learning application. Image recognition for (a) thorax disease diagnosis [10, 11] and (b) blood cell classification [12]. Object detection for (c) instance segmentation of videos [13] and (d) vehicle detection and recognition [14].

By automating feature engineering for unstructured data, which was once a laborious manual work in traditional machine learning [15, 16], deep learning frees up researchers to focus on designing and engineering network components and neural architectures [17–19]. An example of the architecture of a modern deep learning neural network is shown in Fig. 2.

While effective architecture alterations do enhance the performance of deep learning methods, the manual process of searching for these enhanced architectures poses several challenges. First, it is a laborious, time-consuming, and error-prone task that requires substantial expertise and prior knowledge. Consequently, it is difficult for practitioners and newcomers to customize emerging neural architectures to their specific needs [21]. In addition, heavy reliance on experts' prior knowledge may introduce human bias and fixed thinking paradigms to architecture design, partly limiting the discovery of novel architectures [22], as illustrated in Fig. 3. The component of manually-designed ShuffleNet [23] shown in Fig. 3 (a) has less flexible micro-architecture (structure) while the component of automatically-discovered DARTS [24] shown in Fig. 3 (b) has more flexible micro-architecture.

Moreover, neural architectures have been evolving and becoming increasingly complex to meet the high expectations of performance, further hardening the manual architecture search process for performance improvement. These limitations of manual architecture search have sparked an increase in research

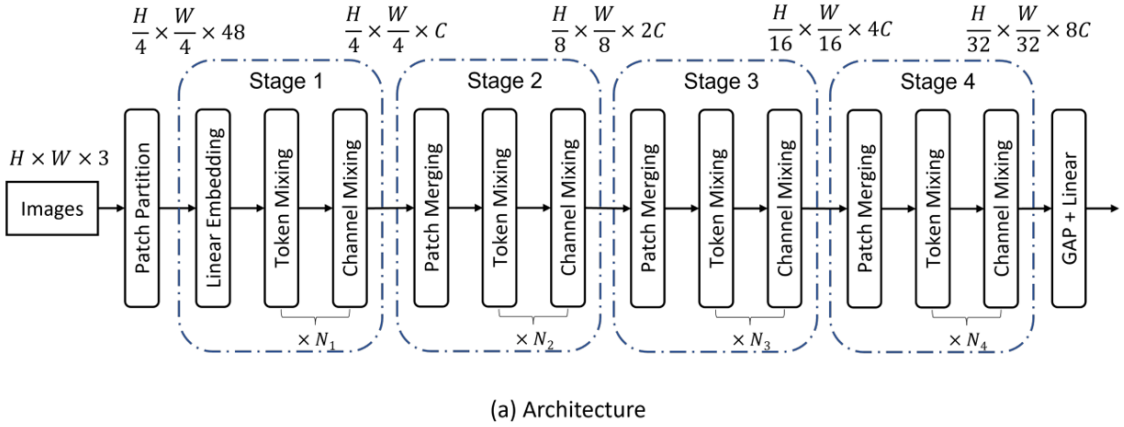


Fig. 2. Many modern, effective deep learning neural networks, especially for image recognition tasks, use only a few different types of motifs or cells within their architectures by stacking the cells repeatedly. An example, the overall architecture of sMLPNet [20], a modern image recognition neural network, is shown here. Each rectangular box with a black border here represents a cell. Each cell has input node/s and output node/s and employs a set of sequential operations in between, as exemplified in Fig. 3. There are 3 repeating cell types in sMLPNet: patch merging, token mixing, and channel mixing. What each type of cell does is not important in this section.

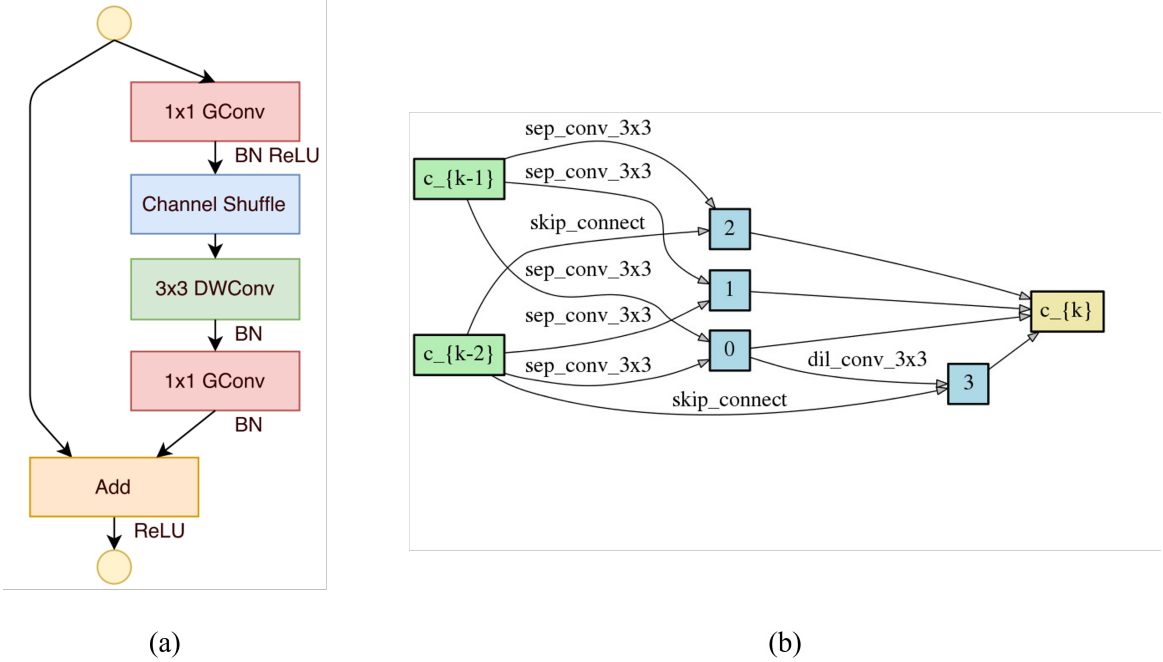


Fig. 3. A cell is a repeating component in neural networks as explained in Fig. 2. The micro-architecture of a cell type used in ShuffleNet [23], which was manually designed. (b) The micro-architecture of a cell type used in DARTS [24], which was found using NAS and achieved a lower test error rate on the ImageNet dataset with a similar number of parameters (weights) and floating point operations per second (flops) compared to ShuffleNet. The cell micro-architecture designed by NAS in (b) appears to have more flexibility and be subject to less fixed thinking paradigms and more creative. What the operations used in the cells do are not important in this section.

efforts in recent years seeking to automate this search process [17–19, 24–26]. As a result, research on the problem of automatic search for optimal architectures known as **Neural Architecture Search** (**NAS**) has grown in importance.

NAS aims to design a neural architecture that maximizes task performance utilizing limited computing resources in an automated manner with minimal human intervention [27, 28]. Arguably, NAS has been proven feasible by two pioneering works in the field: NAS-RL [17] and MetaQNN [18]. These neural architectures, discovered in automated search using reinforcement learning (RL), have achieved state-of-the-art classification accuracy for image classification problems. The viability of NAS has been further verified by the later work of Large-scale Evolution [29], which uses an evolutionary algorithm (EA) to obtain similar accuracy. However, these search methods consume enormous computation resources: hundreds of GPU days or even more.

Consequently, a large body of work has been conducted to reduce the amount of calculation and to speed up the neural architecture search process [19, 30, 31]. Notably, Cai et al. [32] and Pham et al. [19] have managed to reduce the search cost to a few GPU days by leveraging the principle of reuse of learned model parameters, i.e., weight sharing. As search efficiency has improved, NAS methods have also been extensively applied in various fields. NAS has outperformed manually designed architectures on some tasks in the fields of image classification [25], semantic segmentation [33], and boundary detection [34]. NAS has also achieved highly competitive performance comparable to state-of-the-art handcrafted architectures on some tasks in the fields of object detection [35–37], data augmentation [38, 39], architectural scaling [40–42], speech recognition [43], and adversarial learning [44].

NAS frames a search problem to automate the search for optimal architectures, by using a search strategy to explore a predefined finite architecture search space, evaluating and comparing candidate architectures based on task performance [17–19, 32] as illustrated in Fig. 5. In other words, for a typical NAS problem, researchers need to decide how the search problem is formulated, the search space, the search strategy, and the performance evaluation strategy for candidate architectures. For instance, DARTS [24], a differentiable NAS method, formulated a bilevel optimization problem (search problem) that finds the best operations and their placement within a cell, as shown in Fig. 3(b), from only 8 types of operations among other constraints (search space) using a gradient-based search strategy and a gradient-based weight sharing mechanism between all its candidate architectures (performance evaluation strategy). The operations are weighted and the operation weightages are parameters to be learnt. The operation weightages are therefore a type of **hyperparameters $\boldsymbol{\lambda}$** since they control the candidate architecture.

This project focuses on further improving the performance of a popular NAS model in searching optimal architectures, by integrating one of the latest hyperparameter optimization method, i.e., **hypernetworks** [45, 46] as illustrated in Fig. 4, into the **search strategy**. A hypernetwork ϕ is a neural network that generates the weights $\boldsymbol{w}$ for another neural network, termed elementary network. Hypernetworks allow hyperparameters to be optimized using gradient descent and have shown promising results for the application of hyperparameter optimization [45]. After finding optimal architectures using the newly-developed NAS model, the task performance of these found architectures will be evaluated based on image classification or recognition tasks.

Image classification tasks are used for evaluating discovered architectures in this project because it presents a challenging benchmark due to the considerable manual architecture engineering that has been devoted to this field [4–6]. In other words, if the newly developed NAS model can find architectures that perform well in image classification tasks, it is also likely that, after model adaptation, the NAS model can find architectures that perform well in other tasks such as natural language processing [24], object detection [47], and adversarial learning [44]. In addition, as NAS still incorporates human prior knowledge in some aspects of its search problem, especially the search space, it is some-

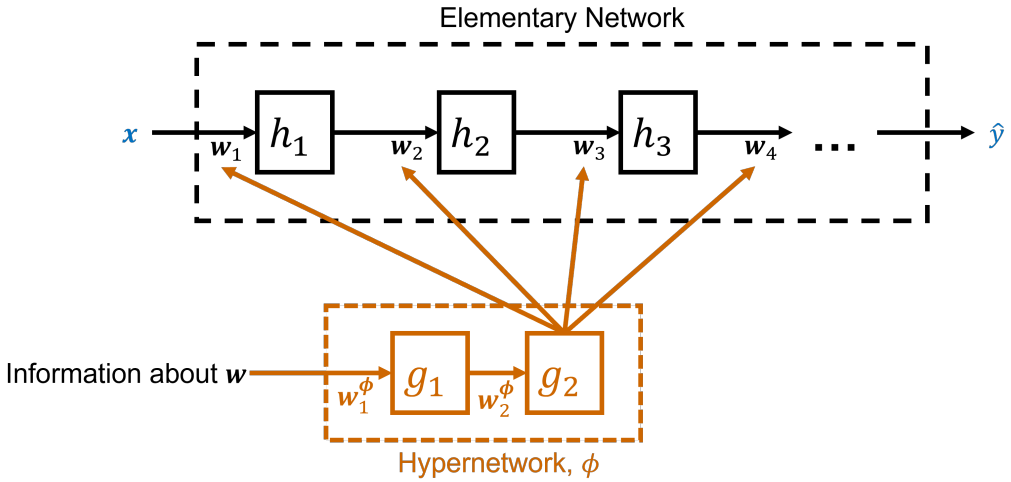


Fig. 4. A hypernetwork ϕ is a neural network that generates the weights $\boldsymbol{w}$ for another neural network, termed elementary network. ϕ has its own weights $\boldsymbol{w}^\phi$. Black components belong to the elementary network while brown components belong to the hypernetwork. A neural network has 3 types of layers: input layer, hidden layer, and output layer. A hidden layer (h in the elementary network and g in ϕ) is a feature vector or a feature map. The input layer $\boldsymbol{x}$ is the input data while the output layer $\hat{y}$ is the output prediction.

what straightforward to define a suitable search space for the image classification problem by utilizing knowledge from the vast source of manual engineering.

Gradient-based search and architecture performance evaluation strategies have been used extensively in NAS problems. They elegantly condense NAS to a bilevel optimization problem in which architecture parameters (also known as hyperparameters in NAS) and candidate architecture weights can be jointly optimized using the same method (gradient descent) without any extra functional blocks [24, 48–50]. However, the current mainstream gradient-based search strategy proposed by DARTS employs an approximate scheme to compute hyperparameter gradient (hypergradient) due to the indirect relationship between hyperparameters and the loss function [24].

This project proposes to replace the mainstream gradient-based search strategy that involves an approximation scheme with hypernetworks (also a gradient-based search strategy) to eliminate the approximation errors. Additionally, a hypernetwork is essentially a small neural network to generate weights for a larger neural network [46]. Therefore, integrating hypernetworks into the NAS search strategy opens up the possibility of further improving NAS performance in the future by leveraging advancements in traditional neural networks to improve hypernetwork capability. The details of gradient-based search strategies are discussed in Section 3.2.2 and Section 3.4 while the details of hypernetwork are discussed in Section 4.4.2.

The most common terms and their meaning are listed in Table 25.

2 Aims and Objectives

2.1 Hypothesis

2.1.1 Background

NAS was first made possible by using RL as the search strategy, which, however, consumes up to thousands of GPU days for searching [17, 18]. Differentiable architecture search (DAS) [24] is the first end-to-end NAS model that manages to reduce the search cost by three orders of magnitudes to a few GPU days while maintaining high test error performance by using a gradient-based search strategy, as discussed in Section 3.2.2. DARTS [24] was the first DAS model to search the entire architecture instead of only some specific parts of the architecture [51, 52].

Since the development of DARTS, the field of DAS has been proliferating and is now one of the reliable NAS methods to achieve state-of-the-art performance on finding optimal architectures [26]. Nevertheless, most recent DAS models still use DARTS as the preliminary or foundational model [48, 53–55], proving the effectiveness and fundamental importance of DARTS in the field of DAS. Therefore, this project of developing a new DAS model follows the prevailing and promising trend of using DARTS as the preliminary model.

DARTS [24] employs unrolling method with only **one** inner unrolling and backpropagating step when approximating hypergradient to update and optimize hyperparameters during cell search to minimize search cost in GPU days. There is no theoretical convergence guarantee of the greatly abbreviated optimization algorithm. Nonetheless, DARTS viability has been proved experimentally as DARTS managed to find architectures that achieve competitive test error performance (evaluated on image recognition and natural language processing tasks) with only two GPU days.

2.1.2 Hypothesis

It is hypothesized that while DARTS can find architectures, with low computational resources, that achieve competitive task performance in terms of test error, its abbreviated approximating optimization algorithm reduces itself to only converge to suboptimal architectures during architecture search. In other words, there is room to improve DARTS performance in terms of computational resources for searching and task performance of discovered architectures by replacing the one-step unrolling hypergradient approximation scheme with a more efficient and stable hyperparameter optimization method.

2.2 Aims

This project aims to develop a new NAS model based on a cutting-edge DAS model, i.e., DARTS, by integrating one of the latest hyperparameter optimization algorithms, i.e., hypernetworks, into the search strategy to improve its performance on finding optimal architectures. The architecture optimality is evaluated based on the architecture performance on image classification tasks.

2.3 Objectives

1. To develop a new NAS model called **hyper**network-**DARTS** (**hyperDARTS**) by replacing DARTS search strategy with hypernetworks, removing the current one-step unrolling hypergradient approximation scheme, to show improvements in finding optimal architectures.
2. To search for architectures on a general image classification dataset, CIFAR-10 using hyperDARTS.
3. To analyze hyperDARTS searching performance, including search cost (time taken for searching), the discovered architectures, and the validation accuracies achieved during searching, to determine the optimal architecture.
4. To evaluate the task performance of architectures discovered by hyperDARTS by performing image recognition task on CIFAR-10, using test accuracy as the metric.
5. To evaluate the task performance and transferability of hyperDARTS-discovered architectures by performing image recognition task on CIFAR-100, a dataset not used in architecture searching, using test accuracy as the metric.
6. To compare the task performance, in terms of test accuracy, achieved by hyperDARTS-discovered architectures on CIFAR-10 and CIFAR-100 with that of DARTS.

3 Literature Review

As discussed in Section 1, the design of NAS primarily involves formulating a search problem, defining a search space, devising a search strategy, and implementing a performance evaluation strategy for candidate architectures. The performance of candidate architectures is usually estimated during searching to reduce computational costs [24, 56, 57]. Therefore, the performance evaluation strategy is also known as the performance estimation strategy. Search problem and search strategy are closely intertwined and often discussed together, given their heavy interdependence [26, 58].

Consequently, this section divides NAS design into three main dimensions: **search space**, **search strategy** (containing both search problem and search strategy aforementioned), and **performance estimation strategy**, as illustrated in Fig. 5.

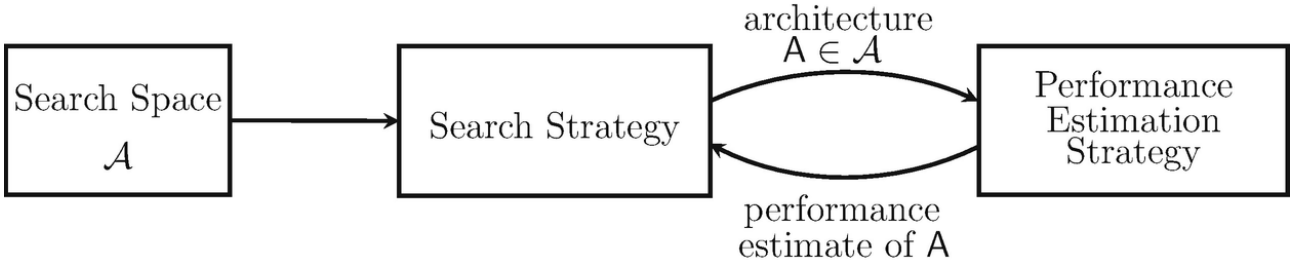


Fig. 5. An abstract illustration of NAS methods [58]. A search strategy chooses an architecture A from a predefined search space $\mathcal{A}$. A is then passed to a performance estimation strategy, which outputs the estimated performance of A to the search strategy.

- **Search Space.** The search space, which defines all possible discoverable architectures, is determined by the predefined operation set and constraints imposed on the possible or searchable architectures such as how operations can be connected. There is a trade-off between the search space size and minimal human intervention: incorporating prior knowledge about typical properties of effective architectures can reduce the search space size but also introduces human bias, which hinders novelties beyond human imagination. The larger the search space, the higher the search cost.
- **Search Strategy.** The search strategy determines how to explore the search space. It incorporates the typical exploration-exploitation trade-off: quick discovery of effective architectures is desirable, but this may indicate a high possibility of premature convergence to suboptimal architectures. Search strategy also determines how the search problem can be formulated.
- **Performance Estimation Strategy.** The main goal of NAS is to discover architectures that show excellent predictive performance on unseen data. Performance Estimation is the process of estimating this performance of candidate architectures during architecture search. Intuitively, the most direct approach is to train and validate candidate architectures using standard methods (until convergence) on data. However, this is exceptionally computationally expensive due to the significant number of searchable architectures. Therefore, reducing the performance estimation costs is one of the focuses of recent studies.

Convolutional neural networks (CNNs) have played a pivotal role in achieving remarkable advancements in deep learning for nearly all computer vision tasks, including image recognition [4–6]. Consequently, this section focuses on NAS methods that explore architectures based on CNNs. This review is structured into four main subsections: **search space**, **search strategy**, **performance estimation strategy**, and **bilevel optimization**. Bilevel optimization is the most prevalent search problem formulated for gradient-based search strategies.

3.1 Search Space

In the language of computational graphs [59], a neural network is a directed acyclic graph (DAG) with a set of nodes Z as exemplified in Fig. 6. Each node $\boldsymbol{z}^{(k)}$ is a tensor (a multidimensional array, e.g., a feature map in CNNs) with an operation (directed edge) $o^{(j,k)} \in O$ on each of its parent nodes $i^{(j,k)} \in I^{(k)}$. In simple terms, each node is the aggregation f results of operations acted on its parent nodes. f is usually summation [23, 24] or concatenation in the channel dimension [25, 60] in CNNs. The input node is an exception that has no parent nodes as it equals the input data $\boldsymbol{x}$. The output node equals the output prediction $\hat{\boldsymbol{y}}$ of a task defined. The computation at a node k is defined as:

$$\boldsymbol{z}^{(k)} = f(o^{(j,k)}(\boldsymbol{i}^{(j,k)})) \tag{1}$$

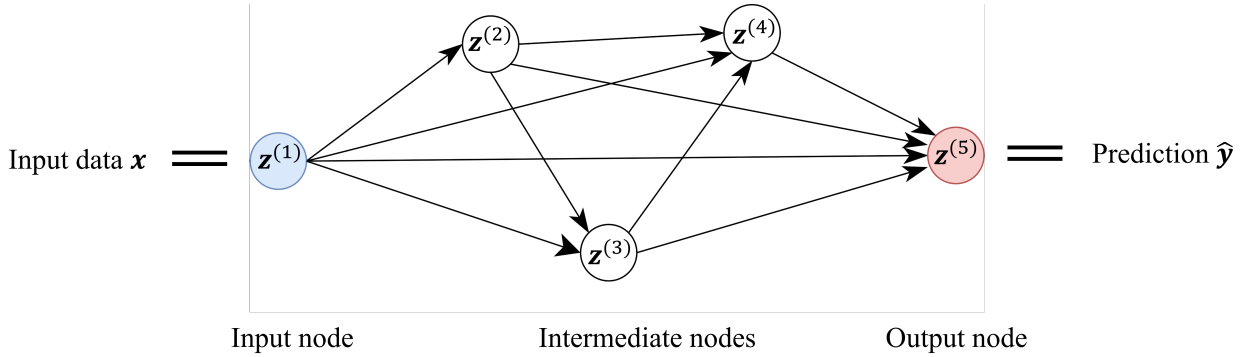


Fig. 6. An example of a DAG with 5 nodes. Node $\boldsymbol{z}^{(1)}$ is the input node which equals the input data $\boldsymbol{x}$. Node $\boldsymbol{z}^{(5)}$ is the output node which equals the output prediction $\hat{\boldsymbol{y}}$. Each intermediate and output node has incoming directed edges from all its predecessors, i.e., parent nodes. During training, training data is used as the input data $\boldsymbol{x}$ while the output prediction $\hat{\boldsymbol{y}}$, of e.g., a image recognition task, is used to compute the training loss. During validation, $\boldsymbol{x}$ is validation data while $\hat{\boldsymbol{y}}$ is used to compute the validation loss. During testing where the actual output $\boldsymbol{y}$ is never used for training nor hyperparameter optimization, $\boldsymbol{x}$ is testing data and $\hat{\boldsymbol{y}}$ is the output prediction.

The operation set O can include unary operations such as pooling, convolutions, and activation functions, multivariate operations such as addition and concatenation, or a sequential flow of operations. The network architecture is entirely defined by a representation that specifies all the operations between each node and its set of parents. Search space is a subspace of this general definition of neural architectures constrained by the pre-defined operation space and other imposed constraints. The search space limits the number of possible solutions (searchable architectures) to the search problem for the search strategy to solve. There are two main types of search space: **global** and **modular**.

3.1.1 Global Search Space

The simplest global search space is the chain-structured search space, as illustrated in Fig. 7 (left). The most primitive form consists of architectures represented by a sequence of ordered nodes, with each node $\boldsymbol{z}^{(k)}$ having only one parent $\boldsymbol{z}^{(k-1)}$. The search space is normally parameterized by:

1. the number of layers n;
2. operation for each layer, e.g., convolution, pooling, or advanced convolution such as dilated convolution [61] or depthwise separable convolution [60];

3. operation-dependant hyperparameters such as number of kernels, kernel size, and stride for a convolutional layer [31, 32, 62].

Since 3. is conditioned on 2., the search space parameterization is conditional and variable instead of having a fixed length. Practical design elements from modern hand-crafted architectures are incorporated in recent work on NAS [56, 63]. Skip connections [64], a notable example, allow the construction of complex, multi-branch architectures as shown in Fig. 7 (middle).

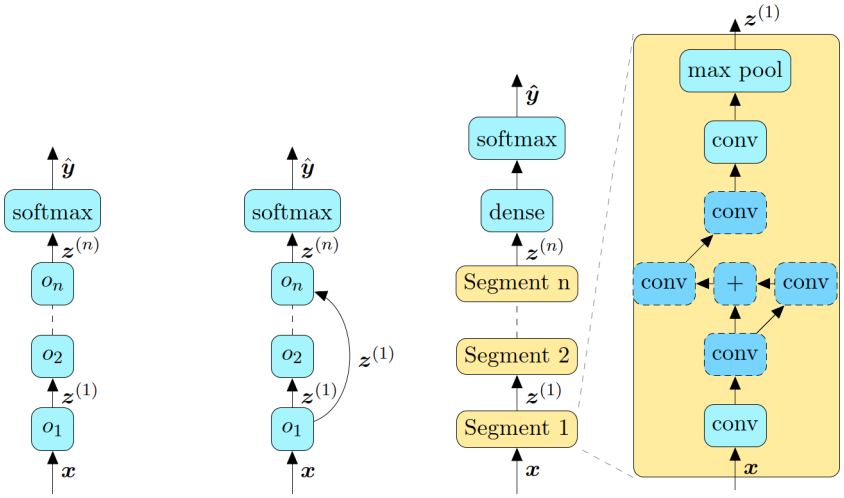


Fig. 7. Three main types of global search space: chain-structured (left) [31], with skips (middle) [17], and architecture template (right) [65].

3.1.2 Modular Search Space

Motivated by effective handcrafted architectures that consist of repeated motifs [64, 66, 67] as exemplified in Fig. 2, these motifs, also known as cells or building blocks, are searched for instead of the entire architecture [56, 68], dubbed modular search space. The modular search space is used to optimize two kinds of cells: a normal cell that preserves spatial dimensionality and a reduction cell that reduces spatial dimension (discussed in Section 4.3). The cells are first stacked as a smaller macro-architecture as exemplified in Fig. 8 (left) for architecture search. Note that the same type of cells contains the same hyperparameters (operation weightages) during architecture search, they only differ in operation weights. After architecture search, the cells are extracted and then stacked as a larger macro-architecture to form the final architecture, as exemplified in Fig. 8 (right). Modular search space **reduces architecture search to cell search**.

3.1.3 Global vs. Modular Search Space

Global search space allows a greater degree of freedom on operation arrangement. Among the three types of global search space shown in Fig. 7, the architectural template type has the greatest degree of freedom. The architectural template defines admissible structural choices per the architecture definition and needs. A typical example, as illustrated in Fig. 7 (right), divides architecture into segments and enforces different operation and connection constraints on different segments.

Modular search space is a special case of global search space, with its architectural template being a repeating pattern of subgraphs. In other words, there are only one or two types of segments (also known as cells) that need to be searched in modular search space as shown in Fig. 8 (left). Despite

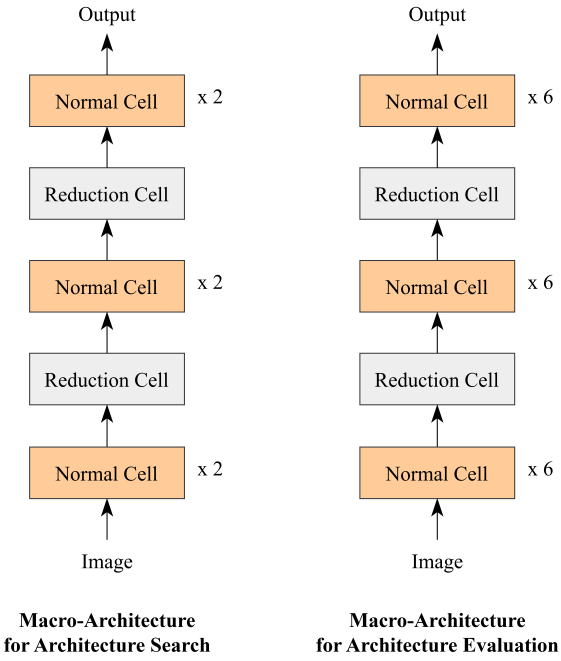


Fig. 8. Predefined macro-architectures stack cells for architecture search and architecture evaluation [56]. The cells are first stacked as a smaller macro-architecture (left) for **architecture search**. Note that the same type of cells contains the same hyperparameters (operation weightages) during architecture search, they only differ in operation weights. After architecture search, the cells are extracted and then stacked as a larger macro-architecture (right) to form the final architecture for architecture evaluation. (right). Modular search space **reduces architecture search to cell search**.

a higher degree of human intervention in deciding the predefined macro-architecture, the modular search space has three major advantages resulting in better performance:

1. As cells are shallower, i.e., have fewer layers than a whole architecture, the search space size is reduced. Zoph et al. [56] showed a seven-times computation acceleration by transforming global search into modular search, which also results in a lower test error rate.
2. Architecture constructed from cells can be easily adapted to other datasets by changing the number of cells and filters. Indeed, transferring cells optimized on CIFAR-10 to ImageNet is a common and effective method to achieve cutting-edge performance in recent studies [48, 54, 56] as exemplified in Fig. 8.
3. Repeating building blocks have proved to be an effective design in handcrafted architectures for image recognition tasks [4, 64].

Due to the reduced computational cost and enhanced performance, modular search space for NAS is used in this project. Therefore, the two terms **architecture search** and **cell search** are used interchangeably in this report.

3.2 Search Strategy

There are five main types of search strategy that can be further classified into two categories: discrete, including **random search** (**RS**), **Bayesian optimization** (**BO**), **reinforcement learning** (**RL**), and, **evolutionary algorithm** (**EA**), and continuous, including **gradient-based** methods, also known as **differentiable architecture search** (**DAS**).

3.2.1 Discrete Search Strategy

NAS has only become a mainstream machine learning research topic after Zoph et al. [17] achieved state-of-the-art performance on CIFAR-10 and Penn Treebank benchmarks by framing NAS as a **reinforcement learning (RL)** problem. The agent's action is the generation of architecture, with the action space being search space and the reward being estimated performance (see Section 3.3). Different approaches to the agent's policy, e.g., recurrent neural network policy [17, 56] and Q-learning policy [18], have emerged. An alternative RL framework is sequential decision processes that generate the architecture sequentially [32]. **Random search** (**RS**) has proved to be a strong baseline [69, 70], but there is inadequate research to show that it can consistently perform better than the other strategies in any certain tasks.

Evolutionary algorithms (**EAs**) evolve a population of models, i.e., networks, by sampling at least one of them to serve as a parent that produces mutated offspring. Mutations are local operations, including adding skip connections, adding or removing a layer, and altering training hyperparameters. The fitness of offspring, i.e., validating performance, is evaluated before being added to the population. Different approaches of sampling parents (e.g., tournament selection [29, 71], and multi-objective Pareto [72]), removing individuals from a population (e.g., the worst [29], the oldest [73], and nobody [25]), and generating offspring networks (e.g., random initialization [29], and Lamarckian inheritance [63]) are employed to develop NAS.

Bayesian optimization (**BO**) has been applied to NAS using classic Gaussian Process-based BO methods [57, 74], tree Parzen estimators [75], and random forests [76]. These attempts to search high-dimensional conditional spaces have achieved highly competitive results on different tasks by jointly optimizing architectures and hyperparameters [77–79].

3.2.2 Continuous Search Strategy

DARTS [24] is the pioneer in NAS to successfully relax discrete search space to become continuous, which makes the employment of **gradient-based** optimization for architecture search (or cell search) possible. NAS models based on gradient-based optimization for cell search are sometimes referred to as **differentiable architecture search** (**DAS**).

Each cell is seen as a DAG containing nodes in the modular search space. It is worth noting that the DAG representation here differs from what has been discussed in Section 4.3. A DAG represents a cell here but a complete neural network there. An edge between each pair of nodes are the weighted candidate operations. The weightage of each operation in an edge is represented by a hyperparameter λ. Each node is the sum of outputs from all edges directed to it, i.e., the sum of outputs from weighted operations operated on its parent nodes. The continuous relaxation of candidate operations evolves the two-stage NAS problem into one single optimization problem for an asset of continuous variables $\boldsymbol{\lambda}$ as illustrated in Fig. 9. Discretization is performed after the optimal $\boldsymbol{\lambda}$ is obtained to decide the final cell. Finally, cells are stacked in a predefined manner to build the final neural architecture, as illustrated in Fig. 8. The detailed methods to realize DARTS are discussed in Section 4.4.1.

Variants of DARTS have been developed to resolve shortcomings of DARTS. I-DARTS [48] associates each node with a softmax considering all input edges, as shown in Fig. 10 to reduce the locality of DARTS. P-DARTS [49] uses progressive search to gradually increase the network depth during the search phase in contrast to DARTS, which only optimizes a shallow architecture as exemplified in Fig. 8 (left) instead of the actual full-depth architecture as exemplified in Fig. 8 (right) during searching. GDAS [50] samples one subgraph in the cell DAG rather than the entire cell DAG in one iteration

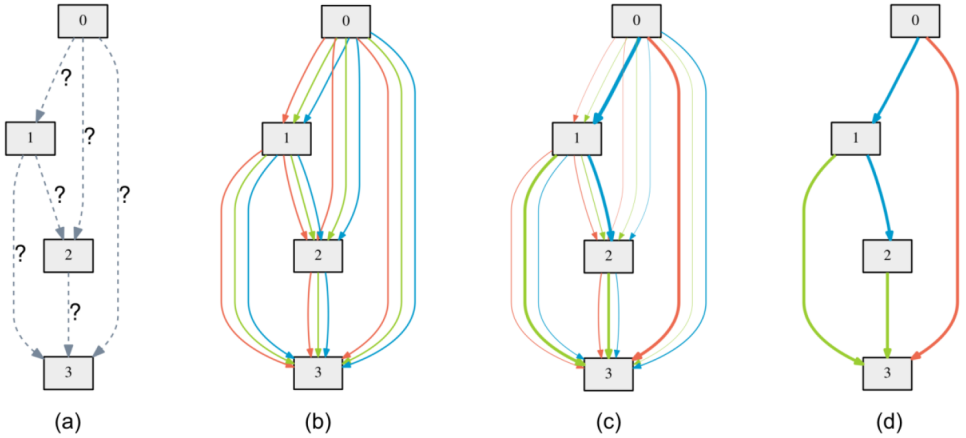


Fig. 9. Cell search strategy of DARTS [24]. (a) Operations on the edges are unknown at first. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. An edge is a set of (all the) arrows pointing from one node to the other rather than one of the arrows. In other words, there are 6 edges here. (c) Joint optimization of $\boldsymbol{\lambda}$ and the network weights as a bilevel optimization problem. During optimization, $\boldsymbol{\lambda}$ is constantly updated so each edge has high and low λs represented by the high and low intensity arrows. (d) Discretization of final cell architecture based on learnt $\boldsymbol{\lambda}$. Only largest λs are kept. Note: Nodes are in rectangular grey boxes. Each colour in line arrows represents a specific operation type from the predefined operation set, such as 3×3 separable convolution.

as illustrated in Fig. 11, improving search efficiency. β-DARTS [80] regularizes the hyperparameters after softmax rather than before softmax (see Section 4.3 regarding the softmax implementation for hyperparameters), improving the robustness and generalization capability of discovered architectures. SmoothDARTS [81] employs perturbation-based regularization to increase the stability.

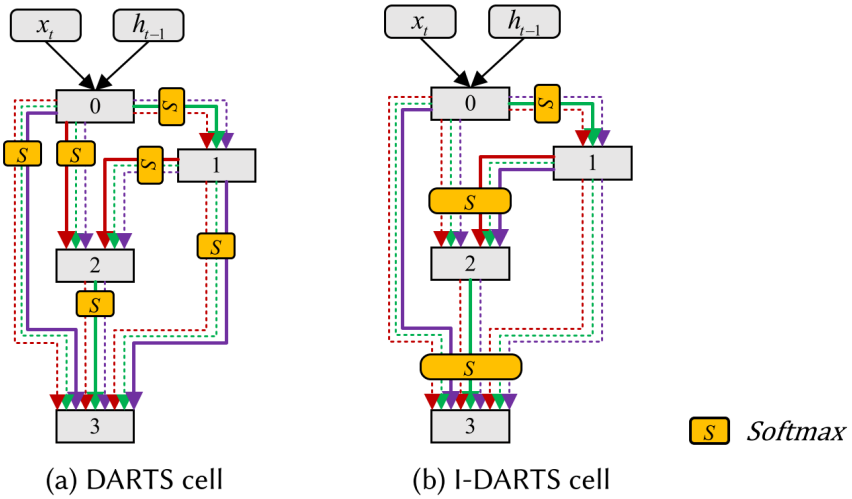


Fig. 10. Search strategy comparison between DARTS and I-DARTS [48]: (a) Softmax is performed on each outgoing edge. (b) Softmax is performed on all incoming edges of a node. Note: Nodes are in rectangular grey boxes and edges are in line arrows. $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ represent the cell outputs of the previous two layers

3.2.3 Discrete vs. Continuous Search Strategy

Discrete search strategy demands high computational resources [56] or complex (not end-to-end) NAS design [17]. The computational inefficiency is hard to resolve using discrete search strategies

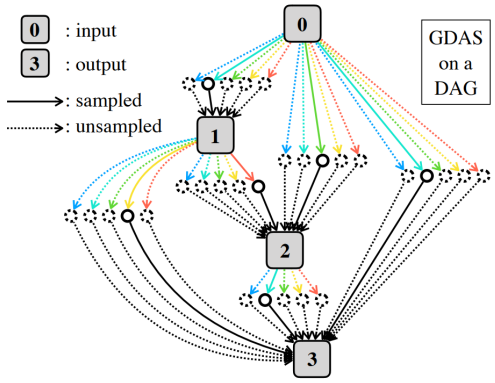


Fig. 11. Only one, i.e., the sampled subgraph in the cell DAG, as indicated by solid connections is trained in one iteration in GDAS [50] instead of the entire cell DAG that takes place in DARTS as illustrated in Fig. 9.

since they regard NAS as a black-box optimization problem that makes either little or ineffective use of information learned from previous queries during the search phase. Adaptive gradient-based continuous search strategy, on the other hand, exploits gradient information to reduce computational costs effectively [24, 48–50].

All discrete search strategies divide the cell search phase into two distinct stages: 1) searching the cell, and 2) evaluating the cell after 1 iteration of searching. (It is important to note that stage 2) here differs from architecture evaluation, which occurs after the completion of the entire cell search phase.) In other words, the two stages can hardly be addressed using the same method. It is because the second stage usually employs the effective gradient descent for cell weight optimization to evaluate cell performance [65, 82]. However, the first stage is solved by a non-differentiable discrete search strategy such as a RL controller [56], as illustrated in Fig. 12. The utilization of different methods for the two stages often leads to a complex NAS design [17, 29, 57], which requires researchers to possess expertise in the two fields involved.

On the other hand, a gradient-based search strategy can elegantly condense NAS to a bilevel optimization problem in which hyperparameters and weight parameters can be optimized jointly using the same method (gradient-based) without any extra functioning block [24, 48–50]. Therefore, a gradient-based search strategy is used in this project.

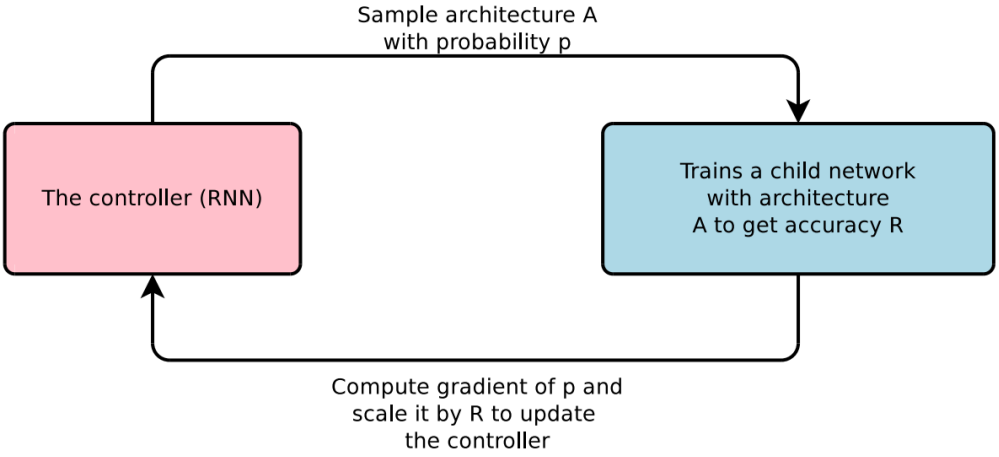


Fig. 12. Framing NAS as a RL problem requires an additional functioning block, i.e., a recurrent neural network (RNN) controller [17].

3.3 Performance Estimation Strategy

Search strategies are used to find a neural architecture A, from search space, that has maximal performance on some measures, such as accuracy on unseen data. These strategies need to compare and rank the performance of the candidate architectures during searching. The simplest approach to ranking performance is to fully train all the candidate architectures and evaluate their performance on the validation set. However, this approach demands enormous computational costs in thousands of GPU days due to the massive number of candidates [29, 73]. This naturally leads to an insurgence of performance estimation methods. There are four main strategies: **lower fidelity estimates**, **learning curve extrapolation**, **weight inheritance**, and **weight sharing**.

3.3.1 Lower Fidelity Estimates

Performance estimations based on **lower fidelities** of the actual performance (achieved by the simplest approach) include fewer training epochs [56, 83], training on data subset [84], on lower-resolution images [85], fewer filters per layer and fewer cells [56], and fewer backpropagation inner steps for weight updates [24]. These methods reduce the computational demand but also introduce bias to the estimate since it underestimates the actual performance of the best architectures. The bias is not harmful if the relative ranking of candidates remains stable. However, recent studies show that the relative ranking can change significantly when the cheap approximations deviate too much from the full evaluation [83], noting the need for a gradual increase in fidelities [86].

3.3.2 Learning Curve Extrapolation

Learning curve extrapolation is another possible method for performance estimation of architecture for early termination of unpromising candidates [57, 78]. The learning curve of architecture is predicted using external trained regression models [31], and trained predictor function [87]. An alternative approach is framing the performance predictor as an optimization problem, which improves continuously during prediction [88].

3.3.3 Weight Inheritance / Network Morphism

In simple terms, **network morphism** is a parameter- or weight-transferring map from a parent network to a child network while preserving the function and outputs of the child [89]. Eliminating the need to train the child networks from scratch when successively increasing their capacity results in methods requiring only a few GPU days [32, 63]. This approach allows search spaces with no intrinsic upper bound on the architecture's size [58].

3.3.4 Weight Sharing / One-Shot Models

One-Shot Architecture Search involves a supergraph (one-shot model/ **supernetwork**) made of different subgraphs (all possible architectures). Common edges are shared weights in these subgraphs. In other words, only a single, and often over-parameterized neural network (one-shot model/ supernetwork) is trained during the search process. The weight sets of all architectures are subsets of weight set of the one-shot model trained. Training one instead of many architectures reduces the

GPU days to just a few. ENAS [19] is the first in the field of NAS to propose weight sharing explicitly and demonstrates a significant reduction in computational cost. Weight sharing is used along with different search strategies such as RL in ENAS, and gradient-based method in DARTS [24], and SNAS [54], resulting in competitive performances.

Nevertheless, this method introduces bias like the lower fidelity estimates method for the same reason (see Section 3.3.1), which prompts work on bias reduction. DNA [90] modularizes the search space to reduce the representation shift induced by parameter sharing. GDAS-NSAS [91] proposes a Novelty Search based Architecture Selection (NSAS) loss function to mitigate the problem of multi-model forgetting (when parameter sharing is used to train a new child architecture sequentially, the performance of the previous child architecture is reduced).

3.3.5 Comparisons

Weight inheritance and weight sharing are the most effective performance estimation strategies for improving search efficiency [31, 32, 92, 93]. However, weight inheritance relies on a parent network and necessitates a complex system structure like Auto-Keras [94] for training a NAS network. On the other hand, weight sharing can be integrated into a simpler end-to-end DAS-based NAS method, such as DARTS [24]. Therefore, this project utilizes weight sharing.

3.4 Bilevel Optimization

NAS, as a subset of hyperparameter optimization, can be cast as a **bilevel optimization** (**BLO**) problem when gradient-based search strategy and performance estimation strategy are used [24]. BLO contains two levels of optimization tasks, where the lower-level task in eq. (3) is nested within the upper-level task in eq. (2) [95, 96]. In NAS, it is usually formulated as:

$$\boldsymbol{\lambda}^* = \arg\min_{\lambda} \mathcal{L}_{upper}(\boldsymbol{\lambda}, \boldsymbol{w}^*) \tag{2}$$

$$s.t.\ \boldsymbol{w}^* = \arg\min_{w} \mathcal{L}_{lower}(\boldsymbol{\lambda}, \boldsymbol{w}) \tag{3}$$

where $\boldsymbol{\lambda}$, as the upper-level variable, is the hyperparameters (parameters that define the architecture) and $\boldsymbol{w}$, as the lower-level variable, is the weights of the supernetwork. The definition of supernetwork refers to Section 3.3.4. $\boldsymbol{w}^*$ is equivalent to the best-response function $\boldsymbol{r}(\boldsymbol{\lambda})$:

$$\boldsymbol{w}^* = \boldsymbol{r}(\boldsymbol{\lambda}) = \arg\min_{w} \mathcal{L}_{in}(\boldsymbol{\lambda}, \boldsymbol{w}) \tag{4}$$

BLO minimizes the upper-level objective $\mathcal{L}_{upper}$ defined in terms of the optimal solution to an lower-level objective $\mathcal{L}_{lower}$. $\mathcal{L}_{upper}$ is usually validation cost function $\mathcal{L}_{val}$ while $\mathcal{L}_{lower}$ is usually training cost function $\mathcal{L}_{train}$:

$$\boldsymbol{\lambda}^* = \arg\min_{\lambda} \mathcal{L}_{val}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) \tag{5}$$

$$s.t.\ \boldsymbol{r}(\boldsymbol{\lambda}) = \arg\min_{w} \mathcal{L}_{train}(\boldsymbol{\lambda}, \boldsymbol{w}) \tag{6}$$

BLO is strongly non-deterministic polynomial-time hard (NP-hard) even if all objectives and constraints are linear [97]. Therefore, it is very hard to obtain exact solutions for BLO problems. Hence, BLO in NAS is usually solved by using a gradient descent-based algorithm [24, 54] which runs 2 steps until convergence: hyperparameter optimization (search strategy) and supernetwork weight optimization (performance estimation strategy).

Hyperparameter optimization involves computing $d\mathcal{L}_{val}/d\boldsymbol{\lambda}$, the total gradient of validation loss with respect to the hyperparameters. Supernetwork weight optimization involves computing $\partial\mathcal{L}_{train}/\partial\boldsymbol{w}$, the partial derivative of training loss with respect to the weights of the child architecture. The former is to perform gradient descent on eq. (5) while the latter is to perform gradient descent on eq. (6). $d\mathcal{L}_{val}/d\boldsymbol{\lambda}$ is used instead of $\partial\mathcal{L}_{val}/\partial\boldsymbol{\lambda}$ as $\mathcal{L}_{val}$ usually does not depend directly on $\boldsymbol{\lambda}$. In other words, $\partial\mathcal{L}_{val}/\partial\boldsymbol{\lambda}$ is usually 0. $d\mathcal{L}_{val}/d\boldsymbol{\lambda}$ is often called hypergradient and can be computed by chain rule:

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) = \underbrace{\frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda}))}_{\text{direct gradient}} + \underbrace{\overbrace{\left(\frac{\partial\boldsymbol{r}}{\partial\boldsymbol{\lambda}}(\boldsymbol{\lambda})\right)^{\top}}^{\text{response Jacobian}} \frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{w}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda}))}_{\text{response gradient}} \tag{7}$$

However, the exact computation of $\partial\boldsymbol{r}/\partial\boldsymbol{\lambda}$ that leverages Implicit Function Theorem (IFT) is computationally expensive as it involves inverting a high-dimensional Hessian $\boldsymbol{H}^{-1}$:

$$\frac{\partial\boldsymbol{r}}{\partial\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = -\underbrace{\left(\frac{\partial^2\mathcal{L}_{train}}{\partial\boldsymbol{w}^2}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda}))\right)^{-1}}_{=\boldsymbol{H}^{-1}} \frac{\partial^2\mathcal{L}_{train}}{\partial\boldsymbol{\lambda}\partial\boldsymbol{w}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) \tag{8}$$

As a result, the hypergradient is usually estimated instead of being calculated in its exact form in the gradient descent-based algorithm. There are three mainstream approximation methods used in hyperparameter optimization problems: **unrolling**, **implicit differentiation**, and lately-developed **hypernetworks**.

3.4.1 Unrolling and Implicit Differentiation

Unrolling, also known as iterative differentiation (ITD), is backpropagation through the computation graph for gradient descent of outer objective with respect to weights as shown in Fig. 13 and the response gradient in eq. (9).

By using chain rule, eq. (7) to find hypergradient can be expressed as

$$\begin{aligned}\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) = \frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) + \frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{w}_{T+1}}\frac{\partial\boldsymbol{w}_{T+1}}{\partial\boldsymbol{\lambda}} + \frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{w}_{T+1}}\frac{\partial\boldsymbol{w}_{T+1}}{\partial\boldsymbol{w}_{T}}\frac{\partial\boldsymbol{w}_{T}}{\partial\boldsymbol{\lambda}} \\ + \frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{w}_{T+1}}\frac{\partial\boldsymbol{w}_{T+1}}{\partial\boldsymbol{w}_{T}}\frac{\partial\boldsymbol{w}_{T}}{\partial\boldsymbol{w}_{T-1}}\frac{\partial\boldsymbol{w}_{T-1}}{\partial\boldsymbol{\lambda}} + ... + \frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{w}_{T+1}}\frac{\partial\boldsymbol{w}_{T+1}}{\partial\boldsymbol{w}_{T}}\frac{\partial\boldsymbol{w}_{T}}{\partial\boldsymbol{w}_{T-1}}...\frac{\partial\boldsymbol{w}_{1}}{\partial\boldsymbol{\lambda}}\end{aligned} \tag{9}$$

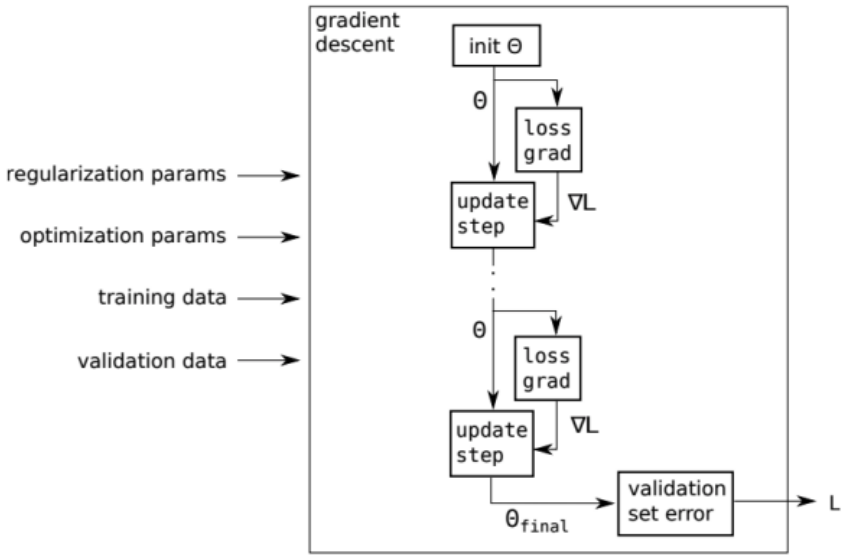


Fig. 13. Computation graph for gradient descent [98]. θs represent weights $\boldsymbol{w}$.

where T is the number of inner steps set for weight optimization (or unrolling) during hyperparameter optimization. The largest T possible is determined by the the number of inner steps needed for the weight optimization to converge.

The hypergradient can be estimated by propagating derivatives backwards through the entire inner optimization [99, 100], through truncated backpropagation which unrolls T ($\geq K$) inner steps but only backprops through the last K steps [101], and by unrolling and backpropagating only one inner step ($T = K = 1$) along with finite difference approximation in DARTS [24]. For example, if K is set to equal 1, eq. 9 will become

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) = \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) + \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}_{T+1}} \frac{\partial \boldsymbol{w}_{T+1}}{\partial \boldsymbol{w}_T} \frac{\partial \boldsymbol{w}_T}{\partial \boldsymbol{w}_{T-1}} ... \frac{\partial \boldsymbol{w}_1}{\partial \boldsymbol{\lambda}} \tag{10}$$

Since the direct gradient is usually 0, in this case, the hypergradient is effectively

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) = \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}_{T+1}} \frac{\partial \boldsymbol{w}_{T+1}}{\partial \boldsymbol{w}_T} \frac{\partial \boldsymbol{w}_T}{\partial \boldsymbol{w}_{T-1}} ... \frac{\partial \boldsymbol{w}_1}{\partial \boldsymbol{\lambda}} \tag{11}$$

It is worth noting that inner weight optimization here is different from the lower-level weight optimization discussed in Section 3.4 even though they decay the same set of weights (supernet weights). w_{T+1} refers to the supernet weight resultant from lower-level weight optimization from the previous iteration. $w_T, w_{T-1}, w_{T-2}, ..., w_2, w_1$ were then computed in the current iteration for hyperparameter optimization. After updating hyperparameter in the current iteration, $w_T, w_{T-1}, w_{T-2}, ..., w_2, w_1$ would be destroyed and the lower-level weight optimization thereafter would be based on w_{T+1}. This is to avoid the supernet weights to converge based on immature architectures (architectures found in early iterations).

Implicit differentiation, also known as **approximate implicit differentiation** (**AID**), first computes the equation for the response Jacobian $\partial \boldsymbol{r}/\partial \boldsymbol{\lambda}$ by using IFT as formulated in eq. (8). The equation is then approximately solved through a two-stage method. The common approximation methods are the fixed-point method [102] and the conjugate gradient method [103].

Both unrolling and AID require inner optimization of supernetwork weights for every outer update of hyperparameters. This results in the same trade-off between computing demand and optimality of weights as the performance estimation of child architectures discussed in Section 3.3.1.

3.4.2 Hypernetworks

Hyperparameter optimization based on hypernetworks allows for the amortization of hypergradient computation costs without facing the trade-off dilemma inherent in implicit differentiation and unrolling discussed in Section 3.4.1. The advantage lies in the fact that hypernetworks do not require supernet weight inner optimization discussed in Section 3.4.1. The details of how hypernetworks work to solve hyperparameter optimization problems are discussed in Section 4.3.

However, the hypernetwork poses a challenge due to its exceptionally high-dimensional mapping, resulting in enormous memory requirements and a lengthy amortization process, as it can have several times more parameters than the elementary model. For instance, in the case of a linear hypernetwork $\boldsymbol{w}_{\boldsymbol{\phi}}$ with H input nodes (representing an input of size H) that outputs D elementary weights, $D \times H$ hypernetwork parameters are needed for $\boldsymbol{\phi}$ ($\boldsymbol{w}_{\boldsymbol{\phi}}$ has a size of $D \times H$).

To address this issue, [46] reduced the number of hypernetwork parameters by implementing a hypernetwork that outputs to any layer within the elementary network feeding embeddings. Each embedding represents a layer within the elementary network. Building upon this progress, [104] introduced an enhanced solution called Graph HyperNetwork (GHN), which combines a hypernetwork with a Graph Neural Network. GHN can better approximate the best response than linear regression as the elementary weights have a graph-like structure. On the other hand, [105] proposed Self-Tuning Network (STN), a compact hypernetwork that approximates the best response for the entire elementary network as an affine transformation of the hyperparameters.

3.4.3 Hypernetworks for Neural Architecture Search

As explained in Section 3.4.2, hypernetworks are among the prominent methods utilized for computing hypergradients in hyperparameter optimization. Within this context, hypernetworks have also been employed for addressing neural architecture search problems, which falls under the umbrella of hyperparameter optimization problems. This subsection highlights the distinctions between the research conducted in this project and the relevant work conducted by others.

Both GHN [104] and SMASH [106] use hypernetworks to model the supernetwork weights globally along with random search to sample candidate architectures. They model the supernetwork weights globally instead of locally, as proposed by this project, resulting in a high search cost. In addition, they integrate the hypernetworks into a discrete search strategy, i.e., random sampling. In other words, they employ two types of strategies for architecture search and performance estimation: gradient-based and random-based. On the other hand, this project proposes a simpler approach by integrating hypernetworks into a DAS model, enabling the same method (gradient-based) to be used for both search strategy and performance estimation strategy. No work has been done to integrate hypernetworks into a DAS model.

4 Methodology and Methods

DARTS (Differentiable Architecture Search) [24] is a pioneering gradient-based approach to NAS, using a continuous relaxation of the architecture space allowing for efficient optimization through backpropagation. DARTS outperforms most discrete search strategies, effectively discovering competitive neural architectures while reducing computational costs. DARTS tackles the search problem with bilevel optimization, jointly optimizing architecture parameters $\boldsymbol{\lambda}$ (hyperparameters) and candidate network weights $\boldsymbol{w}$ using gradient descent. In its search strategy, DARTS approximates the computationally expensive hyperparameter gradient (hypergradient) through a one-step unrolling scheme.

DARTS is used as the foundational model in this project due to its popularity [26, 48, 53], simplicity, efficiency, and pioneering status in the field of differentiable neural architecture search (DAS). In fact, the majority of DAS studies are more or less based on DARTS, making it a widely recognized and influential approach [26]. This project proposes to develop a new DARTS-based model, namely hyperDARTS to improve its performance in finding optimal architectures.

4.1 Overview

Workflow of hyperDARTS is similar to DARTS discussed in Section 3.1.2:

1. Define a modular search space.
2. Search the optimal normal and reduction cells using a smaller macro-architecture.
3. Extract the optimal normal and reduction cells to stack as a larger macro-architecture (briefly explained in Fig. 8).
4. Evaluate the larger macro-architecture on image classification datasets to showcase its task performance.

Therefore, hyperDARTS design comprises three key steps: defining the search space (step 1), architecture search (step 2 and 3), and architecture evaluation (step 4). Architecture search involves the use of search strategy and performance estimation strategy. Both the search space, the performance estimation strategy, and the architecture evaluation phase are the same for hyperDARTS and DARTS. The main difference lies in the search strategy.

Before diving into technical details, the overviews of DARTS and proposed hyperDARTS are shown in Fig. 14 and Fig. 15 respectively. Basically, in DARTS, the hypergradient $\partial\mathcal{L}_{val}/\partial\boldsymbol{\lambda}$ is computational expensive and DARTS uses an one-step unrolling scheme to approximate it. This incurs approximation error, reducing DARTS to only find suboptimal architectures [24]. Therefore, this project proposes to use **hypernetworks** ϕ to model $\boldsymbol{w}$ so that $\partial\mathcal{L}_{val}/\partial\boldsymbol{\lambda}$ can be computed using normal backpropagation, removing the approximation scheme and approximation error in DARTS. Given ϕ can model good enough $\boldsymbol{w}$, hyperDARTS should be able to find better architectures than DARTS. The technical details are explained in the remaining subsections.

The following subsections center around the design and the implementation of hyperDARTS. The **datasets** used (Section 4.2) is first discussed. Then **search space** used in hyperDARTS is explained (Section 4.3). The following subsection is all about **search strategy** (Section 4.4). Section 4.4 is divided into 3 parts. The preliminary **search strategy used in DARTS** (Section 4.4.1) is first

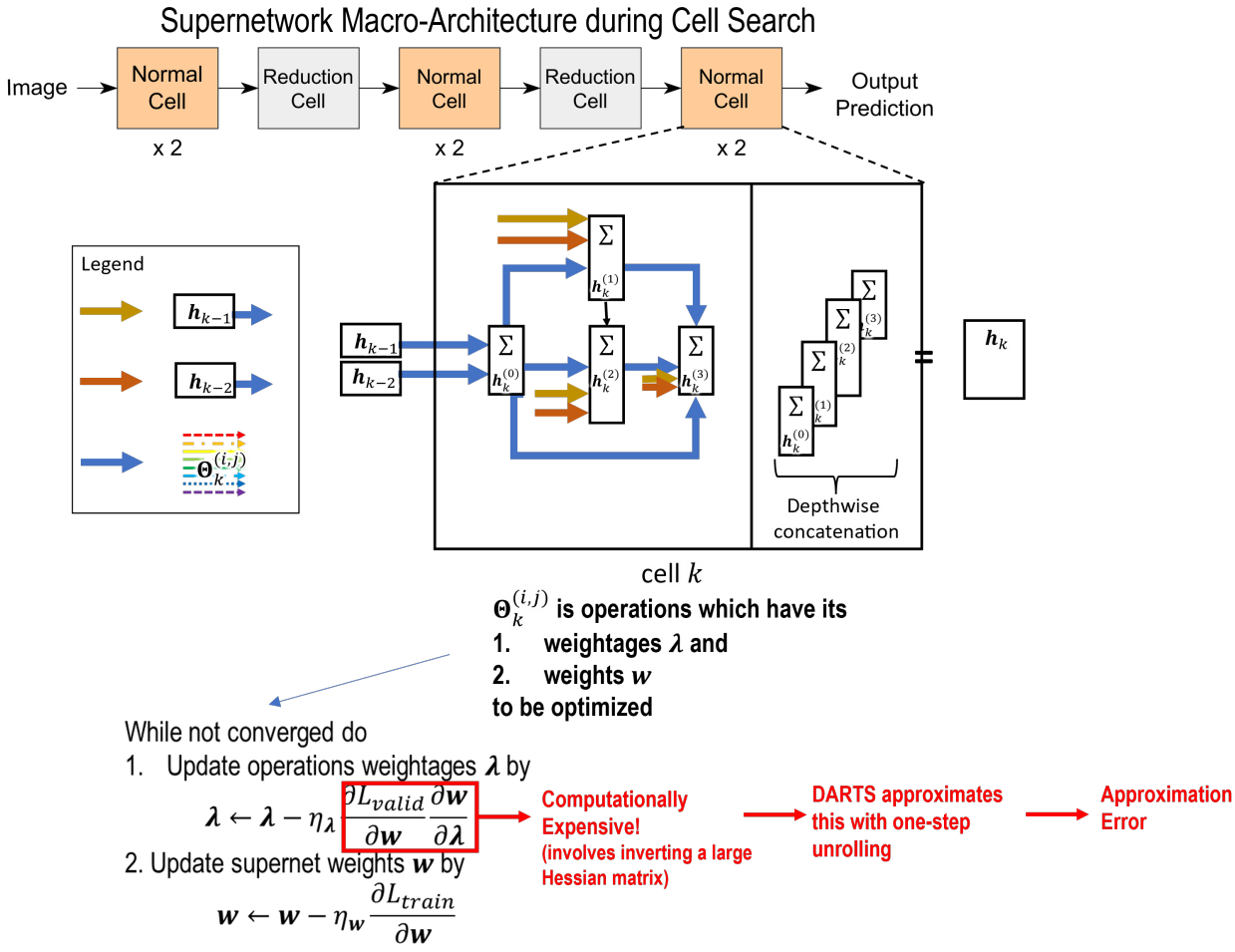


Fig. 14. Overview of DARTS.

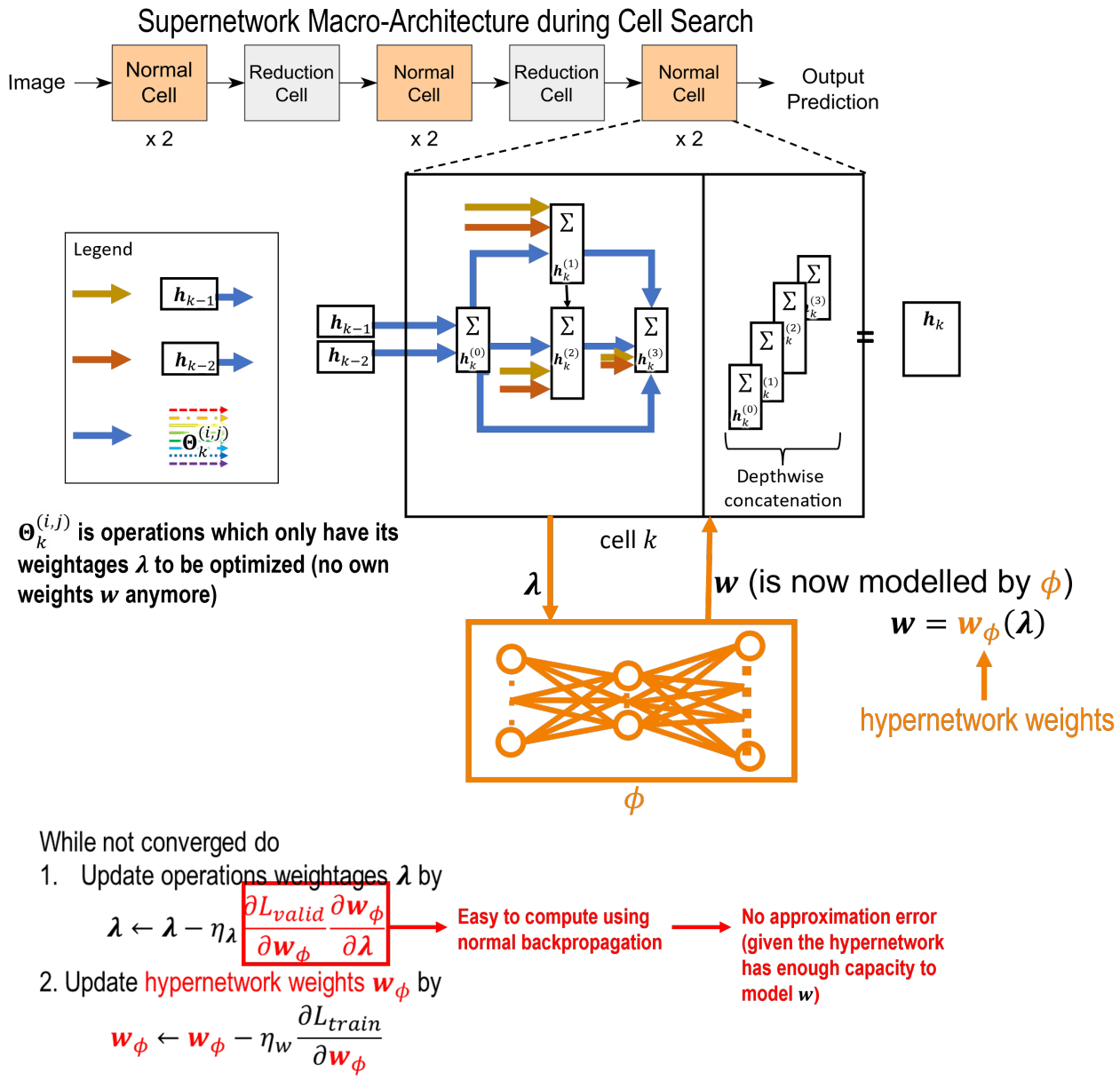


Fig. 15. Overview of the proposed hyperDARTS. λ outputs $\boldsymbol{w}$ for the operations of all the cells. Only 1 cell is illustrated here.

discussed. Search strategy used in DARTS is the part where this project tries to replace. Therefore, the theoretical concept of **using hypernetworks as search strategy** are then presented (Section 4.4.2), followed by the practical **hyperDARTS algorithm developed for cell searching** (Section 4.4.3). After defining the search strategy used in hyperDARTS, the **hyperDARTS macro-architecture and performance estimation strategy** are explained (Section 4.5). Finally, a more detailed overview of the proposed **hyperDARTS models** (Section 4.6), methods to **selecting the best architecture** from all found architectures for **architecture evaluation** (Section 4.7) and **experimental setup** (Section 4.8) are presented.

4.2 Datasets

In this project, two image classification datasets are used: CIFAR-10 (Canadian Institute For Advanced Research, 10 classes) [107] and CIFAR-100 (Canadian Institute For Advanced Research, 100 classes) [107]. CIFAR-10 consists of 60000 32 $\times$32 colour images equally distributed across 10 different classes of vehicles and animals. CIFAR-100 consists of 60000 32 $\times$32 colour images equally distributed across 100 different classes of animals, plants, scenes, humans, food, and other non-living objects. Despite having some common specifications, the two datasets have mutually exclusive classes and do not contain shared images. This property allows any one of them to be used as negative examples for the other.

CIFAR-10 is used for both cell search and architecture evaluation, while the more challenging CIFAR-100 is used only for architecture evaluation.

4.3 Search Space

Before discussing the search space, an overview of neural network is given to provide background information.

4.3.1 Background

A neural network is illustrated in Fig. 16. Generally, a neural network has 3 types of layers: input layer, hidden layer, and output layer. The input layer $\boldsymbol{x}$ is the input data while the output layer $\hat{y}$ is the output prediction. A hidden layer h is a feature vector or a feature map. $\boldsymbol{h}_k$ is the results of operations acted on $\boldsymbol{h}_{k-1}$. $\boldsymbol{h}_0$ equals $\boldsymbol{x}$. The arrows represent the operations. Only some types of operations contain trainable parameters $\boldsymbol{w}$, e.g., linear transformation in multi-level perceptron (MLP) and convolution operations in convolutional neural network (CNN). MLP and CNN are illustrated in Fig. 17.

MLP linear transforms $\boldsymbol{h}_{k-1}$ and then applies non-linear function to it, resulting in $\boldsymbol{h}_k$. $\boldsymbol{w}$ is the linear transformation matrix. It is worth noting that it is common to call a neural network N-layer neural network with N being the total number of hidden layers and output layer. For example, if a MLP has 1 hidden layer and 1 output layer, it is called 2-layer MLP. CNN applies convolutional operations, pooling operations, and non-linear function to $\boldsymbol{h}_{k-1}$, resulting in $\boldsymbol{h}_k$. Only convolutional operations contain trainable parameters $\boldsymbol{w}$ for their kernels.

As discussed in Section 3, this project focuses on finding optimal CNN architectures. CNN architecture variability includes the number of hidden layers, which type of operations for which hidden layer, etc. In NAS model, these architectural properties to be optimized are parameterized and are called

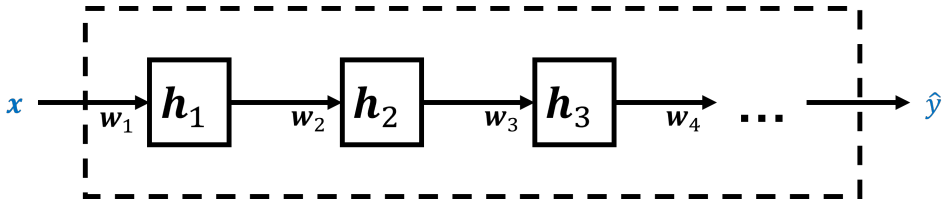
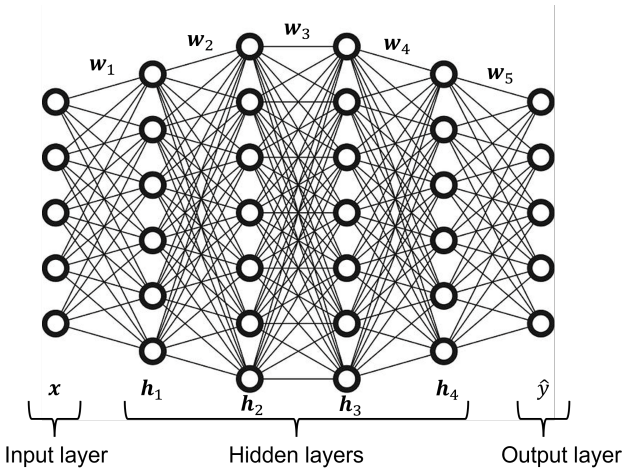
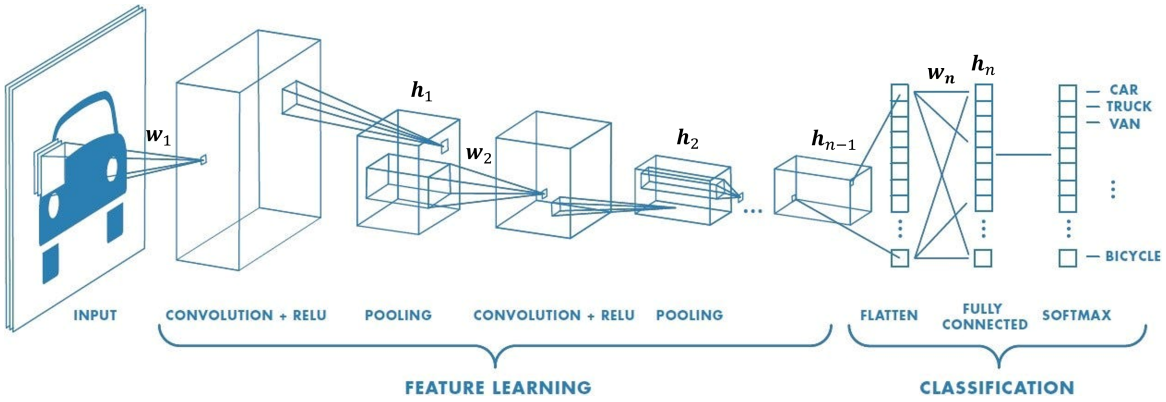


Fig. 16. A neural network. Generally, a neural network has 3 types of layers: input layer, hidden layer, and output layer. The input layer $\boldsymbol{x}$ is the input data while the output layer $\hat{y}$ is the output prediction. A hidden layer h is a feature vector or a feature map. $\boldsymbol{h}_k$ is the results of operations acted on $\boldsymbol{h}_{k-1}$. $\boldsymbol{h}_0$ equals $\boldsymbol{x}$. The arrows represent the operations. Only some types of operations contain trainable parameters $\boldsymbol{w}$, e.g., linear transformation in multi-level perceptron (MLP) and convolution operations in convolutional neural network (CNN). MLP and CNN are illustrated in Fig. 17.



(a) Multilevel Perceptron (MLP).



(b) Convolutional Neural Network (CNN).

Fig. 17. MLP [108] linear transforms $\boldsymbol{h}_{k-1}$ and then applies non-linear function to it, resulting in $\boldsymbol{h}_k$. $\boldsymbol{w}$ is the linear transformation matrix. CNN [109] applies convolutional operations, pooling operations, and non-linear function to $\boldsymbol{h}_{k-1}$, resulting in $\boldsymbol{h}_k$. Only convolutional operations contain trainable parameters $\boldsymbol{w}$ for their kernels.

hyperparameters $\boldsymbol{\lambda}$. This project focuses on finding the best operation types and their placement to build an optimal architecture, i.e., optimizing the operation types and operation placements.

As discussed in Section 3, the search space of architectures can be defined entirely by the operation space and other related constraints. In other words, optimal hyperparameter $\boldsymbol{\lambda}^*$ defines the architecture found, therefore, $\boldsymbol{\lambda}$ also refers to architecture parameters. $\boldsymbol{\lambda}$ is continuous in this project through a relaxation of search space, which is discussed in next subsection (Section 4.3.2).

4.3.2 Search Space

As discussed in Sections 3.1.2, by using a modular search space, architecture search can be reduced to cell search. As illustrated in Fig. 15, two types of cells have to be found from the modular search space: a normal cell and a reduction cell. The difference between a normal cell and a reduction cell is shown in Fig. 18.

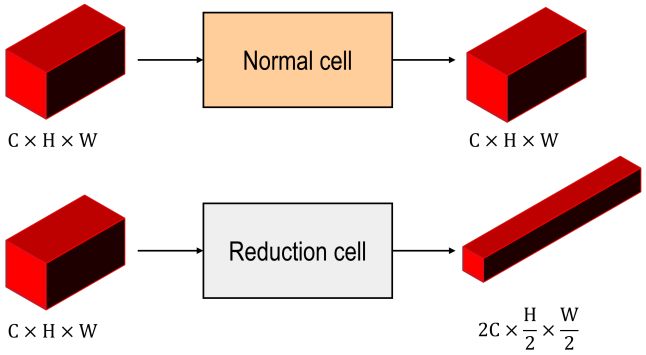


Fig. 18. A normal cell preserves spatiality while a reduction cell reduces spatiality. Both are common in modern CNNs [20, 25, 64]. The red 3D box represents a feature map. A feature map is the output of operations acted on input. C represents the number of channels, H represents the height while W represents the width.

A cell is a DAG consisting of

1. an ordered sequence of 7 nodes (including 2 input nodes, 1 output node, and 4 intermediate nodes),
2. with 1 directed edge between every intermediate node and all its predecessor nodes (including input nodes) as illustrated in Fig. 20.

Each node $\boldsymbol{h}^{(j)}$ is a latent representation (a feature map in CNN). Each directed edge $\Theta^{(i,j)}$ (directed from $\boldsymbol{h}^{(i)}$ to $\boldsymbol{h}^{(j)}$) is a weighted sum of operations $o \in O$ that transforms $\boldsymbol{h}^{(i)}$.

Fig. 19 shows the changes of a hyperDARTS cell during cell search. The predefined operation set has a dimension of 8. In other words, there are 8 operations possible. The details of the operation set are in Section 4.8. 1 color in line arrows represents 1 type of operation. Each directed edge, as a weighted sum of the 8 distinct operations, is shown by 8 lines of distinct colors. Fig. 19 shows a simplified cell for explanation, each cell in DARTS in fact has two input nodes which also direct edges to intermediate nodes as illustrated in Fig. 20.

Fig. 20 shows the micro-architecture in hyperDARTS cell during cell search phase. The 2 input nodes are the outputs of previous two cells (cells are stacked into a **small**, predefined architecture as shown in Fig. 23 (middle) for architecture search). The output node is the depthwise (the channel dimension) concatenation of all intermediate nodes. Each intermediate node is a function of all its predecessors:

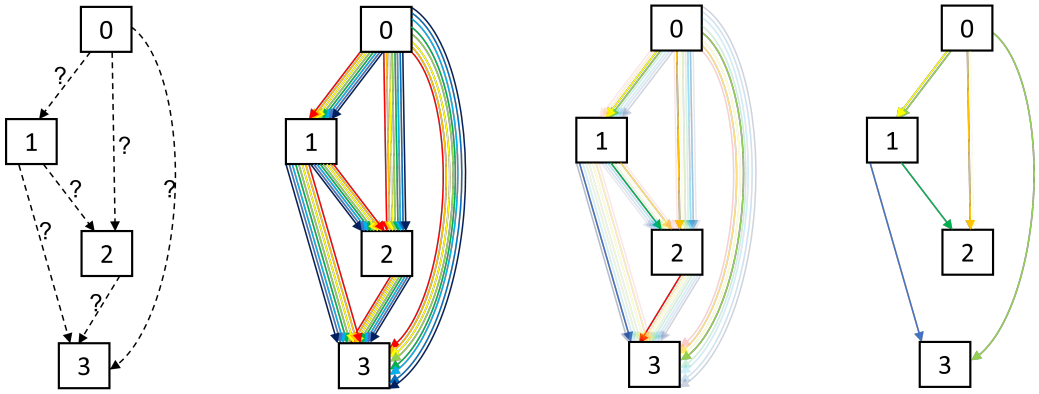


Fig. 19. Cell search strategy of hyperDARTS [24]. (a) Operations on the edges are unknown at first. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. An edge is a set of (all the) arrows pointing from one node to the other rather than one of the arrows. In other words, there are 6 edges here. (c) Joint optimization of $\boldsymbol{\lambda}$ and the hypernetwork weights $\boldsymbol{w}_\phi$ as a bilevel optimization problem. During optimization, $\boldsymbol{\lambda}$ is constantly updated so each edge has high and low λs represented by the high and low intensity arrows. (d) Discretization of final cell architecture based on learnt $\boldsymbol{\lambda}$. Only the 2 largest λs are kept for each node. Note: Nodes are in rectangular grey boxes. Each colour in line arrows represents a specific operation type from the predefined operation set, such as 3×3 separable convolution.

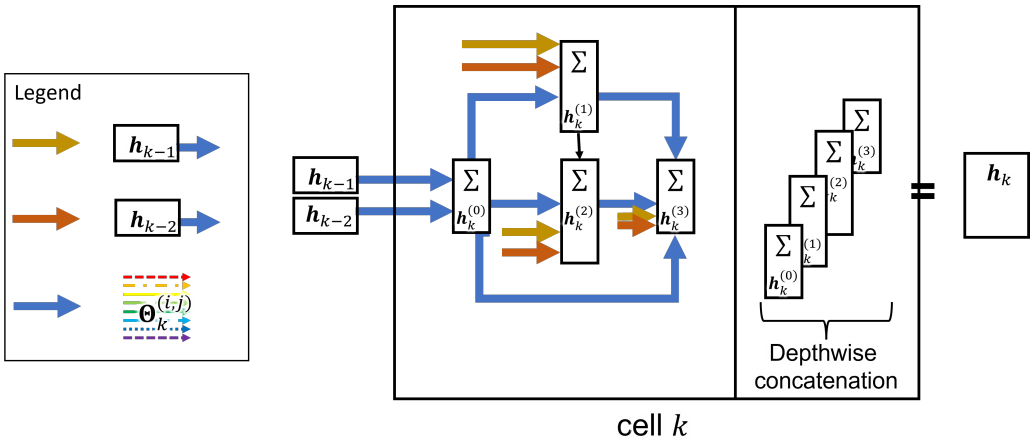


Fig. 20. A cell in DARTS has two inputs $\boldsymbol{h}_{k-1}$ and $\boldsymbol{h}_{k-2}$ which are the output from the previous two cells and one output node $\boldsymbol{h}_k$. There are 4 intermediate nodes in between. The predefined operation set used in DARTS has a dimension of 8, representing by the 8 rainbow-coloured arrows that form the directed edge $\Theta^{(i,j)}$ where the directed edge points from node i to node j.

$$\boldsymbol{h}_k^{(j)} = \sum_{i<j} \Theta_k^{(i,j)} \left(\boldsymbol{h}_k^{(i)} \right) \tag{12}$$

Let O be the set of predefined candidate operations (e.g., max pooling, zero, 5×5 separable convolution). Every directed edge $\Theta^{(i,j)}$ is a weighted sum (scaled by softmax of corresponding operation weights $\lambda_o^{(i,j)}$) of all candidate operations $o \in O$. To relax the search space from being discrete (since each operation is inherently discrete) to being continuous, the categorical choice of a particular operation is relaxed to a softmax over all candidate operations in a directed edge $\Theta^{(i,j)}$:

$$\Theta^{(i,j)}(\boldsymbol{x}) = \sum_{o \in O} \frac{exp(\boldsymbol{\lambda}_o^{(i,j)})}{\sum_{o' \in O} exp(\boldsymbol{\lambda}_{o'}^{(i,j)})} o(\boldsymbol{x}) \tag{13}$$

where $\boldsymbol{\lambda}^{(i,j)}$ is a $|O|$-dimensional vector representing operation weightages (architecture parameters/ hyperparameters) for a pair of nodes (i, j). The cell search task thus reduces to learning a set of continuous variables (can be seen as a matrix) $\boldsymbol{\lambda} = \{\boldsymbol{\lambda}^{(i,j)}\}$. At the end of search, a discrete architecture of a cell can be determined by replacing each directed edge $\Theta^{(i,j)}$ with the most likely operation, i.e., the discretized directed edge $o^{(i,j)}$:

$$o^{(i,j)} = \arg\max_{o \in O} \boldsymbol{\lambda}^{(i,j)} \tag{14}$$

$\boldsymbol{\lambda}$ is the hyperparameters and the upper-level variable in the BLO problem (eq. (5) and eq. (6)). The BLO question is now defined. Section 4.4 discusses the search strategy used by hyperDARTS to solve the BLO problem.

4.4 Search Strategy

HyperDARTS uses hypernetworks as search strategy. Before discussing that, the unrolling search strategy used in DARTS is discussed first so to demonstrate the difference in search strategy between DARTS and hyperDARTS.

4.4.1 Preliminary - Unrolling in DARTS Search Strategy

This subsection centers around the one-step unrolling hypergradient approximation scheme employed in DARTS. Additionally, a concise overview of the overall search strategy utilized in DARTS is presented. The algorithm for DARTS [24] is outlined as in Algorithm 1.

Algorithm 1 DARTS - Differentiable Architecture Search

Create a mixed operation $\Theta^{(i,j)}$ parameterized by $\lambda^{(i,j)}$ for edge edge (i, j)
while not converged **do**
  $\boldsymbol{x} \sim$ Training data, $\boldsymbol{x'} \sim$ Validation data
  Update architecture $\boldsymbol{\lambda}$ by descending $\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{w} - \xi \frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{\lambda}), \boldsymbol{\lambda}, \boldsymbol{x'})$
  Update supernet weights $\boldsymbol{w}$ by descending $\frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{\lambda}, \boldsymbol{x})$
Derive the final architecture based on the learned $\boldsymbol{\lambda}$

The $\boldsymbol{\lambda}$ update happens every upper-level step while the $\boldsymbol{w}$ update happens every lower-level step. Each iteration has 1 upper- and 1 lower-level step. The upper-level step optimizes hyperparameters $\boldsymbol{\lambda}$ while the lower-level step optimizes supernetwork weights $\boldsymbol{w}$. The supernetwork is a neural network which contains all the candidate architectures via weight sharing as discussed in Section 3.3.4.

DARTS approximates hypergradient as follows:

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{w}^*(\boldsymbol{\lambda}), \boldsymbol{\lambda}) \approx \frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{w} - \xi \frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{\lambda}), \boldsymbol{\lambda}) \tag{15}$$

where $\boldsymbol{w}$ represents the supernetwork weights in a specific iteration, which are the resultant weights from the weight update in the previous iteration. ξ, the inner step learning rate, is chosen to be the same as the learning rate of the lower-level step as a simple working strategy. Only one inner

step is performed to descend $\boldsymbol{w}$ during $\boldsymbol{\lambda}$ optimization. Applying chain rule to the hypergradient approximation (right-hand side of eq. (15)) results in:

$$\frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{\lambda}}(\boldsymbol{w}', \boldsymbol{\lambda}) - \xi \frac{\partial \mathcal{L}^2_{train}}{\partial \boldsymbol{\lambda} \partial \boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{\lambda}) \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}'}(\boldsymbol{w}', \boldsymbol{\lambda}) \tag{16}$$

where $\boldsymbol{w}' = \boldsymbol{w} - \xi \frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{\lambda})$. The second term in eq. (16) is approximated using finite difference method to avoid the expensive computation of the matrix-vector product lying inside the term:

$$\frac{\partial \mathcal{L}^2_{train}}{\partial \boldsymbol{\lambda} \partial \boldsymbol{w}}(\boldsymbol{w}, \boldsymbol{\lambda}) \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}'}(\boldsymbol{w}', \boldsymbol{\lambda}) \approx \frac{\frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{\lambda}}(\boldsymbol{w}^+, \boldsymbol{\lambda}) - \frac{\partial \mathcal{L}_{train}}{\partial \boldsymbol{\lambda}}(\boldsymbol{w}^-, \boldsymbol{\lambda})}{2\epsilon} \tag{17}$$

where $\boldsymbol{w}^{\pm} = \boldsymbol{w} \pm \epsilon \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}'}(\boldsymbol{w}', \boldsymbol{\lambda})$ and ϵ is a small scalar.

As discussed in Section 3.4.1, inner weight optimization is different from lower-level weight optimization. Inner weight optimization results in $\boldsymbol{w}'$ which is only used in $\boldsymbol{\lambda}$ optimization as shown in Algorithm 1. In each iteration, $\boldsymbol{w}'$ is discarded after updating $\boldsymbol{\lambda}$ and is not used for updating $\boldsymbol{w}$.

The one-step unrolling hypergradient approximation scheme used in DARTS is equivalent to the unrolling with $T = K = 1$ discussed in Section 3.4.1. This method bears no theoretical guarantee for convergence. Nonetheless, DARTS viability has been proved experimentally [24].

4.4.2 HyperDARTS uses Hypernetworks as Search Strategy

In this approach, a network (hypernetwork) generates the weights for another network (supernetwork) [45, 105], as illustrated in Fig. 21. In simple terms, the hypernetwork learns the best response function ($\boldsymbol{r}(\boldsymbol{\lambda}) = \arg\min_w \mathcal{L}_{train}(\boldsymbol{\lambda}, \boldsymbol{w})$) in eq. (6) by inputting hyperparameters $\boldsymbol{\lambda}$. The hypernetwork outputs supernetwork weights $\boldsymbol{w}^*$ at convergence.

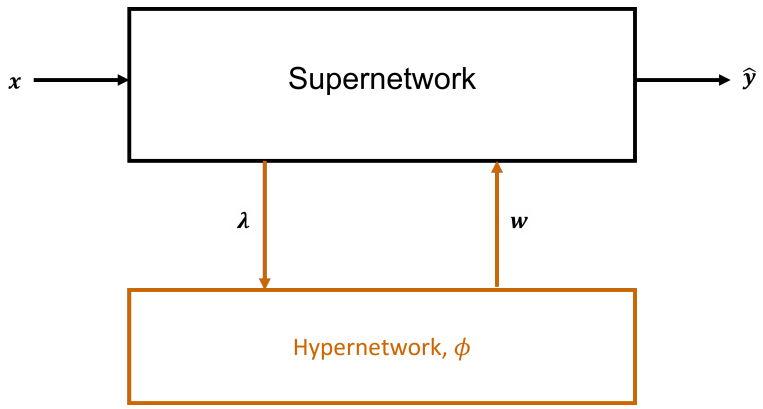


Fig. 21. The hypernetwork is fed $\boldsymbol{\lambda}$ and outputs the $\boldsymbol{w}$ after every lower-level step of $\boldsymbol{w}_{\phi}$ updates.

The hypernet $\boldsymbol{\phi}$ models $\boldsymbol{w}$ with hypernetwork weights $\boldsymbol{w}_{\phi}$:

$$\boldsymbol{w} = \boldsymbol{w}_{\phi}(\boldsymbol{\lambda}) \tag{18}$$

At convergence,

$$\boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda}) = \boldsymbol{w}^* = \boldsymbol{r}(\boldsymbol{\lambda}) \tag{19}$$

[45] has proved theoretically that:

1. if ϕ is a universal approximator, the hypernet can learn continuous best response $\boldsymbol{w}^*$, and
2. if the hypernetwork learns $\boldsymbol{w}^*$, it will simultaneously minimize the loss for every point in its hyperparameter distribution $p(\boldsymbol{\lambda})$ with sufficient support.

Therefore, the hypernetwork can learn the best response in the support of the hyperparameter distribution under mild assumptions. If the best response is differentiable, then there is a neighborhood about each hyperparameter where the best response is approximately linear. If the support of the hyperparameter distribution is in this neighborhood, then the local best response can be learnt by using linear regression as the hypernetwork.

In practice, there is no guarantee about the hypernetwork being a univesal approximator as it has limited parameters. As a result, the optimal hypernetwork will also depend on $p(\boldsymbol{\lambda})$ rather than just the support of its distribution. Nevertheless, [45] has proved experimentally that a local best-response can be learnt using linear regression as demonstrated in Fig. 22.

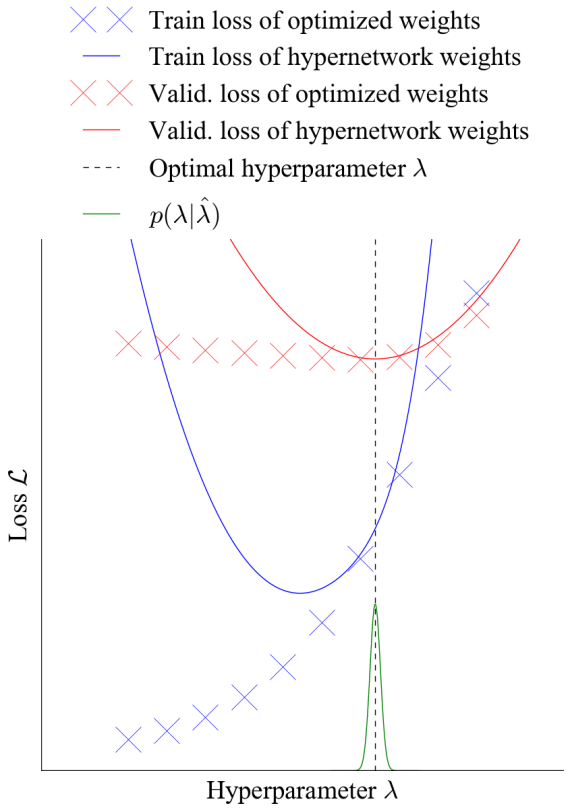


Fig. 22. Learning a local best-response using a hypernetwork [45]. Training and validation losses of a neural network, estimated by cross-validation (crosses) or a linear hypernetwork (lines). The hypernetwork's limited capacity makes it only accurate where the hyperparameter distribution puts mass i.e., the conditional hyperparameter distribution $p(\boldsymbol{\lambda}|\hat{\boldsymbol{\lambda}})$ which only puts mass close to $\hat{\boldsymbol{\lambda}}$.

In other words, for $\boldsymbol{\lambda} \in p(\boldsymbol{\lambda}|\hat{\boldsymbol{\lambda}})$,

$$\mathcal{L}_{val}(\boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda})) = \mathcal{L}_{val}(\boldsymbol{w}^*(\boldsymbol{\lambda})) \tag{20}$$

Given the hypernet is accurate in the vicinity of $\boldsymbol{\lambda}$, joint optimization of hyperparameters and parameters (elementary weights) can be used for hyperparameter optimization without the need of computing the inverse Hessian. It is because $\boldsymbol{\lambda}$ can be optimized by descending $d\mathcal{L}_{val}(\boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}), \boldsymbol{\lambda})/d\boldsymbol{\lambda}$ instead of $d\mathcal{L}_{val}(\boldsymbol{w}, \boldsymbol{\lambda})/d\boldsymbol{\lambda}$. In other words, the hypergradient is now:

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) = \frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda})) = \underbrace{\frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda}))}_{\text{direct gradient}} + \underbrace{\overbrace{\left(\frac{\partial \boldsymbol{w}_{\boldsymbol{\phi}^*}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda})\right)^{\top}}^{\text{response Jacobian}} \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}_{\boldsymbol{\phi}}}(\boldsymbol{\lambda}, \boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda}))}_{\text{response gradient}} \tag{21}$$

The response gradient in eq. 21 can now be computed easily using ordinary backpropagation through the hypernet as it only involves the computation of the derivatives of the validation loss $\mathcal{L}_{val}$ with respect to the output of the hypernet $\boldsymbol{w}_{\boldsymbol{\phi}}$ and the output of the hypernet $\boldsymbol{w}_{\boldsymbol{\phi}}$ with respect to the input of the hypernet $\boldsymbol{\lambda}$. According to the BLO formulated in Section 4.3, the direct gradient in eq. 21 is 0 as the validation loss does not directly depend on the hyperparameters. Therefore, by using a hypernetwork to replace the unrolling method in DARTS, hyperDARTS can update the hyperparameter $\boldsymbol{\lambda}$ by descending:

$$\frac{\partial \mathcal{L}_{val}(\boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}))}{\partial \boldsymbol{w}_{\boldsymbol{\phi}}} \left(\frac{\partial \boldsymbol{w}_{\boldsymbol{\phi}}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda})\right) \tag{22}$$

4.4.3 HyperDARTS Algorithm for Cell Searching

Search strategy used in hyperDARTS combines Section 4.3 and 4.4.2, resulting in the following algorithm:

Algorithm 2 Local HyperDARTS - Simplified Joint Optimization of Hyperparameters and Hypernetwork

Create a mixed operation $\Theta^{(i,j)}$ parameterized by $\lambda^{(i,j)}$ for edge edge (i, j)
while not converged **do**
  $\boldsymbol{x} \sim$ Training data, $\boldsymbol{x'} \sim$ Validation data
  Update architecture $\boldsymbol{\lambda}$ by descending $\frac{\partial \mathcal{L}_{val}(\boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}), \boldsymbol{x'})}{\partial \boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda})} \frac{\partial \boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}$
  Update hypernetwork weights $\boldsymbol{w}$ by descending $\frac{\partial \mathcal{L}_{train}(\boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}), \boldsymbol{\lambda}, \boldsymbol{x})}{\partial \boldsymbol{w}_{\boldsymbol{\phi}}}$
Derive the final architecture based on the learned $\boldsymbol{\lambda}$

Instead of modelling the supernet $\boldsymbol{w}$ globally, Algorithm 2 only models $\boldsymbol{w}$ locally, i.e., near the current guess of hyperparameter $\boldsymbol{\lambda}$. A global hyperDARTS would first learn a hypernetwork that can output $\boldsymbol{w}$ for any $\boldsymbol{\lambda}$, then only optimize $\boldsymbol{\lambda}$, requiring high hypernetwork capacity and a high search cost. The local hyperDARTS used in this project only needs to train the hypernetwork to estimate good enough $\boldsymbol{w}$ for a small set of $\boldsymbol{\lambda}$.

Optimal $\boldsymbol{\lambda}$ distribution is unknown. Algorithm 2 uses gradient updates on $\boldsymbol{\lambda}$ as a source of noise to form $\boldsymbol{\lambda}$ distribution. This variant has no asymptotic guarantees, but perform similarly to normal distribution as $\boldsymbol{\lambda}$ distribution in practice [45].

4.5 HyperDARTS Macro-Architecture and Performance Estimation Strategy

HyperDARTS follows DARTS to use a supernetwork as performance estimation strategy (discussed in Section 3.3). A supernetwork contains all candidate architectures via weight sharing. For example, if the candidate architecture a and another candidate architecture b contain exactly the same operation (edge) at the same position (say, from node 1 to node 2 in the first normal cell), the weights $\boldsymbol{w}$ of this operation are exactly the same for both a and b.

The hyperDARTS macro-architecture is same as that of DARTS for fair comparison as shown in Fig 23. A smaller macro-architecture is used for cell searching as shown in Fig. 23 (middle). During cell searching, every edge in every cells in the supernetwork contains all possible operations from the predefined operation set so to represent all possible candidate architectures. After edge discretization, each node in each type of cell only keeps two operations (between the node and all its parent nodes). The final normal and reduction cells are then extracted to stack the macro-architecture as shown in Fig. 23 (bottom) to be ready for architecture evaluation.

As shown in Fig. 23, the supernetwork can be dissected into 4 parts: stem preprocessing, (normal and reduction) cell preprocessing, (normal and reduction) cells, and linear layer classifier. Stem preprocessing transforms the input image, which has a shape of $3 \times 32 \times 32$, into a feature map with a shape of $64 \times 32 \times 32$. The linear layer classifier outputs the probability vector for each class.

For a normal cell, cell preprocessing receives the output feature maps from the previous two cells, each has a shape of $4C \times H \times W$. Cell preprocessing then transforms each of these two feature maps into a shape of $C \times H \times W$. The transformed feature maps become $\boldsymbol{h}_{k-2}$ and $\boldsymbol{h}_{k-1}$, which are subsequently fed into the cell. The cell contains 4 intermediate nodes, as shown in Fig. 20. Each intermediate node preserves the spatiality of the input feature map by outputting a feature map with a shape of $C \times H \times W$. As the output of a cell is obtained by depthwise concatenating the outputs from each of its intermediate nodes, the resulting output of a normal cell has the same shape as the input feature maps provided to the cell preprocessing. In other words, it retains the spatial shape of $4C \times H \times W$.

For a reduction cell, cell preprocessing transforms each of the input feature maps from the shape of $4C \times H \times W$ into $2C \times H \times W$, which are subsequently fed into the cell. Each intermediate node outputs a feature map with a shape of $2C \times H/2 \times W/2$. The output of a reduction cell thus has a shape of $8C \times H/2 \times W/2$.

4.6 Proposed HyperDARTS Models

Fig. 24 depicts a hyperDARTS model. As shown in Fig.20, there are $2 + 3 + 4 + 5$ edges within each cell. Each edge consists of 8 operations. Therefore, each hypernetwork is fed 112 hyperparameters. In other words, both $\boldsymbol{\lambda}_{normal}$ and $\boldsymbol{\lambda}_{reduce}$ have a size of 112.

In this project, two hyperDARTS models are developed: hyperDARTS-1 and hyperDARTS-2. The two models differ only in the hypernetworks they employ.

HyperDARTS-1 utilizes linear regression, i.e., **1**-layer linear neural network for its hypernetworks. As discussed in Section 4.3.1, when refering a neural network as N-layer neural network, N is the total number of hidden layers and output layer. Since linear regression has no hidden layer, it is called 1-layer linear network. It is common to employ 1-layer linear network [45] or its variants [46] when applying hypernetworks in a new field, given its simplicity. Since, to the author's knowledge, hypernetworks have never been applied to a DAS model before, this project starts hyperDARTS with

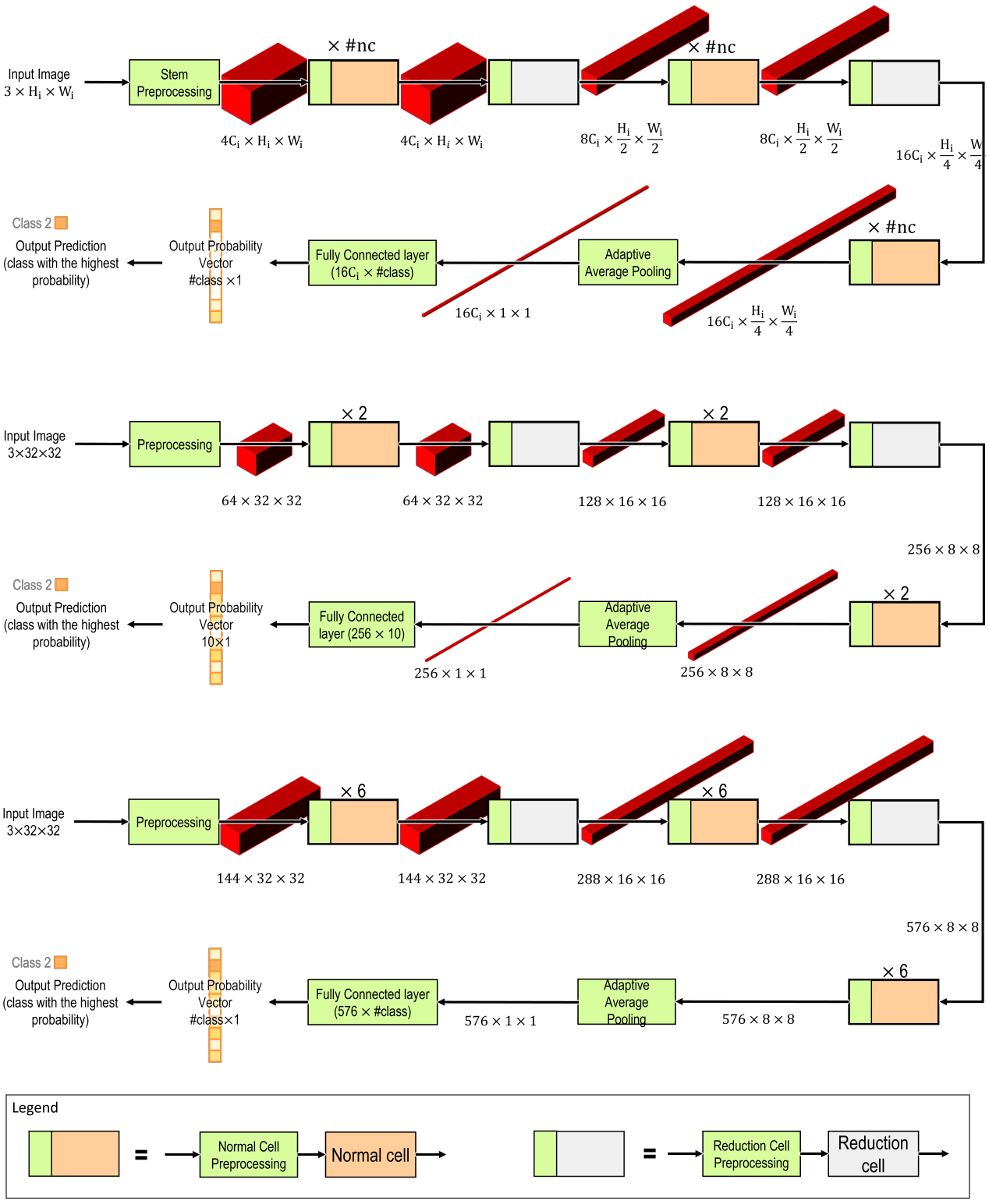


Fig. 23. The general macro-architecture (top). The small macro-architecture used for **cell searching** on CIFAR-10 (middle). The large macro-architecture used for **architecture evaluation** on CIFAR-10 and CIFAR-100 (bottom). The red boxes are feature maps. #class represents the number of classes. CIFAR-10 has 10 classes while CIFAR-100 has 100 classes. Some feature maps are omitted for clarity.

1-layer linear network.

The second model is called hyperDARTS-2. HyperDARTS-2 was developed to further improve hyperDARTS performance. HyperDARTS-2 employs a **2**-layer multilevel perceptron (MLP) model for

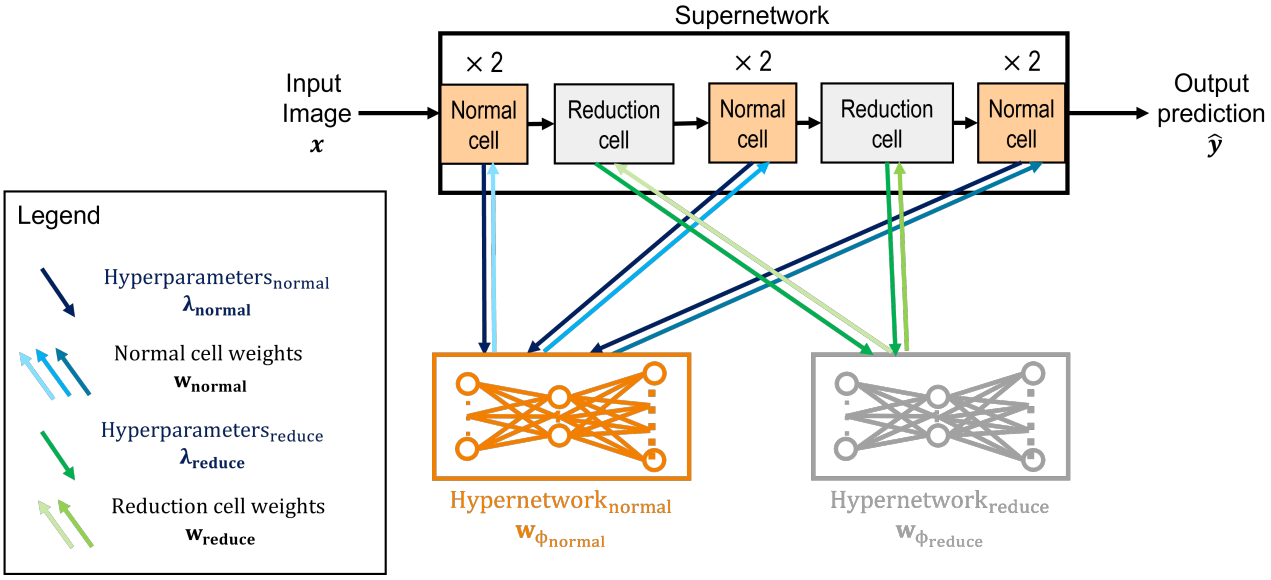


Fig. 24. HyperDARTS-2 model. Each hypernetwork is fed $\boldsymbol{\lambda}$ and outputs $\boldsymbol{w}$. There are 2 types of hyperparameters: hyperparameters for normal cell $\boldsymbol{\lambda}_{normal}$ and for reduction cell $\boldsymbol{\lambda}_{reduce}$. All normal cells have the same hyperparameters $\boldsymbol{\lambda}_{normal}$ but different weights $\boldsymbol{w}_{\boldsymbol{normal}}$, so do all reduction cells. Two hypernetworks are used: one for normal cell and the other for reduction cell.

its hypernetworks. Each hypernetwork is a MLP with 1 hidden layer. The number of nodes in the hidden layer has half the size of the input nodes, resulting in 56 nodes in total. The hidden layer uses sigmoid activation function to introduce non-linearity into hyperDARTS-2.

Experiments conducted for hyperDARTS-1 and hyperDARTS-2 are in Section 5.

4.7 Architecture Selection and Evaluation

Each hyperDARTS model searches for normal and reduction cells according to Algorithm 2 using the CIFAR-10 dataset. Following the architecture selection method of DARTS, this experiment is repeated four times with different random seeds. DARTS determines the best cells based solely on their validation accuracy, selecting the cells that yield the highest validation accuracy across the four runs. HyperDARTS loosely follows this criterion set by DARTS by introducing an additional rule for determining the best cells: it disregards any run that results in a normal cell with no skip connections.

This rule was introduced to address the discrepancy between the shallower macro-architecture used for cell searching and the deeper macro-architecture used for architecture evaluation. It is widely recognized that skip connections alleviate the issue of gradient diminishing in deep neural networks [64]. A shallower neural network experiences less severe gradient diminishing. Consequently, the cells discovered using a shallower macro-architecture may not fully recognize the significance of skip connections, particularly in normal cells where their number is increased for architecture evaluation, resulting in overfitting, i.e., discrepancy in performance between cell searching and architecture evaluation. Nevertheless, the rule added in hyperDARTS does not affect its search cost, as the cell search is still performed 4 times only.

After determining the best cells, the normal and reduction cells stack to form a deeper macro-architecture as shown in Fig. 23 (bottom). The final architecture is then used to perform image recognition on CIFAR-10 and CIFAR-100 for architecture evaluation. The image recognition performance on CIFAR-100 also demosntrates the architecture transferability as the discovered architectures

are never fed any CIFAR-100 image during architecture searching. CIFAR-100 is used as the metric for evaluating architecture transferability because it is a more challenging dataset than CIFAR-10 as it contains 100 classes.

To evaluate the selected architecture, the architecture is trained from scratch with randomly-initialized weights (the cell weights learned during the searching process are discarded). The architecture performance on the test set is then reported. The test set is never used in architecture search or architecture selection.

4.8 Experimental Setup

The experimental setup closely follows DARTS, with five exceptions:

1. the introduction of an additional rule during architecture selection,
2. the batch size used in hyperDARTS-1 cell search,
3. the number of epochs used in hyperDARTS-1 cell search,
4. the hyperparameter learning rate, and
5. the hypernetwork learning rate.

HyperDARTS-1 uses a batch size of 48 instead of 64 due to memory limitation. It also searches for 49 instead of 50 epochs due to overfitting issue. The last two aspects deviate from the original DARTS approach as DARTS does not utilize hypernetworks. The use of hypernetworks in hyperDARTS leads to significant alterations in the magnitude of the hypergradient compared to DARTS, necessitating adjustments to the hyperparameter learning rate.

This project utilizes Python and PyTorch [110] for coding purposes. The code implementation for this project is built upon the existing codebase of DARTS.

4.8.1 Architecture Search

The operation set includes 8 operations: 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity (skip connection), and zero. All operations are of stride one (if applicable) and the convolved feature maps are padded to preserve their spatial resolution. The ReLU-Conv-BN order is used for all convolutional operations, and each separable convolution is always applied twice [19, 29, 87].

Each cell consists of a total of 7 nodes: 2 input nodes, 4 intermediate nodes, and 1 output node. In cell k, the first and second nodes (the two input nodes) are set equal to the outputs of cell $k-2$ and cell $k-1$, respectively. To ensure compatibility in the number of channels, 1×1 convolutions are introduced as needed. The cells positioned at 1/3 and 2/3 of the network's overall depth are designated as reduction cells, where all operations neighboring the input nodes employ a stride of two (if applicable).

The search cost reported in this project is based on single NVIDIA GTX 1080Ti GPU. During architecture search, batch-specific statistics instead of the global moving average is used for batch normal-

ization since the architectures vary throughout the search process. The learnable affine parameters in all batch normalizations are disabled to prevent rescaling the outputs of candidate operations.

Half of the CIFAR-10 training set is used as the validation set. A shallower network consisting of 8 cells is trained using hyperDARTS-1 for 49 epochs and hyperDARTS-2 for 50 epochs. For both the training and validation sets, hyperDARTS-1 uses a batch size of 48 and hyperDARTS-2 uses 64. The initial number of channels C_i is set to 16, ensuring that the network can fit into a single GPU. Momentum stochastic gradient descent is used as the optimization algorithm for the hypernetwork weights $\boldsymbol{w_\phi}$, with an initial learning rate η_{w_ϕ} of 0.05 for hyperDARTS-1 and 0.075 for hyperDARTS-2. The learning rate is annealed down to zero following a cosine schedule without restart. The momentum is 0.9 and the weight decay is 3×10^{-4}.

$\boldsymbol{\lambda}$ for both normal and reduction cells are initialized with 0.001 the standard normal distribution noise, indicating a similar amount of attention (after softmax) over all possible operations. This initialization ensures that weights in every candidate operation receive sufficient learning signal, promoting exploration during the early stage of the search process. Adam is used as the optimizer for $\boldsymbol{\lambda}$, with an initial learning rate η_λ of 6×10^{-4} for hyperDARTS-1 and 9×10^{-4} for hyperDARTS-2. The Adam is set to have momentum values β of (0.5, 0.999), and weight decay of 10^{-3}. The difference in configuration between hyperDARTS-1 and hyperDARTS-2 is shown in Table 1.

Table 1: Difference in configuration between hyperDARTS-1 and hyperDARTS-2.

	Configuration				
Model	batch size	η_{w_ϕ}	η_λ	#epoch	ϕ
hyperDARTS-1	48	0.050	6×10^{-4}	49	1-layer linear
hyperDARTS-2	64	0.075	9×10^{-4}	50	2-layer MLP + sigmoid

4.8.2 Architecture Evaluation

A deeper network consisting of 20 cells, as shown in Fig. 8 (bottom) is trained for 600 epochs, using a batch size of 96. To ensure comparability with other baselines in the literature (around 3M model size), the initial number of channels C_i is increased from 16 to 36. The remaining hyperparameters are kept the same as those used for architecture search. The network utilizes cutout, path dropout with a probability of 0.2, and auxiliary towers with a weight of 0.4.

The training process takes approximately 1.5 GPU days to complete. Considering the high variance observed in CIFAR results, even with an identical setup [25], the mean and standard deviation of 10 independent runs for CIFAR-10 and 5 independent runs for CIFAR-100 are reported for all hyperDARTS models, respectively. The DARTS baseline model reports 5 independent runs for both CIFAR-10 and CIFAR-100.

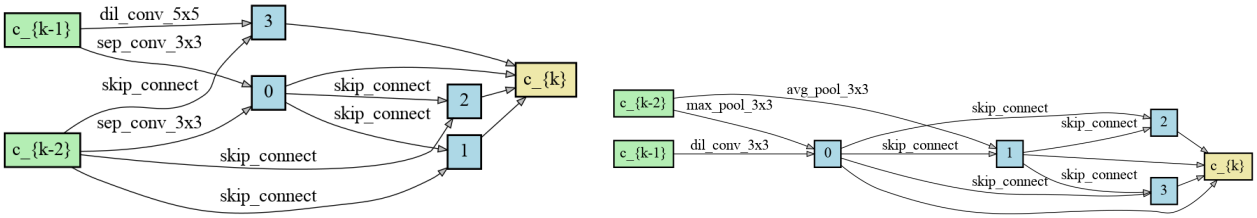
5 Results and Discussion

5.1 DARTS as Baseline Model

To ensure consistency across the experimental environments employed for both DARTS and hyperDARTS, the DARTS experiments were replicated in this project. The results of the DARTS architecture search on CIFAR-10 are presented in Table 2. The resulting architecture, as illustrated in Fig. 25, was obtained from run 4, which achieved the highest validation accuracy. The test performances of DARTS-discovered architecture on CIFAR-10 and CIFAR-100 are presented in Table 3. The search cost, i.e., the GPU hours to execute the 4 runs of architecture search is 48 hours.

Table 2: Validation performance of DARTS across 4 runs of **architecture search**.

Run	Random Seed	Best Validation Accuracy (%)
1	2	88.2840
2	20	87.9720
3	200	88.2480
4	2000	88.4160



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

Fig. 25. DARTS-discovered architecture.

Table 3: Test performances of DARTS-discovered architecture on CIFAR-10 and CIFAR-100 for for **architecture evaluation**.

Best Testing Accuracy (%)		
Run	CIFAR-10	CIFAR-100
1	97.04	82.20
2	97.25	82.22
3	97.31	82.38
4	97.33	82.20
5	97.25	82.51
Mean ± Std. Dev.	97.24 ± 0.10	83.30 ± 0.08

5.2 Experiment 1: HyperDARTS-1

The hypernetworks in hyperDARTS-1 are designed as 1-layer linear networks. This section primarily focuses on experiments conducted for other aspects of hyperDARTS-1. HyperDARTS do not follow DARTS in these aspects because:

1. These aspects fall outside the scope of DARTS, or
2. The DARTS configuration is not applicable to hyperDARTS due to its utilization of a different search strategy.

5.2.1 Experiment 1.1: Weights Outside of Cells

As discussed in Section 4.5, hyperDARTS can be dissected into 4 parts: stem preprocessing, (normal and reduction) cell preprocessing, (normal and reduction) cells, and linear layer classifier. Both stem and cell preprocessing involve convolutional operations, while the linear layer classifier consists of a fully connected layer. Both convolutional operations and the fully connected layer contain weights. Therefore, it was a matter of debate whether these weights outside of cells should:

1. Learn and update on their own, or
2. Be modeled by the hypernetworks.

The weightage of each cell operation $\boldsymbol{\Theta}$ is controlled by $\boldsymbol{\lambda}$. In other words, there exists a direct relationship between the cell operation weights ($\boldsymbol{w_{normal}}$ and $\boldsymbol{w_{reduce}}$) and $\boldsymbol{\lambda}$. Conversely, the weights outside of cells do not have a direct relationship with $\boldsymbol{\lambda}$. Hence, it is unnecessary for the hypernetworks to model the weights outside of the cell, as the hypergradient computation does not rely on these weights. Moreover, the direct relationship between $\boldsymbol{\lambda}$ and the cell operation weights ($\boldsymbol{w_{normal}}$ and $\boldsymbol{w_{reduce}}$) is easier to model compared to the indirect relationship between $\boldsymbol{\lambda}$ and the weights outside of cells. In short, conceptually, method 1 is expected to perform better than method 2, as method 2 demands higher hypernetwork modeling capacity. The first paper that introduces linear hypernetworks for elementary network weight modeling [46] also employs method 1 (although it did not attempt method 2).

Both methods are tested by conducting 2 runs of architecture search. This experiment uses a batch size of 32 and same η_{w_ϕ} ($= 0.025$), η_λ ($= 3 \times 10^{-4}$) and number of epochs (# epoch=50) as DARTS. A batch size of 32 is used because it is among the most common batch sizes (32, 64, 128, 256,...) used in deep learning [111, 112] and it can fit the 12GB GPU memory available. The validation performance is presented in Table 4.

Table 4: Validation performance obtained from architecture search in experiment 1.1.

Best Validation Accuracy (%)			
Run	Random Seed	Method 1	Method 2
1	2	90.0240	89.9600
2	20	90.2360	89.6400
Note: batch size = 32, $\eta_{w_\phi} = 0.025$, $\eta_\lambda = 3 \times 10^{-4}$, #epoch = 50			

The experimental results align with the hypothesis that method 1 outperforms method 2. Therefore, hyperDARTS-1 adopts method 1 for this project.

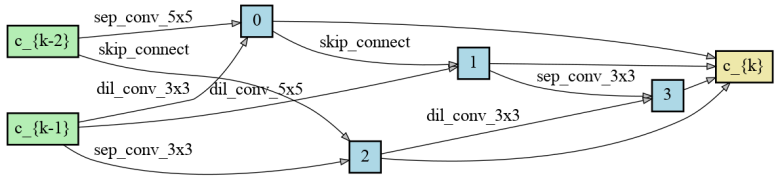
5.2.2 Experiment 1.2: Batch Size and Learning Rate

5.2.2.1 Experiment 1.2.1

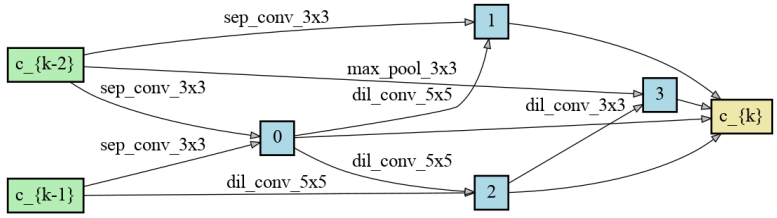
Using method 1 from experiment 1.1 with a batch size of 32, $\eta_{w_\phi} = 0.025$, $\eta_\lambda = 3 \times 10^{-4}$, and #epochs = 50, 4 runs of architecture search are executed. The validation performance obtained during architecture search is presented in Table 5. The resulting architecture, as illustrated in Fig. 26, was obtained from run 2, which achieved the highest validation accuracy. The test performance of the architecture discovered on CIFAR-10 are presented in Table 6. The search cost of is 64 GPU hours.

Table 5: Validation performance obtained from 4 runs of **architecture search** in experiment 1.2.1.

Run	Random Seed	Best Validation Accuracy (%)
1	2	90.0240
2	20	**90.2360**
3	200	90.1440
4	2000	89.8920
Note: batch size = 32, $\eta_{w_\phi} = 0.025$, $\eta_\lambda = 3 \times 10^{-4}$, number of epoch = 50		



(a) Normal Cell



(b) Reduction Cell

Fig. 26. Best architecture discovered by the preliminary hyperDARTS-1 in experiment 1.2.1.

The best architecture shows a higher validation accuracy then DARTS during architecture search. This aligns with the improvement in testing accuracy achieved by the best architecture found by the preliminary hyperDARTS-1 in this experiment compared to DARTS. Even though there is only 0.11% test accuracy improvement on CIFAR-10, this experiment proved the feasibility of hyperDARTS concept and its superiority to DARTS in finding better architectures that can achieve better task performance.

5.2.2.2 Experiment 1.2.2

It is found that, during architecture search, the preliminary hyperDARTS-1 in experiment 1.2.1 only consumes around 67% of the 12GB GPU memory available while DARTS consumes around 91%. This shows that the batch size used in the preliminary hyperDARTS-1 in experiment 1.2.1 can be increased

Table 6: Test performance of the architecture found by the preliminary hyperDARTS-1 in experiment 1.2.1 on CIFAR-10 for **architecture evaluation**.

Best Testing Accuracy (%)	
Run	CIFAR-10
1	97.46
2	97.40
3	97.33
4	97.34
5	97.21
Mean ± Std Dev	97.35 ± 0.08

to 48 that increases memory consumption to around 89%, comparable to DARTS, and thus reduce search cost.

Keeping all other settings unchanged, 4 runs of architecture search are again conducted. The validation performance obtained during architecture search is presented in Table 7. The increase in batch speeds up the search process and reduces the search cost from 64 hours needed in experiment 1.2.1 to 44 GPU hours.

However, the validation accuracies achieved by this preliminary hyperDARTS-1 are significantly lower than that in experiment 1.2.1. This shows that the preliminary hyperDARTS-1 in this experiment found worse architectures than experiment 1.2.1 since there is a positive correlation between validation accuracies during cell search and testing accuracies during architecture evaluation [24].

This deterioration in architecture searching performance is attributed to the increase in batch size. [113] shows that there is a positive correlation between batch size and optimal learning rate. In other words, when the batch size is increased, the learning rate should also increase to achieve optimal results. Therefore, experiment 1.2.3 is conducted to find the optimal learning rates given a batch size of 48 is used.

Table 7: Validation performance obtained from 4 runs of **architecture search** in experiment 1.2.2.

Run	Random Seed	Best Validation Accuracy (%)
1	2	97.8760
2	20	89.2160
3	200	89.6160
4	2000	89.6240
Note: batch size = 32, $\eta_{w_\phi} = 0.025$, $\eta_\lambda = 3 \times 10^{-4}$, #epoch = 50		

5.2.2.3 Experiment 1.2.3

Different from traditional neural networks, there are 2 learning rates in hyperDARTS: η_{w_ϕ} and η_λ. Both η_{w_ϕ} and η_λ are scaled by the same factors in this experiment. For each set of scaled learning rates, only 1 run of architecture search is conducted, using the random seed of 2, due to computational and time constraints. The results are presented in Table 8 and Fig. 27.

$\eta_{w_\phi} = 0.05$ and $\eta_\lambda = 6.0 \times 10^{-4}$ show the best validation performance for architecture search. Therefore, this set of learning rates is used in hyperDARTS-1 in this project.

Table 8: Validation performance of the preliminary hyperDARTS-1 in experiment 1.2.3 with different learning rates in **architecture search**.

Scaling Factor	η_{w_ϕ}	η_λ	Best Validation Accuracy (%)
1.0	0.0250	3.0×10^{-4}	89.8760
1.5	0.0375	4.5×10^{-4}	89.8280
2.0	0.0500	6.0×10^{-4}	90.2280
2.5	0.0625	7.5×10^{-4}	90.0120
3.0	0.0750	9.0×10^{-4}	89.8440

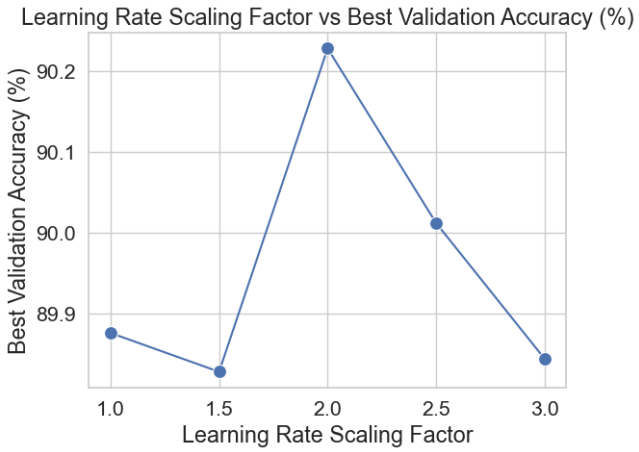


Fig. 27. Validation performance of the preliminary hyperDARTS-1 in experiment 1.2.3 with different learning rates in **architecture search**.

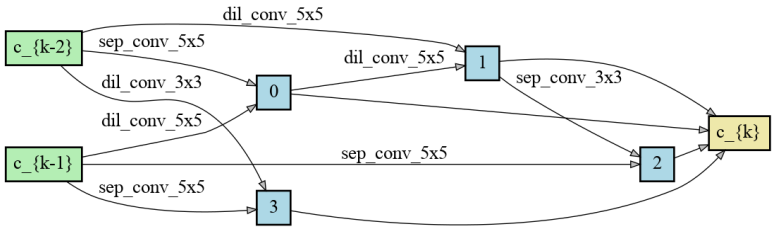
5.2.2.4 Experiment 1.2.4

This experiment uses batch size = 48, $\eta_{w_\phi} = 0.05$, $\eta_\lambda = 6 \times 10^{-4}$, and number of epoch = 50. The validation performance obtained during architecture search is presented in Table 9. The resulting architecture, as illustrated in Fig. 28, was obtained from run 2, which achieved the highest validation accuracy. The test performance of architecture discovered on CIFAR-10 is presented in Table 10. The search cost is 48 GPU hours.

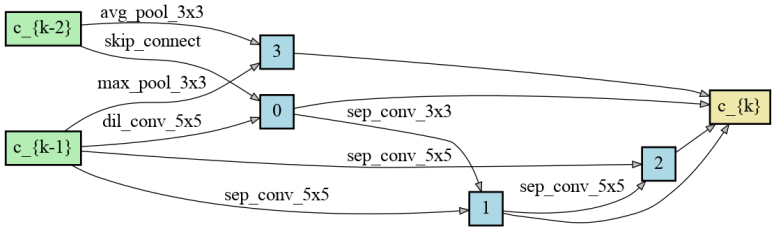
Table 9: Validation performance obtained from 4 runs of **architecture search** in experiment 1.2.4.

Run	Random Seed	Best Validation Accuracy (%)
1	2	90.2280
2	20	**90.3680**
3	200	90.1160
4	2000	89.9040
Note: batch size = 48, $\eta_{w_\phi} = 0.05$, $\eta_\lambda = 6 \times 10^{-4}$, #epoch = 50		

Already visually, one might suspect that the discovered architecture is suboptimal as there is no skip



(a) Normal Cell



(b) Reduction Cell

Fig. 28. Best architecture discovered by the preliminary hyperDARTS-1 in experiment 2.4.

Table 10: Test performance of the architecture found by the preliminary hyperDARTS-1 in experiment 1.2.4 on CIFAR-10 for **architecture evaluation**.

Best Testing Accuracy (%)	
Run	CIFAR-10
1	97.13
2	96.91

connection in its normal cell. The importance of skip connection and why a DARTS-based model can fail to recognize its significance are discussed in Section 4.7. The test performance obtained from architecture evaluation further validates the suspect. The discovered architecture achieves much lower test accuracies than the preliminary hyperDARTS-1 in experiment 1.2.1. Only 2 runs are conducted as the early results show suboptimal architecture found.

The validation accuracy obtained from architecture search fails to align with the testing performance of the architecture found. The discovered architecture in this experiment achieves a validation accuracy of 90.3680% during architecture search, but only 97.13% in its testing accuracy during architecture evaluation. In contrast, experiment 1.2.1 found an architecture that achieves 90.2360% during architecture search and 97.35% during architecture evaluation.

This is because of overfitting. Traditionally, a neural network is trained with training data only. Validation data is used to evaluate the generalization performance of the neural network during network design phase. Testing data is used to evaluate the generalization performance of the neural network after design phase.

In other words, even though validation data is used during model design phase for researchers to evaluate the network performance in order to improve the model design, the model is not trained with validation data. In simple terms, traditional neural networks never see validation images during training. This makes the validation accuracy a rather unbiased metric for evaluating model generalization performance. However, this is not the case in the field of NAS.

During architecture search, NAS involves a two-stage solution to optimize the hyperparameters and to estimate task performance of candidate architectures. It is a common and effective practice to

use validation data for hyperparemeter optimization and training data for performance estimation [19, 24, 25]. In other words, a NAS network is trained with both training and validation data during architecture search. One might argue that the candidate architectures are not trained directly with the training data. However, this does not downplay the fact that, during NAS network architecture search, the candidate arhictectures are updated as a function of hyperparameters and hyperparameters are updated as a function of validation data as discussed in Section 4.4.3.

As validation data is used in searching the optimal candidate architecture, the discovered architecture can overfit to the validation data. In this case, the validation accuracy is no longer an unbiased metric to evaluate the generalization performance of the discovered architecture.

One might argue that the 4 runs, each using a different random seed, conducted during architecture search can be seen as a type of regularization method [114] as model stability is a function of random seed (random-seed based perturbation) [115]. In simple terms, the overfitting issue can be resolved just by selecting another architecture found from the 4 runs, say, the second best-performing architecture from run 1.

This simple strategy might work and it is in fact used for hyperDARTS-2. However, it is not used to alleviate the overfitting issue in this experiment for this preliminary hyperDARTS-1. This is because the second best-performing architecture also contains no skip connection in its normal cell. Resolving to the third best-performing architecture might not be a good idea as the inferiority of this preliminary hyperDARTS-1 model is obvious (even second best-performing architecture has an overfitting issue). Therefore, next experiment attempts to alleviate the overfitting issue using another strategy. To save computational and time resources, an architecture found from any run of architecture search is identified as overfitted (without conducting architecture evaluation) if there is no skip connection/s in its normal cells. The justification is provided in Section 4.7.

5.2.3 Experiment 1.3: Number of Epochs

The overfitting issue also happens in DARTS model when changing search space [114] and using CIFAR-100 for architecture search [116]. In DARTS-related studies [49, 117, 118], the issue is generalized as the optimization gap, i.e., performance drop after pruning (discretization). Attempts to resolve the performance drop in DARTS include early stopping [80, 116], regularization [114, 114, 118], and progressively increasing architecture depth during architecture search [49].

Early stopping is the easiest approach as it only involves changing the number of epoch for architecture search. Therefore, this experiment approaches the overfitting issue using a naive method: manually adjusting the number of epochs. For each value, only 1 run of architecture search is conducted, using the random seed of 2, due to limiting computing and time resources. The results are presented in Table 16.

This experiment does not adopt adaptive early stopping by automatically stopping the search procedure based on manually-designed criterion such as the number of skip connection/s in normal cell [116]. It is because the author has experimented with increasing the maximum number of epochs for architecture search to 100 and found that, interestingly, the model reaches only around 86% in validation accuracy at epoch 50. This is observed in all runs (more than 10). Note that the preliminary hyperDARTS-1 models reach around 90% at epoch 50 when setting the maximum number of epochs as 50 as demonstrated in experiment 1.2.1, 1.2.2, 1.2.3, and 1.2.4. The maximum number of epochs equals the number of epochs if no adaptive early stopping strategy is applied.

Based on this observation, it is suspected that PyTorch adjusts its exploration-exploitation strategy

according to the maximum number of epochs. Assuming PyTorch optimizes its non-deterministic algorithms based on this value, modifying the maximum number of epochs appears to be a preferable option compared to adopting an adaptive early stopping strategy that fixes the maximum number of epochs. However, the author refrained from further delving into the complexity of how PyTorch performs its non-deterministic algorithms due to time constraints.

A number of epochs of 49 results in the highest validation accuracy with skip connection/s in the normal cell. Therefore, the same configuration is used to repeat the 4 runs of architecture search with random seeds. If there is any overfitting architecture found within the 4 runs with the number of epochs being 49, the number of epochs will then be further reduced until there is no overfitting architecture found within the 4 runs.

Table 11: Validation performance of the preliminary hyperDARTS-1 in experiment 1.3 with different number of epochs in **architecture search**.

Number of Epochs	Best Validation Accuracy (%)	Skip Connection/s in Normal Cell?
45	89.6920	Yes
48	90.0200	Yes
49	90.2000	Yes
50	90.2280	No

The validation performance during architecture search is presented in Table 12. The resulting architecture, as illustrated in Fig. 29, was obtained from run 1, which achieved the highest validation accuracy with no overfitting. The test performances of the hyperDARTS-1-discovered architecture on CIFAR-10 and CIFAR-100 are presented in Table 13. The search cost is 47 GPU hours. The hyperDARTS-1 model used in this experiment is the final hyperDARTS-1.

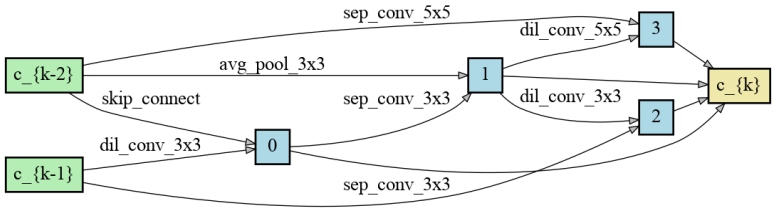
Table 12: Validation performance of hyperDARTS-1 across 4 runs of **architecture search** in experiment 1.3.

Run	Random Seed	Best Validation Accuracy (%)	Skip Connection/s in Normal Cell?
1	2	90.2000	Yes
2	20	89.8120	Yes
3	200	90.0720	Yes
4	2000	90.1120	Yes
Note: batch size = 48, $\eta_{w_\phi} = 0.05$, $\eta_\lambda = 6 \times 10^{-4}$, #epoch = 49			

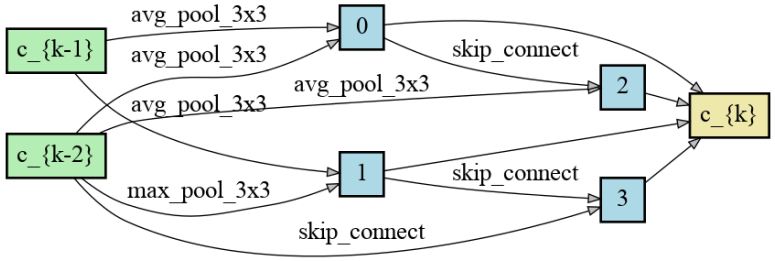
5.2.4 HyperDARTS-1 vs DARTS

From experiment 1.3, HyperDARTS-1 adopts a batch size of 48, $\eta_{w_\phi} = 0.05$, $\eta_\lambda = 6 \times 10^{-4}$, and a number of epochs being 49 for architecture search. These are the only experimental settings where HyperDARTS-1 differs from DARTS.

The architecture discovered by DARTS achieves test accuracies of 97.24% and 82.30% on CIFAR-10 and CIFAR-100, respectively. On the other hand, the architecture discovered by HyperDARTS-1 achieves test accuracies of 97.40% and 83.38%, respectively. Compared to the DARTS-discovered architecture, the HyperDARTS-1-discovered architecture demonstrates better task performance in two aspects:



(a) Normal Cell



(b) Reduction Cell

Fig. 29. Best architecture discovered by hyperDARTS-1 in experiment 2.5.

Table 13: Test performance of the architecture found by hyperDARTS-1 in experiment 1.3 on CIFAR-10 and CIFAR-100 for for **architecture evaluation**.

Best Testing Accuracy (%)		
Run	CIFAR-10	CIFAR-100
1	97.42	83.39
2	97.53	83.67
3	97.25	83.38
4	97.36	83.55
5	97.37	82.92
6	97.49	-
7	97.33	-
8	97.40	-
9	97.36	-
10	97.48	-
Mean ± Std. Dev.	97.40 ± 0.08	83.38 ± 0.25

1. It exhibits a higher image recognition capability, with higher testing accuracies on both CIFAR-10 and CIFAR-100.
2. It demonstrates higher transferability, or generalization capability, with a higher testing accuracy on CIFAR-100.

In summary, the aim of this project, which was to develop a new DARTS-based model (hyperDARTS-1) capable of finding architectures with superior image recognition performance compared to DARTS, has been achieved. All the objectives have also been fulfilled in Experiment 1.

However, the overfitting problem of hyperDARTS-1 has been addressed through a simple manual early stopping technique. It is important to note that early stopping does not modify the loss landscape of the model. In other words, early stopping does not improve the model performance but rather prevent the model from becoming worse. Consequently, in Experiment 2, hyperDARTS-2 attempts to utilize strategies that can alter the landscape to reduce the overfitting issue and thus improve hyperDARTS performance in finding optimal architectures.

5.3 Experiment 2: HyperDARTS-2

Overfitting is a widely recognized issue in supervised machine learning which prevents the learnt model from generalizing well to the unseen data [119]. An overfitted model memorized the training data, including the unavoidable noise, rather than learning the discipline hidden behind the training data that is also applicable to the unseen data.

The common solutions include early stopping, network reduction, increased training data, and regularization. Early stopping [114, 116], increased training data [116], and regularization [80, 81] have been successfully applied to reduce overfitting in DARTS-based models. However, to the best of the author's knowledge, network reduction has never been applied to DARTS-based models. It is because the supernetwork used in DARTS-based models is intended to be over-parameterized to allow weight sharing between candidate architectures. In other words, reducing the supernetwork capacity would counter its effectiveness in weight sharing.

However, hyperDARTS employs a hypernetwork to model the supernetwork weights. This signifies that it is possible to reduce the hypernetwork capacity to alleviate overfitting in hyperDARTS, given the reduced hypernetwork does not underfit. Network reduction reduces the network complexity so that the model has not enough capacity to model the unwanted noise but the underlying useful patterns from the training data. A simple illustration of this concept is shown in Fig. 30.

Network capacity often links to network complexity. The more complex a neural network, the higher its capacity. There are several approaches to define and quantify the complexity of a neural network, including architectural complexity [121], geometric complexity [122, 123], complexity of the implemented function [124], and the generalization capability of limited models based on the Vapnik-Chervonenkis Dimension [125]. However, none of these approaches provide a comprehensive description of neural network complexity and only allow for complexity comparisons between neural networks with similar configurations.

Therefore, hypernetwork modification for hyperDARTS-2 begins with the simplest metric: architectural complexity. This metric encompasses various measures such as the number of parameters, the number of layers, and the number of nodes. The lower the number of parameters, the lower the architectural complexity of the model. Considering the well-known issue of hypernetworks having excessive parameters $\boldsymbol{w}_\phi$ [45], reducing the number of hypernetwork parameters (hypernetwork weights)

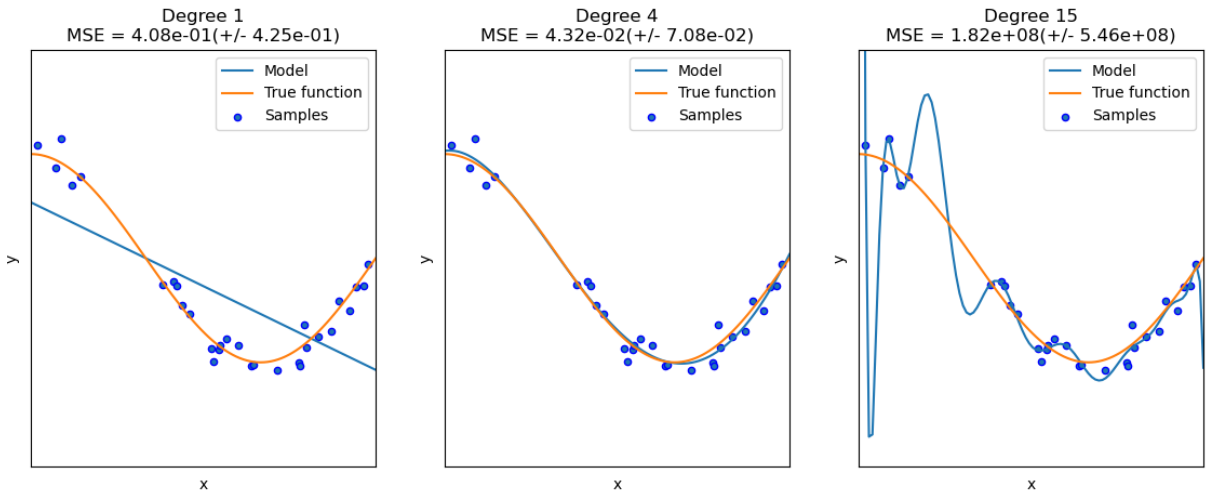


Fig. 30. Polynomial regression to demonstrate model underfitting due to too low complexity (left), good fit with just right complexity (middle), and model overfitting due to too high complexity (right) [120].

$\boldsymbol{w}_\phi$ is chosen as the starting point in modifying the hypernetwork. In fact, hyperDARTS-1 contains 207 million trainable parameters, primarily due to the massive number of hypernetwork parameters $\boldsymbol{w}_\phi$ (number of $\boldsymbol{\lambda}_{normal}\times$ number of normal cell operation weights + number of $\boldsymbol{\lambda}_{reduce}\times$ number of reduce cell operation weights).

The large number of trainable parameters in hyperDARTS-1 imposed a constraint on its batch size, limiting it to 48, whereas DARTS can accommodate a batch size of 64. For reference, DARTS has less than 2 million trainable parameters. One might wonder why hyperDARTS-1, with over 100 times the number of trainable parameters, can utilize a batch size that is 3/4 the DARTS value. The reason lies in the hypergradient approximation scheme of DARTS, which involves inner weight optimization that also consumes substantial GPU memory.

5.3.1 Experiment 2.1: Hypernetworks

5.3.1.1 Experiment 2.1.1: Embedding Vector

There are 3 main types of operations in each cell: convolutional operations, pooling operations, and zero or skip operations. Among these, only convolutional operations involve kernels with weights that are modeled by the hypernetworks. The kernel dimensions vary across cells but are integer multiples of a basic size. This characteristic of varying kernel dimensions in integer multiples of a basic size with depth is common in the widely-utilized residual network family of architectures [64].

[46] leverages the consistently varying kernel dimensions in its elementary CNN by employing a small hypernetwork to output kernel weights of the basic size. For layers with larger kernels, multiple sets of basic kernel weights are sequentially modeled by the hypernetwork. These basic kernel weights are then concatenated to form the larger kernel. Each basic kernel is represented by an embedding vector $\boldsymbol{z}$, which is fed into the hypernetwork to produce the corresponding basic kernel weights. An example can best illustrate this concept.

Take the first convolutional layer in the 3×3 Separated Convolution (sepconv3 $\times$ 3) operation in normal cells of hyperDARTS as an example (each sepconv3 $\times$ 3 operation consists of two back-to-back

convolutional layers). Each convolutional layer has a kernel dimension of $C_{in} \times C_{out} \times k \times k$, where C_{in} is the number of input channels, C_{out} is the number of output channels, and k represents the kernel height, which is the same as the kernel width. The kernel details are presented in Table 14. Note that cells 0, 1, 3, 4, 6, and 7 are normal cells, while cells 2 and 5 are reduction cells in hyperDARTS.

Table 14: Dissecting the kernels in the first convolutional layer in sepconv3×3 in normal cells into basic kernels.

Cell	Kernel Size in 1st Convolutional Layer ($C_{in} \times C_{out} \times k \times k$)	Number of Basic Kernels
0 & 1	$1 \times 16 \times 3 \times 3 = 144$	1
3 & 4	$1 \times 32 \times 3 \times 3 = 288$	2
6 & 7	$1 \times 64 \times 3 \times 3 = 576$	4

In this example, a small hypernetwork is utilized to generate the weights for a basic kernel. This hypernetwork receives a vector (formed by concatenating hyperparameters and an embedding vector $\boldsymbol{z}$ that represents a basic kernel) as input. By employing 13 different embedding vectors, the hypernetwork can model 13 distinct sets of basic kernel weights. 4 of which are concatenated to form the kernel in cell 6 in this example. The size of the hypernetwork is given by $(\#\boldsymbol{\lambda}_{normal} + \#z) \times 144 = (112 + \#z) \times 144$, where # denotes the number of.

It is important to note that for all operations in all cells, each cell only has a different value of C_{in} and/ or C_{out} from the others. A kernel of size $C_{in} \times C_{out} \times k \times k$ can be understood as a collection of $C_{in} \times C_{out}$ $k \times k$ kernels. Consequently, generating a basic kernel can be viewed as, in this example, producing 16 sets of 3×3 kernel weights. Similarly, the second basic kernel produced using the second $\boldsymbol{z}$ represents another set of 16 3×3 kernel weights. This approach ensures that the coherent structure of any 3×3 kernel is maintained, as all 9 weights of any 3×3 kernel always originate from the same output nodes of the same hypernetwork. In hyperDARTS, the hypernetworks employed are either linear regression or MLP. Consequently, in this example, the first 9 outputs of the hypernetwork constitute a 3×3 kernel, the subsequent 9 outputs form the next 3×3 kernel, and so on.

This approach allows for the adjustment of the number of hypernetwork parameters using $\#\boldsymbol{z}$. However, before exploring the relationship between $\#\boldsymbol{z}$ and usable batch size, the feasibility of using embedding vectors to reduce hypernetwork size in hyperDARTS is first validated. For this purpose, an architecture search run is conducted using this approach, using a batch size of 48 and $\#\boldsymbol{z} = 32$. All the other configurations are same as experiment 1.3.

Unfortunately, this approach failed to yield satisfactory results due to its exceptionally high search cost: 20 GPU hours for a single run and 80 GPU hours for 4 runs. It is worth noting that DARTS (with a batch size of 64) required 48 GPU hours for 4 runs, while hyperDARTS-1 (with a batch size of 48) took 47 GPU hours. The substantial search cost arises from the fact that the hypernetworks must sequentially output the weights for each basic kernel. As a result, the backpropagation process, which updates the hypernetworks and hyperparameters using each set of basic kernel weights, becomes sequential as well. The inability of this approach to be applied in hyperDARTS poses a challenge for the author in referencing many modern hypernetworks, as they are built upon this method [104, 126, 127].

5.3.1.2 Experiment 2.1.2: 2-layer Linear Network

Another approach to reduce the number of parameters in the linear hypernetworks of hyperDARTS-1 is to employ 2-layer linear networks. Linear regression, i.e., 1-layer linear network, used in hyperDARTS-1 has only a input layer and a output layer, without any hidden layer. This results in the two

hypernetworks of size $\#\boldsymbol{\lambda}_{normal} \times \#\boldsymbol{w}_{normal}$ and $\#\boldsymbol{\lambda}_{reduce} \times \#\boldsymbol{w}_{reduce}$. $\#\boldsymbol{\lambda}_{normal}$ and $\#\boldsymbol{\lambda}_{reduce}$ are only 112, but $\#\boldsymbol{w}_{normal}$ and $\#\boldsymbol{w}_{reduce}$ are around 1 million.

This experiment adds an additional layer between the input layer and the output layer. This layer is known as the hidden layer. By setting the number of nodes in the hidden layer (the number of hidden nodes) to be smaller than $\#\boldsymbol{\lambda}(= 112)$, the number of hypernetwork parameters $\boldsymbol{w}_\phi$ can be reduced. Reducing the number of $\boldsymbol{w}_\phi$ should decrease model complexity and improve generalization capability, provided that the model does not underfit [128]. For this experiment, $\#\boldsymbol{\lambda}/2$ is chosen as the number of hidden nodes. The number is based on $\#\boldsymbol{\lambda}$ so that the number of hypernetwork parameters $\boldsymbol{w}_\phi$ is always reduced even when a different $\#\boldsymbol{\lambda}$ is used.

With the introduction of the hidden layer, the number of trainable parameters is reduced to 113 million, allowing a batch size of 64 to be used for architecture search while consuming a similar amount of GPU memory as DARTS. Given the increased batch size, different learning rates are once again experimented with to determine the optimal learning rates for this batch size. The validation performance is presented in Table 15. The search cost is 44 GPU hours.

Table 15: Validation performance with different learning rates in experiment 2.1.2 for **architecture search**.

Learning Rate Scaling Factor	η_{w_ϕ}	η_λ	Validation Accuracy (%)	Skip Connection in Normal Cell?
2.0	0.0500	6.0×10^{-4}	90.1040	Yes
2.5	0.0625	7.5×10^{-4}	90.1800	No
3.0	0.0750	9.0×10^{-4}	90.2720	No
3.5	0.0875	10.5×10^{-4}	90.2600	No
4.0	0.1000	12.0×10^{-4}	90.5040	No
Note: Learning rate scaling factors are based on DARTS learning rate.				

Despite the larger batch size, increasing the learning rates does not appear to be a favorable approach as the model tends to overfit when higher learning rates are used. The current model shows a propensity for overfitting the loss landscape.

A larger batch size often results in smaller gradients according to triangle inequality [113]. A hypothetical scenario illustrating this concept is shown in Fig. 31. This observation holds true in hyperDARTS as well. The average magnitudes of hypergradients of 8 hyperparameters of a randomly-selected edge in a random-selected normal cell, obtained from 3 randomly-selected upper-level steps are shown in Fig. 32. As discussed in Section 4.4.3, an upper-level step updates hypergradients.

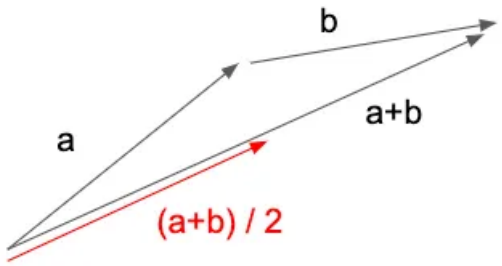


Fig. 31. Gradient is $a + b$ when batch size $= 1$ and $\frac{a+b}{2}$ when batch size $= 2$ [113].

In other words, if the loss landscape has no major changes, increasing the batch size allows the use

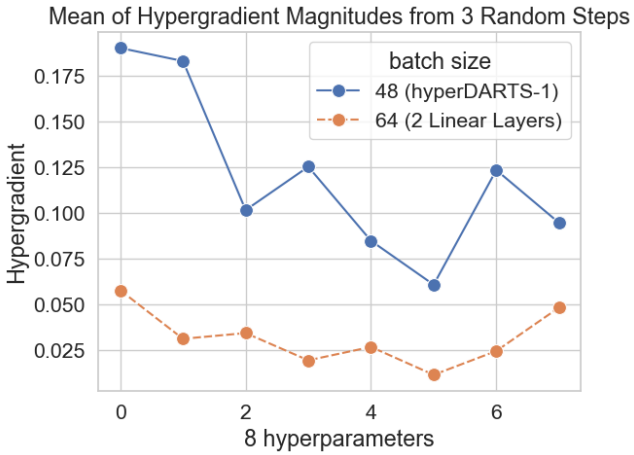


Fig. 32. The mean of hypergradient magnitdues of a random edge in a random normal cell, obtained from 3 randomly-selected upper-level steps. As discussed in Section blah, an upper-level step updates hypergradients. The blue line represents hyperDARTS-1 which uses a batch size of 48 while the orange line represents the modified hyperDARTS in experiment 2.1.2 which uses a batch size of 64.

of larger learning rates, resulting in comparable or even improved validation performances in terms of accuracy and model fitting compared to a smaller batch size with smaller learning rates. However, it appears that the model in this experiment significantly alter the loss landscape, making the model more susceptible to sharp minima. A model optimized to a sharp minimum is prone to overfit [114]. In short, hyperDARTS with 2-layer linear hypernetworks is susceptible to overfitting and shows no performance improvement compared to hyperDARTS-1.

5.3.1.3 Experiment 2.1.3: Non-linear Activation Function

[129] shows that a 2-layer MLP with a non-linear activation function results in a flatter landscape around the true solution than 1-layer and 2-layer linear networks. Therefore, this experiment employs 2-layer MLPs for hypernetworks. 3 common types of non-linear activation functions: tanh, sigmoid, and ReLU are experimented with. For each type of non-linear activation function, only 1 run of architecture search is conducted, using the random seed of 2. The results are presented in Table 16.

Table 16: Validation performance obtained from **architecture search** in experiment 2.1.3. Different non-linear activation functions are used.

Non-linear Activation Function	Best Validation Accuracy (%)	Skip Connection/s in Normal Cell?
tanh	89.9880	Yes
sigmoid	90.3320	Yes
ReLU	90.1920	Yes
Note: batch size = 64, $\eta_{w_\phi} = 0.05$, $\eta_\lambda = 6 \times 10^{-4}$, number of epoch = 50		

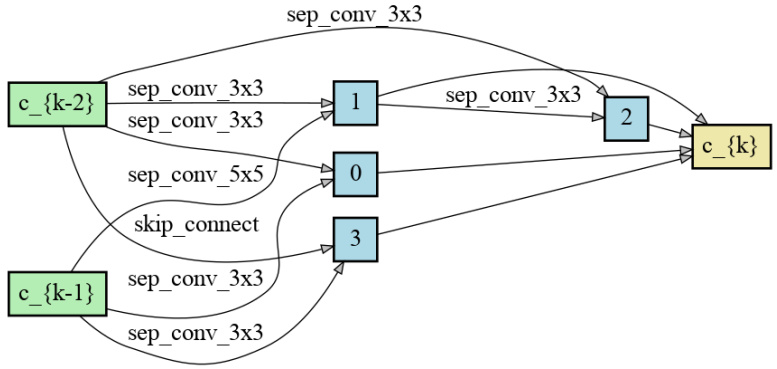
Sigmoid shows the best validation accuracy without overfitting issue. Therefore, sigmoid is used subsequently to conduct 4 runs of architecture search. The results are presented in Table 17. All found architectures from the 4 runs of architecture search have skip connection/s in their normal cells. It signifies that a 2-layer MLP with sigmoid is superior than 2-layer linear networks for $\boldsymbol{w}$ modelling such that the landscape around the true solution is flatter, incurring no overfitting.

The resulting architecture, as illustrated in Fig. 33, was obtained from run 3, which achieved the

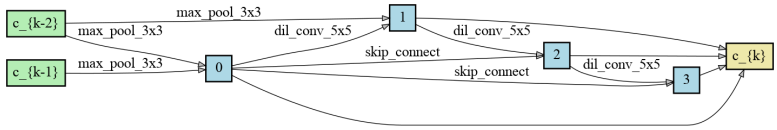
highest validation accuracy. The test performance of DARTS-discovered architecture on CIFAR-10 are presented in Table 18. The search cost is 44 hours.

Table 17: Validation performance obtained across 4 runs of **architecture search** in experiment 2.1.3. The hypernetworks are 2-layer MLP with sigmoid.

Run	Random Seed	Best Validation Accuracy (%)	Skip Connection/s in Normal Cell?
1	2	90.3320	Yes
2	20	90.0960	Yes
3	200	90.5440	Yes
4	2000	90.2520	Yes
Note: batch size = 64, $\eta_{w_\phi} = 0.05$, $\eta_\lambda = 6 \times 10^{-4}$, #epoch = 50, ϕ = 2-layer MLP + sigmoid			



(a) Normal Cell



(b) Reduction Cell

Fig. 33. Best architecture discovered by the preliminary hyperDARTS-2 in experiment 2.1.3.

5.3.2 Experiment 2.2: Learning Rate

Experiment 2.1.2 uses 2-layer linear hypernetworks, resulting in a failure of increasing learning rates despite a larger batch size was used. This experiment uses 2-layer MLP with sigmoid acquired from experiment 2.1.3 and experiments with different larger learning rates. The results are presented in Table 19. Note that experiment 2.1.3 uses 2 times the DARTS values for its learning rates.

$\eta_{w_\phi} = 0.075$ and $\eta_\lambda = 9 \times 10^{-4}$ show the best validation performance for architecture search. Therefore, this set of learning rates is used for hyperDARTS-2. The final hyperDARTS-2 uses the configuration of batch size = 64, $\eta_{w_\phi} = 0.075$, $\eta_\lambda = 9 \times 10^{-4}$, #epoch = 50, and ϕ = 2-layer MLP + sigmoid, 4 runs of architecture search are conducted. The results are presented in Table 20. Among the 4 architectures found across the 4 runs, the architecture with the highest validation accuracy shows overfitting behaviour. Nevertheless, the author has experimented with running 2 more times the 4 runs of architecture search (in total 8 more runs), only 1 overfitting architecture appears every 4 runs. In other words, hyperDARTS-2 seems to always have all the other 3 architectures found have no overfitting issues when 4 runs are conducted. Due to its consistent performance and time constraints, the author settled with this configuration as the final hyperDARTS-2 by choosing the architecture with the second highest validation accuracy and no overfitting.

Table 18: Test performance obtained from **architecture evaluation** in experiment 2.1.3. ϕ = 2-layer MLP + sigmoid.

Best Testing Accuracy (%)	
Run	CIFAR-10
1	97.42
2	97.37
3	97.38
4	97.40
5	97.44
6	97.53
7	97.35
8	97.43
9	97.46
10	97.43
Mean ± Std. Dev.	97.42 ± 0.05

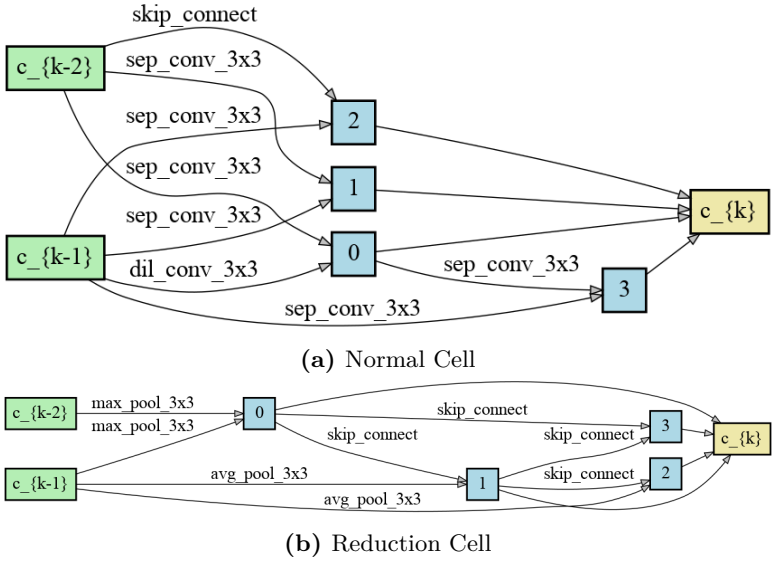
Table 19: Validation performance obtained from **architecture search** in experiment 2.2. Different learning rates are used.

Learning Rate Scaling Factor	η_{w_ϕ}	η_λ	Validation Accuracy (%)	Skip Connection in Normal Cell?
2.0	0.0500	6.0×10^{-4}	90.3320	Yes
2.5	0.0625	7.5×10^{-4}	90.5080	Yes
3.0	0.0750	9.0×10^{-4}	90.6000	Yes
Note: Learning rate scaling factors are based on DARTS learning rate.				

The resulting architecture, as illustrated in Fig. 25, was obtained from run 1, which achieved the second highest validation accuracy with no overfitting. The test performances of hyperDARTS-2-discovered architecture on CIFAR-10 and CIFAR-100 are presented in Table 21. The search cost is 44 hours.

Table 20: Validation performance obtained across 4 runs of **architecture search** in experiment 2.2.

Run	Random Seed	Best Validation Accuracy (%)	Skip Connection/s in Normal Cell?
1	2	90.6000	Yes
2	20	90.5960	Yes
3	200	90.9440	No
4	2000	90.2200	Yes
Note: batch size = 64, $\eta_{w_\phi} = 0.075$, $\eta_\lambda = 9 \times 10^{-4}$, #epoch = 50, ϕ = 2-layer MLP + sigmoid			



(a) Normal Cell

(b) Reduction Cell

Fig. 34. Best architecture discovered by the final hyperDARTS-2 in experiment 2.2.

Table 21: Test performance obtained from **architecture evaluation** in experiment 2.4.

Best Testing Accuracy (%)		
Run	CIFAR-10	CIFAR-100
1	97.48	83.77
2	97.47	83.83
3	97.64	83.51
4	97.40	83.51
5	97.44	83.45
6	97.38	-
7	97.34	-
8	97.44	-
9	97.63	-
10	97.41	-
Mean ± Std. Dev.	97.46 ± 0.09	83.61 ± 0.15

5.4 HyperDARTS vs DARTS

Table 22: Test performances of the best architectures found by DARTS, hyperDARTS-1 and hyperDARTS-2.

	Configuration						Best Testing Accuracy (%)	
Model	batch size	η_{w_ϕ}	η_λ	#epoch	ϕ	Search Cost (GPU hours)	CIFAR-10	CIFAR-100
DARTS	64	0.025	3×10^{-4}	50	-	48	97.24 ± 0.10	82.30 ± 0.08
hyperDARTS-1	48	0.050	6×10^{-4}	49	1-layer linear	47	97.40 ± 0.08	83.38 ± 0.25
hyperDARTS-2	64	0.075	9×10^{-4}	50	2-layer MLP + sigmoid	44	97.46 ± 0.09	83.61 ± 0.15

An overview of task performances achieved by the architectures found by DARTS, hyperDARTS-1 and hyperDARTS-2 are presented in Table 22. The architecture discovered by either hyperDARTS model shows better task performance than DARTS-discovered architecture in two aspects:

1. It exhibits a higher image recognition capability, with higher testing accuracies on both CIFAR-10 and CIFAR-100.
2. It demonstrates higher transferability, or generalization capability, with a higher testing accuracy on CIFAR-100.

The search cost also shows minor improvements. In summary, the aim of this project, which was to develop a new DARTS-based model (hyperDARTS-1 and hyperDARTS-2) capable of finding architectures with superior image recognition performance compared to DARTS, has been achieved. All the objectives have also been fulfilled in Experiment 1 and Experiment 2.

HyperDARTS-2 shows better performance in finding optimal architectures than hyperDARTS-1 due to the different hypernetworks used. The 2-layer MLPs with sigmoid used in hyperDARTS-2 result in a different loss landscape compared to hyperDARTS-1 which employs 1-layer linear networks. The hyperDARTS-2 loss landscape seems to be flatter around the true solution, reducing the overfitting issue in hyperDARTS-1. The non-linear capacity of hypernetworks used in hyperDARTS-2 also allows $\boldsymbol{w}$ to be modelled more accurately compared to hyperDARTS-1, resulting in architectures with higher task performance.

The absolute improvements might seem trivial and it is sensible to put them into a broader context by comparing the hyperDARTS performance with other state-of-the-art (SOTA) DARTS variants. Since most DAS-based models are as well DARTS variants, these DARTS variants are also SOTA DAS models. The comparison is shown in Fig. 35 and Table. 23. All the results of the SOTA DARTS variants are borrowed from [80]. All the models shown, including hyperDARTS, have a similar parameter size for their final architecture (3 millions to 4 millions).

Overall, hyperDARTS has managed to achieve comparable improvements to other SOTA DARTS variants on both CIFAR-10 and CIFAR-100, showing its ability to discover architectures with high image recognition capability and transferability. It is worth noting that most SOTA DARTS variants still employ the search strategy used in DARTS. Therefore, hyperDARTS can be used to combine with other SOTA DARTS variants to result in further improvements.

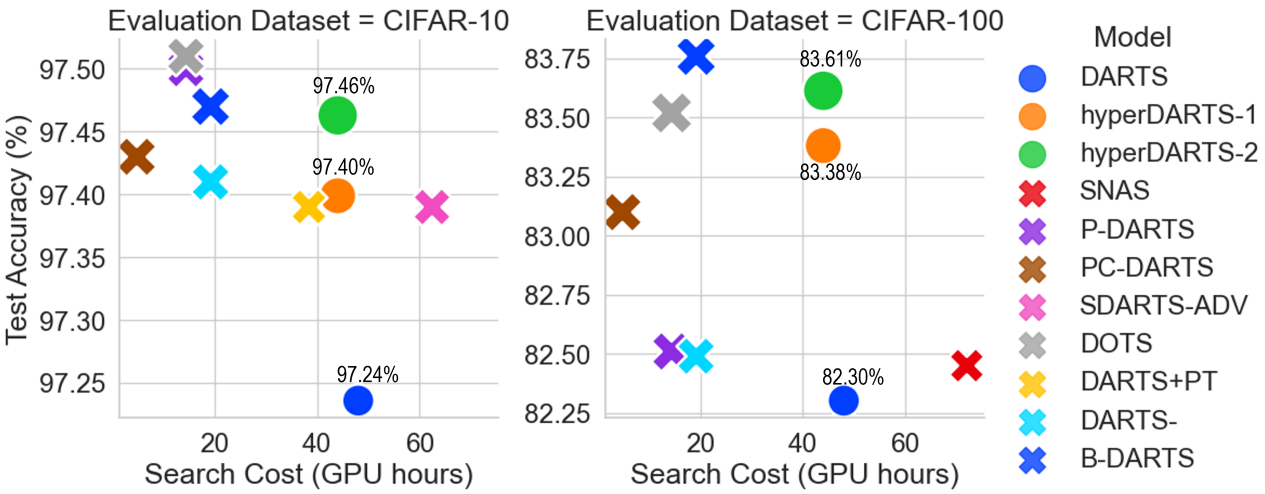


Fig. 35. Comparison of SOTA DARTS variants.

Table 23: Comparison of SOTA DARTS variants.

		Best Testing Accuracy (%)	
Model	Search Cost (GPU days)	CIFAR-10	CIFAR-100
SNAS [54]	3	97.15 ± 0.02	82.45
P-DARTS [130]	0.2	97.50	82.51
PC-DARTS [53]	0.2	97.43 ± 0.07	83.10
SDARTS-ADV [81]	2.6	97.39 ± 0.02	-
DOTS [131]	0.6	97.51 ± 0.06	83.52 ± 0.13
DARTS+PT [132]	1.6	97.39 ± 0.08	-
DARTS- [133]	0.8	97.41 ± 0.08	82.49 ± 0.25
β-DARTS[80]	0.8	97.47 ± 0.08	83.76 ± 0.22
DARTS[24]	2	97.24 ± 0.10	82.30 ± 0.08
hyperDARTS-1	2	97.40 ± 0.08	83.38 ± 0.25
hyperDARTS-2	1.8	97.46 ± 0.09	83.61 ± 0.15

5.5 Findings

Overall, integrating hypernetworks into DARTS, resulting hyperDARTS, eliminates the approximation error in DARTS and changes the loss landscape such that hyperDARTS can find better architectures with a lower search cost and better task performance. Two hypernetworks have been implemented and both provide sufficient capacities to model $\boldsymbol{w}$ locally.

The hypothesis stated that while DARTS can find architectures, with low computational resources, that achieve competitive task performance in terms of test error, its abbreviated approximating optimization algorithm reduces itself to only converge to suboptimal architectures during architecture search. In other words, there is room to improve DARTS performance in terms of computational resources for searching and task performance of discovered architectures by replacing the one-step unrolling appproximation scheme with a more efficient and stable hyperparameter optimization method.

The hypothesis has been validated through hyperDARTS models that replaced the one-step unrolling approximation scheme with hypernetworks. HyperDARTS can find better architectures than DARTS that demonstrate better task performance with a lower search cost.

The project aim and objectives were to develop a new NAS model based on a cutting-edge DAS model, i.e., DARTS, by integrating one of the latest hyperparameter optimization algorithms, i.e., hypernetworks, into the search strategy to improve its performance on finding optimal architectures. The architecture optimality would be evaluated based on the architecture performance on image classification tasks. The results and discussion section proved that the project aim and objectives have been fulfilled.

5.6 Limitations and Future Work

5.6.1 Limitations of HyperDARTS

5.6.1.1 Limitations of Experiments Conducted

To determine the best configurations for hyperDARTS, several experiments (experiments 1.2.3, 1.3, 2.1.2, 2.1.3, and 2.2) were conducted to explore different values of learning rates, batch sizes, and the number of epochs. In these experiments, each value was tested only once with 1 run of architecture search. This limitation was mainly due to time constraints (2 semesters) and computational resources (1 GTX1080 Ti GPU assigned officially). Ideally, multiple runs of architecture search should be performed for each value to obtain a more accurate performance comparison based on the average performance, which better represents the normal or true case. This would ensure that only the best configurations are chosen.

5.6.1.2 Limitations as a DARTS-based Model

HyperDARTS was developed based on DARTS. In other words, most parts of hyperDARTS are same as DARTS, except the search strategy. Therefore, hyperDARTS can only provide incremental improvements, hooking to DARTS performance. However, even though DARTS is the pioneer and has a powerful influence in the field of differentiable architecture search (DAS), its performance for finding optimal architectures in terms of image recognition-based task performance and computational resources has been surpassed by many other later non-gradient-based NAS algorithms.

These non-gradient-based NAS algorithms include EA-based CARS-I [134], zero-shot based Zen-NAS [135], training-free TE-NAS [136], and semi-supervised SemiNAS [137]. All these methods achieve better test error rates while consuming similar or several times lower GPU days compared to DARTS. However, the performance superiority achieved by the non-gradient-based NAS algorithms (compared to DAS models) mentioned is outside the scope of this project and may not be tractable, at least in the near time.

Until today, there is not a single gradient-based NAS algorithm that managed to achieve over 80% in top-1 accuracy on ImageNet [135] while keeping the search cost below 4 GPU days. DARTS has a 73.1% top-1 accuracy on ImageNet, with the search cost being 4 GPU days. However, the best NAS algorithm for ImageNet classification up-to-date is Zen-NAS which attains a top-1 accuracy of 83.6% after searching for only 0.5 GPU days. Cai et al. [55] noted that weight sharing, the performance estimation strategy used by gradient-based NAS algorithms, incurs model interfering. They designed OFANet that employs a progressive-shrinking strategy to mitigate the issue, resulting in 80.1% of top-1 accuracy with the search cost being 51.6 GPU days.

In short, hyperDARTS performance might not be comparable to SOTA non-gradient-based NAS models. Nonetheless, comparing hyperDARTS performance with NAS models out of the field of DAS is out of the scope of this project as this project focuses only on the field of DAS. Therefore, these limitations are rather the limitations of the entire field of DAS.

5.6.2 Future Work

SOTA DAS models have failed to perform as well as other SOTA non-gradient-based NAS models. The main reason is the weight sharing mechanism commonly used in DAS models. DARTS and its variants also employ the weight sharing mechanism. Weight sharing was a star in the early days [25] as it was the first performance estimation strategy that managed to reduce search cost significantly. However, it was found that this mechanism incurs model interference issue [55].

In simple terms, the over-parameterized supernetwork contain all the candidate architectures, therefore the the weights of each candidate architecture are a subset of supernetwork weights. In other words, all the candidate architectures share weights. The candidate architectures cannot show optimal performance during cell searching as they do not have true optimal weights (which can be differed between candidate architectures). This causes the performance of candidate architectures to be estimated inaccurately, and thus inferior architecture would be chosen based on the inaccurate performance estimated.

Even though hyperDARTS still employs supernetwork, it does not use the weight sharing mechanism in a strict sense. Instead, it models the supernetwork weights as a function of hyperparameters that represent the architecture ($\boldsymbol{w} = \boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda})$). In other words, different candidate architectures result in different supernetwork weights. Therefore, **if** the hypernetworks are capable of modelling the weights locally with enough accuracy, hyperDARTS should have a less severe model inteference issue. Moreover, **if** the hypernetworks are capable of modelling the weights **globally** with enough accuracy, the model interference issue can be eliminated. In short, the model interference issue can be reduced and hyperDARTS performance can be improved by increasing hypernetwork capacity.

So far hyperDARTS has used simple hypernetworks: linear regression and 2-layer MLP with sigmoid. Replacing the hypernetworks with graph neural networks might improve hyperDARTS performance, since the hypernetwork output (a.k.a the supernetwork weights) has a graph nature. Nonetheless, it is unclear how far the hypernetwork capacity can be improved given limited memory resources and it is opened for future work.

6 Conclusion

6.1 Experiments and Results

This project demonstrated that, by replacing the search strategy used in DARTS from the one-step unrolling approximation scheme with hypernetworks, the resultant hyperDARTS managed to find better architectures than DARTS in terms of task performance (image recognition capability and architecture transferability) with a lower search cost. The image recognition capability is quantified by the testing accuracies achieved by the discovered architecture on CIFAR-10 and CIFAR-100 while the architecture transferability is quantified by the testing accuracy on CIFAR-100.

DARTS took 48 GPU hours to find the optimal architecture which achieved testing accuracies of 97.24% and 82.30% on CIFAR-10 and CIFAR-100 respectively. HyperDARTS-1 took 47 GPU hours to find the optmal architecture which achieved testing accuracies of 97.40% and 83.38% on CIFAR-10 and CIFAR-100 respectively. HyperDARTS-2 took 44 GPU hours to find the optmal architecture which achieved testing accuracies of 97.46% and 83.61% on CIFAR-10 and CIFAR-100 respectively. The improvements in image recognition capability and architecture transferability achieved by hyperDARTS-discovered architectures are definitely not trivial as they are comparable to other SOTA DARTS variants.

HyperDARTS performance validated the hypothesis that the hypergradient approximation scheme used in DARTS is inferior which reduces DARTS to find suboptimal architectures. A hypergradient approximation scheme is used in DARTS because the validation loss does not depend directly on hyperparameters, so the computationally expensive response hypergradient has to be approximated. HyperDARTS connects the validation loss and the hyperparameters ($\mathcal{L}_{val}(\boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}))$) by using hypernetworks ϕ (with hypernetwork weights bmw_{ϕ}) to model $\boldsymbol{w}$ locally. $\mathcal{L}_{val}$ still does not depend directly on $\boldsymbol{\lambda}$. However, they are now indirectly connected using hypernetworks, the response hypergradients can thus be computed inexpensively using backpropagation.

HyperDARTS fulfills the project aim and objectives by replacing the approximation scheme with hypernetworks. The simple hypernetworks used provided enough capacity for local $\boldsymbol{w}$ modelling that the resultant hyperDARTS-1 and hyperDARTS-2 are capable of finding better architectures than DARTS.

6.2 Future Work

As discussed in Section 5.6.2, hyperDARTS performance can be improved and the inherent limitations in a DARTS-based model can be reduced by increasing hypernetwork capacity. Replacing the simple hypernetworks used in this project with graph neural networks might improve hyperDARTS performance, since the hypernetwork output (a.k.a the supernetwork weights) has a graph nature. Nonetheless, it is unclear how far the hypernetwork capacity can be improved given limited memory resources and it is opened for future work.

7 Reflection on Project Management

7.1 Project Scope

This project's scope is to develop a new NAS model (hyperDARTS) by integrating one of the latest hyperparameter optimization methods (hypernetworks) into the search strategy of an existing DAS solution (DARTS).

The following is in-scope:

1. All coding of hyperDARTS.
2. Compare the performance of hyperDARTS with DARTS by testing on CIFAR-10 and CIFAR-100 datasets.

The following is out-of-scope:

1. Mathematical evaluation of hypernetworks.
2. Mathematical analysis of hyperDARTS.
3. Improving other parts of DARTS except search strategy.

7.2 Project Plan & Timeline

Tasks for FYP A in sequence:

1. Learn about NAS (to obtain a general understanding of the field).
2. Identify an existing highly influential gradient-based NAS solution (DARTS) as reference model.
3. Learn about DARTS.
4. Understand the code of DARTS.
5. Code DARTS out (since some parts are deprecated) and test it on its original experiments.
6. Learn about BLO.
7. Learn and understand the common solutions for BLO (i.e., implicit differentiation, unrolling, hypernetwork and hypernetworks).
8. Identify a latest gradient-based bilevel optimization solution (hypernetwork was chosen).
9. Code a simple BLO solution (e.g., unrolling) to test on a toy problem.
10. Understand hypernetworks thoroughly.
11. Understand the code of hypernetworks.
12. Code a linear hypernetwork to solve a toy problem.

Tasks for FYP B in sequence:

1. Integrate linear hypernetworks into DARTS in code, resulting in hyperDARTS-1.
2. Adjust the configuration of hyperDARTS-1 by conducting experiments on chosen datasets.
3. Finalize hyperDARTS-1 configuration.
4. Compare and analyze the performance of DARTS and hyperDARTS on chosen datasets.
5. Experiment with different hypernetworks to identify a more competitive type of hypernetworks, resulting in hyperDARTS-2.
6. Adjust the configuration of hyperDARTS-2 by conducting experiments on chosen datasets.
7. Finalize hyperDARTS-2 configuration.
8. Compare and analyze the performance of DARTS, hyperDARTS-1 and hyperDARTS-2 on chosen datasets.

Along with project briefing and documentation required, the tasks listed above can be categorized into six stages:

1. Project Initialization,
2. Requirement Analysis,
3. BLO Solution,
4. HyperDARTS-1,
5. HyperDARTS-2, and
6. Project Documentation.

Fig. 36 shows the Gantt Chart for the project timeline. The black line arrow indicates the critical path. In Fig. 36, within each phase, the corresponding tasks are listed in sequence, i.e., one must be finished before the next one can finish. The first five stages are as well in sequence, i.e., one stage must be finished before the next stage can finish, except stage 2 (Requirement Analysis) can be finished before stage 1 (Project Initialization). Therefore, all the tasks can be started later than planned only if they can still be finished on time. The critical path thus lies in the tasks which end any specific stage. It is because the delay of any of these tasks can incur the delay of ending the stage the task is in and all the subsequent stages. The Project Documentation stage is a special case that spans the whole FYP. Periods outside of teaching periods are colored in grey. During these periods, only tasks that need a continuous flow of work such as understanding code and testing of models are continued.

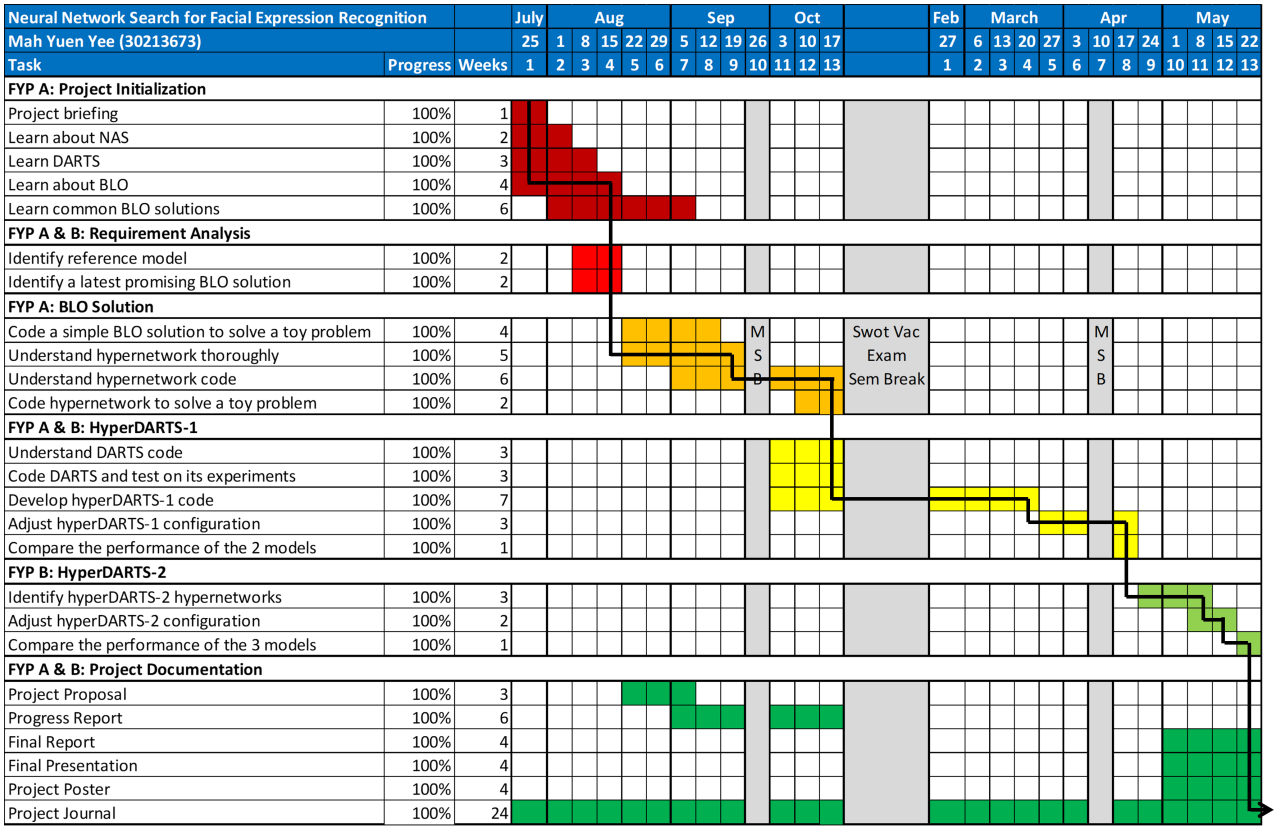


Fig. 36. The Gantt Chart for the project. MSB stands for Mid-Sememster Break.

7.3 Reflection on Project

Overall, the project scope remains the same. There are slight modifications in the project plan. Progress Report proposed to develop 2 new NAS models by integrating a type of hypernetworks into 2 reference models. However, the final project develops 2 new NAS models by integrating 2 types of hypernetworks into 1 reference model (DARTS). This is because there are just too many types and configurations of hypernetworks possible. It would be wasteful to stop at the first hypernetwork type and stop exploring how different hypernetworks perform in the DAS problem.

There are slight modifications in the project timeline as well. The development of hyperDARTS-1 code consumed more time and effort than expected, extending the task duration from 3 weeks to 7 weeks. Luckily, this task could be started earlier along with DARTS coding task. Therefore, the hyperDARTS-1 stage has only be extended by 1 week. Nonetheless, this causes the subsequent hyperDARTS-2 and project documentation stages to delay.

Fortunately, hyperDARTS-2 code can be developed based on hyperDARTS-1 code and, unexpectedly, require only minor modification. Therefore, the entire hyperDARTS-2 stage focuses more on experimenting different types of hypernetworks to identify the best hypernetwork type for hyperDRTS-2. As a result, the hyperDARTS-2 stage has managed to produce hyperDARTS-2 model with decent results. In fact, the time spent on experimenting with different types of hypernetworks for hyperDARTS-2 is same as the planned duration. However, the planned duration is shorter than it should be as the difficulty of identifying a more competitive hypernetwork type (than linear network used in hyperDARTS-1) is higher than expected. The design of each experiment requires more time and effort than expected to analyze the results from the previous experiments and to study how other researchers tackle the problems arised. Nonetheless, the most time-consuming part is the time taken for each experiment. A run of architecture search consumes around 12 GPU hours while a run of architecture evaluation consumes around 30 GPU hours. It was often difficult to start a new experiment before the previous one has finished since the design of the new experiment would rely on the results of the previous experiment.

Moreover, unexpectedly, it was very difficult to start project report, video and poster before finishing hyperDARTS-1 stage. In addition to the high workload from other units, the time taken for these 3 tasks have been reduced from the planned 7 weeks to only 4 weeks. This causes the resultant work quality to depreciate. From hindsight, the author should have finished developing hyperDARTS-1 code during semester break so that all the subsequent tasks can be completed with sufficient time.

8 References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory." *Neural computation*, vol. 9, no. 8, pp. 1735–80, Nov 1997.

[2] M. X. Chen, O. Firat, A. Bapna, M. Johnson, W. Macherey, G. F. Foster, L. Jones, N. Parmar, M. Schuster, Z. Chen, Y. Wu, and M. Hughes, "The best of both worlds: Combining recent advances in neural machine translation," *CoRR*, vol. abs/1804.09849, 2018. [Online]. Available: http://arxiv.org/abs/1804.09849

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, . Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016. [Online]. Available: https://arxiv.org/abs/1609.08144

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot MultiBox detector," in *Computer Vision – ECCV 2016*. Springer International Publishing, 2016, pp. 21–37. [Online]. Available: https://doi.org/10.1007%2F978-3-319-46448-0_2

[9] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020.

[10] R. Summers, "Nih chest x-ray dataset of 14 common thorax disease categories," 2019.

[11] Z. Huang, J. Lin, L. Xu, H. Wang, T. Bai, Y. Pang, and T.-H. Meen, "Fusion high-resolution network for diagnosing chestx-ray images," *Electronics*, vol. 9, no. 1, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/1/190

[12] Lion10, "Blood cell classifier." [Online]. Available: https://lion10.com/bloodcell

[13] A. Olafenwa, "Simplifying object segmentation with pixellib library," *Online.(2021). https://vixra. org/abs/2101.0122*, 2021.

[14] A. Dehghan, S. Z. Masood, G. Shu, and E. G. Ortiz, "View independent vehicle make, model and color recognition using convolutional neural network," 2017. [Online]. Available: https://arxiv.org/abs/1702.01721

[15] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, pp. 886–893.

[16] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, pp. 1150–1157 vol.2.

[17] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016. [Online]. Available: https://arxiv.org/abs/1611.01578

[18] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016. [Online]. Available: https://arxiv.org/abs/1611.02167

[19] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018. [Online]. Available: https://arxiv.org/abs/1802.03268

[20] C. Tang, Y. Zhao, G. Wang, C. Luo, W. Xie, and W. Zeng, "Sparse mlp for image recognition: Is self-attention really necessary?" 2021. [Online]. Available: https://arxiv.org/abs/2109.05422

[21] D. A. Tedjopurnomo, Z. Bao, B. Zheng, F. M. Choudhury, and A. K. Qin, "A survey on modern deep neural network for traffic prediction: Trends, methods and challenges," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 4, pp. 1544–1561, 2022.

[22] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018.

[23] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[24] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

[25] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=BJQRKzbA-

[26] 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," 2021.

[27] X. Cheng, Y. Zhong, M. Harandi, Y. Dai, X. Chang, T. Drummond, H. Li, and Z. Ge, "Hierarchical neural architecture search for deep stereo matching," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[28] M. Zhang, H. Li, S. Pan, X. Chang, C. Zhou, Z. Ge, and S. Su, "One-shot neural architecture search: Maximising diversity to overcome catastrophic forgetting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2921–2935, 2021.

[29] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 2902–2911.

[30] Y. Shu, W. Wang, and S. Cai, "Understanding architectures learnt by cell-based neural architecture search," 2019. [Online]. Available: https://arxiv.org/abs/1909.09569

[31] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2017. [Online]. Available: https://arxiv.org/abs/1705.10823

[32] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[33] T. Kerssies and J. Vanschoren, "Neural architecture search for visual anomaly segmentation," 2023.

[34] W. Zhu, Y. Huang, X. Xie, W. Liu, J. Deng, D. Zhang, Z. Wang, and J. Liu, "Autoshot: A short video dataset and state-of-the-art shot boundary detection," 2023.

[35] Y. Chen, T. Yang, X. Zhang, G. Meng, X. Xiao, and J. Sun, "Detnas: Backbone search for object detection," 2019. [Online]. Available: https://arxiv.org/abs/1903.10979

[36] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "NAS-FPN: Learning scalable feature pyramid architecture for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[37] J. Peng, M. Sun, Z. Zhang, T. Tan, and J. Yan, *Efficient Neural Architecture Transformation Search in Channel-Level for Object Detection.* Red Hook, NY, USA: Curran Associates Inc., 2019.

[38] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[39] X. Zhang, Q. Wang, J. Zhang, and Z. Zhong, "Adversarial autoaugment," 2019. [Online]. Available: https://arxiv.org/abs/1912.11188

[40] S. Cao, X. Wang, and K. M. Kitani, "Learnable embedding space for efficient neural architecture compression," 2019. [Online]. Available: https://arxiv.org/abs/1902.00383

[41] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1905.11946

[42] X. Dong and Y. Yang, *Network Pruning via Transformable Architecture Search.* Red Hook, NY, USA: Curran Associates Inc., 2019.

[43] S. Ding, T. Chen, X. Gong, W. Zha, and Z. Wang, "Autospeech: Neural architecture search for speaker recognition," 2020. [Online]. Available: https://arxiv.org/abs/2005.03215

[44] X. Gong, S. Chang, Y. Jiang, and Z. Wang, "AutoGAN: Neural architecture search for generative adversarial networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[45] J. Lorraine and D. Duvenaud, "Stochastic hyperparameter optimization through hypernetworks," 2018. [Online]. Available: https://arxiv.org/abs/1802.09419

[46] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," 2016.

[47] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda, and J. E. Gonzalez, "Fbnetv3: Joint architecture-recipe search using predictor pretraining," 2021.

[48] Y. Jiang, C. Hu, T. Xiao, C. Zhang, and J. Zhu, "Improved differentiable architecture search for language modeling and named entity recognition," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3585–3590. [Online]. Available: https://aclanthology.org/D19-1367

[49] 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 (ICCV)*, October 2019.

[50] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[51] R. Shin*, C. Packer*, and D. Song, "Differentiable neural network architecture search," 2018. [Online]. Available: https://openreview.net/forum?id=BJ-MRKkwG

[52] K. Ahmed and L. Torresani, "Maskconnect: Connectivity learning by gradient descent," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[53] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "Pc-darts: Partial channel connections for memory-efficient architecture search," *arXiv preprint arXiv:1907.05737*, 2019.

[54] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rylqooRqK7

[55] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," 2019. [Online]. Available: https://arxiv.org/abs/1908.09791

[56] 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 (CVPR)*, June 2018.

[57] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M. A. Osborne, "Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces," 2014. [Online]. Available: https://arxiv.org/abs/1409.4011

[58] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," vol. 20, no. 1, p. 1997–2017, jan 2019.

[59] F. L. Bauer, "Computational graphs and rounding error," *SIAM Journal on Numerical Analysis*, vol. 11, no. 1, pp. 87–96, 1974. [Online]. Available: http://www.jstor.org/stable/2156433

[60] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," 2016. [Online]. Available: https://arxiv.org/abs/1610.02357

[61] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015. [Online]. Available: https://arxiv.org/abs/1511.07122

[62] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 497–504. [Online]. Available: https://doi.org/10.1145/3071178.3071229

[63] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1711.04528

[64] 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 (CVPR)*, June 2016.

[65] L. Xie and A. Yuille, "Genetic CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[66] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.

[67] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[68] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[69] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=H1loF2NFwr

[70] A. Yang, P. M. Esperança, and F. M. Carlucci, "Nas evaluation is frustratingly hard," 2019. [Online]. Available: https://arxiv.org/abs/1912.12522

[71] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," ser. Foundations of Genetic Algorithms, G. J. RAWLINS, Ed. Elsevier, 1991, vol. 1, pp. 69–93. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080506845500082

[72] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," 2018. [Online]. Available: https://arxiv.org/abs/1804.09081

[73] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Aging evolution for image classifier architecture search," in *AAAI conference on artificial intelligence*, vol. 2, 2019, p. 2.

[74] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 2020–2029.

[75] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[76] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, ser. LION'05. Berlin, Heidelberg: Springer-Verlag, 2011, p. 507–523. [Online]. Available: https://doi.org/10.1007/978-3-642-25566-3_40

[77] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 1. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 115–123. [Online]. Available: https://proceedings.mlr.press/v28/bergstra13.html

[78] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)*, Q. Yang and M. Wooldridge, Eds., 2015, pp. 3460–3468, 1st International Workshop on Social Influence Analysis / 24th International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, ARGENTINA, JUL 25-31, 2015.

[79] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, M. Urban, M. Burkart, M. Dippel, M. Lindauer, and F. Hutter, *Towards Automatically-Tuned Deep Neural Networks*. Cham: Springer International Publishing, 2019, pp. 135–149. [Online]. Available: https://doi.org/10.1007/978-3-030-05318-5_7

[80] P. Ye, B. Li, Y. Li, T. Chen, J. Fan, and W. Ouyang, "β-darts: Beta-decay regularization for differentiable architecture search," 2022.

[81] X. Chen and C.-J. Hsieh, "Stabilizing differentiable architecture search via perturbation-based regularization," 2021.

[82] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Chapter 15 - evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, R. Kozma, C. Alippi, Y. Choe, and F. C. Morabito, Eds. Academic Press, 2019, pp. 293–312. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128154809000153

[83] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," *CoRR*, vol. abs/1807.06906, 2018. [Online]. Available: http://arxiv.org/abs/1807.06906

[84] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, ser. Proceedings of Machine Learning Research, vol. 54. PMLR, Apr. 2017, pp. 528–536. [Online]. Available: http://proceedings.mlr.press/v54/klein17a.html

[85] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the CIFAR datasets," *CoRR*, vol. abs/1707.08819, 2017. [Online]. Available: http://arxiv.org/abs/1707.08819

[86] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 6765–6816, jan 2017.

[87] 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 *COMPUTER VISION - ECCV 2018, PT I*, ser. Lecture Notes in Computer Science, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., vol. 11205, 2018, pp. 19–35, 15th European Conference on Computer Vision (ECCV), Munich, GERMANY, SEP 08-14, 2018.

[88] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/933670f1ac8ba969f32989c312faba75-Paper.pdf

[89] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research,

M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 564–572. [Online]. Available: https://proceedings.mlr.press/v48/wei16.html

[90] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, "Block-wisely supervised neural architecture search with knowledge distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[91] M. Zhang, H. Li, S. Pan, X. Chang, and S. Su, "Overcoming multi-model forgetting in one-shot nas with diversity maximization," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7806–7815.

[92] F. Runge, D. Stoll, S. Falkner, and F. Hutter, "Learning to design RNA," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ByfyHh05tQ

[93] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 550–559. [Online]. Available: https://proceedings.mlr.press/v80/bender18a.html

[94] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1946–1956. [Online]. Available: https://doi.org/10.1145/3292500.3330648

[95] H. Von Stackelberg, *Market structure and equilibrium*. Springer Science & Business Media, 2010.

[96] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, no. 1, pp. 235–256, 2007.

[97] P. Hansen, B. Jaumard, and G. Savard, "New branch-and-bound rules for linear bilevel programming," *SIAM J. Sci. Stat. Comput.*, vol. 13, no. 5, p. 1194–1217, sep 1992. [Online]. Available: https://doi.org/10.1137/0913069

[98] D. Duvenaud, "Csc 2541: Neural net training dynamics (lecture 11 - bilevel optimization)," 2022.

[99] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International conference on machine learning*. PMLR, 2015, pp. 2113–2122.

[100] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1165–1173.

[101] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots, "Truncated back-propagation for bilevel optimization," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1723–1732.

[102] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, S. Chiappa and R. Calandra, Eds., vol. 108. PMLR, 26–28 Aug 2020, pp. 1540–1552. [Online]. Available: https://proceedings.mlr.press/v108/lorraine20a.html

[103] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *International conference on machine learning*. PMLR, 2016, pp. 737–746.

[104] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," 2020.

[105] M. MacKay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse, "Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions," 2019. [Online]. Available: https://arxiv.org/abs/1903.03088

[106] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: One-shot model architecture search through hypernetworks," 2017. [Online]. Available: https://arxiv.org/abs/1708.05344

[107] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[108] S. Mahapatra, "Why deep learning over traditional machine learning?" Jan 2019. [Online]. Available: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063

[109] S. Saha, "A comprehensive guide to convolutional neural networks - the eli5 way," May 2023. [Online]. Available: https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/

[110] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library."

[111] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ICT Express*, vol. 6, no. 4, pp. 312–315, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959519303455

[112] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 2012.

[113] D. Chang and A. Pathak, "Effect of batch size on neural net training," Aug 2020. [Online]. Available: https://medium.com/deep-learning-experiments/effect-of-batch-size-on-neural-net-training-c5ae8516e57

[114] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," 2020.

[115] P. Madhyastha and R. Jain, "On model stability as a function of random seed," 2019.

[116] H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, and Z. Li, "Darts+: Improved differentiable architecture search with early stopping," 2020.

[117] K. Bi, C. Hu, L. Xie, X. Chen, L. Wei, and Q. Tian, "Stabilizing darts with amended gradient estimation on architectural parameters," 2020.

[118] P. Zhou, C. Xiong, R. Socher, and S. C. H. Hoi, "Theory-inspired path-regularized differential network architecture search," 2020.

[119] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022022, feb 2019. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1168/2/022022

[120] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[121] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio, "Architectural complexity measures of recurrent neural networks," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2016/file/860320be12a1c050cd7731794e231bd3-Paper.pdf

[122] B. Dherin, M. Munn, M. Rosca, and D. Barrett, "Why neural networks find simple solutions: The many regularizers of geometric complexity," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 2333–2349. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/0ff3502bb29570b219967278db150a50-Paper-Conference.pdf

[123] T. Liang, T. Poggio, A. Rakhlin, and J. Stokes, "Fisher-rao metric, geometry, and complexity of neural networks," in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and M. Sugiyama, Eds., vol. 89. PMLR, 16–18 Apr 2019, pp. 888–896. [Online]. Available: https://proceedings.mlr.press/v89/liang19a.html

[124] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

[125] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the vapnik-chervonenkis dimension," *J. ACM*, vol. 36, no. 4, p. 929–965, oct 1989. [Online]. Available: https://doi.org/10.1145/76359.76371

[126] Y. Li, S. Gu, K. Zhang, L. Van Gool, and R. Timofte, "Dhp: Differentiable meta pruning via hypernetworks," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 608–624.

[127] A. Navon, A. Shamsian, G. Chechik, and E. Fetaya, "Learning the pareto front with hypernetworks," 2021.

[128] X. Hu, L. Chu, J. Pei, W. Liu, and J. Bian, "Model complexity of deep learning: A survey," 2021.

[129] J. Ma and S. Fattahi, "Blessing of depth in linear regression: Deeper models have flatter landscape around the true solution," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 34 334–34 346. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/de04896f011beff76c91e094f72727f4-Paper-Conference.pdf

[130] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive darts: Bridging the optimization gap for nas in the wild," 2020.

[131] Y.-C. Gu, L.-J. Wang, Y. Liu, Y. Yang, Y.-H. Wu, S.-P. Lu, and M.-M. Cheng, "Dots: Decoupling operation and topology in differentiable architecture search," 2021.

[132] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, "Rethinking architecture selection in differentiable nas," 2021.

[133] X. Chu, X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan, "Darts-: Robustly stepping out of performance collapse without indicators," 2021.

[134] Z. Yang, Y. Wang, X. Chen, B. Shi, C. Xu, C. Xu, Q. Tian, and C. Xu, "Cars: Continuous evolution for efficient neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1829–1838.

[135] M. Lin, P. Wang, Z. Sun, H. Chen, X. Sun, Q. Qian, H. Li, and R. Jin, "Zen-nas: A zero-shot nas for high-performance image recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 347–356.

[136] W. Chen, X. Gong, and Z. Wang, "Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective," *arXiv preprint arXiv:2102.11535*, 2021.

[137] R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, and T.-Y. Liu, "Semi-supervised neural architecture search," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 547–10 557, 2020.

[138] G. H. Brundtland, "Our common future—call for action," *Environmental Conservation*, vol. 14, no. 4, pp. 291–294, 1987.

[139] T. Seager, E. Selinger, and A. Wiek, "Sustainable engineering science for resolving wicked problems," *Journal of agricultural and environmental ethics*, vol. 25, no. 4, pp. 467–484, 2012.

[140] S. Connelly, "Mapping sustainable development as a contested concept," *Local environment*, vol. 12, no. 3, pp. 259–278, 2007.

[141] E. Woodhouse and D. Sarewitz, "Science policies for reducing societal inequities," *Science and Public Policy*, vol. 34, no. 2, pp. 139–150, 2007.

[142] A. Borgmann, *Technology and the character of contemporary life: A philosophical inquiry*. University of Chicago Press, 1987.

[143] R. U. Ayres, "The price-value paradox," *Ecological Economics*, vol. 25, no. 1, pp. 17–19, 1998.

[144] F. Albertao, J. Xiao, C. Tian, Y. Lu, K. Q. Zhang, and C. Liu, "Measuring the sustainability performance of software projects," in *2010 IEEE 7th International Conference on E-Business Engineering*. IEEE, 2010, pp. 369–373.

9 Appendices

9.1 Appendix A: Project Risk Assessment

The approved OHS Risk Assessment for the project from SARAH is attached below:

42073	RISK DESCRIPTION	TREND	CURRENT	RESIDUAL
	ENG4701_MA_2022_S2_YuenYeeMah_NeuralNetworkSearchForFacialExpressionRecognition		Medium	Medium

RISK TYPE
1. Activity or Task Based Risk Assessment

RISK OWNER	RISK IDENTIFIED ON	LAST REVIEWED ON	NEXT SCHEDULED REVIEW
YUEN YEE MAH	28/08/2022	28/08/2022	28/08/2025

RISK FACTOR(S)	EXISTING CONTROL(S)	CURRENT	PROPOSED CONTROL(S)	TREATMENT OWNER	DUE DATE	RESIDUAL
Carry laptop which stores all relevant digital documents, and all paper notes back and forth between the university and my hostel to work on the project. The need of going to the on-campus laboratory is to access the computer assigned inside the laboratory to run and test code that demands high computing resources. By frequently carrying the laptop and the paper notes that constitute more than 5kg of weights using backpack, musculoskeletal stress and discomforts such as back and shoulder pain and discomfort can be incurred.	Control: Use AnyDesk to run and test code on the on-campus computer remotely from my hostel. Control Effectiveness:	Low				Low
Continuous laptop usage can lead to ergonomics issues or even cumulative trauma disorder (CTD). Fixed posture, repetitive typing and high mouse usage can lead to musculoskeletal and nervous system injuries. Starting at the laptop screen for a long time can lead to excessive straining and deterioration of eye muscles.	Control: Refer to scientific ergonomics suggestions and recommendations to setup workstation, such as the proper positioning of the laptop and mouse, usage of external keyboard, usage of leg rests, and adequate chair height. Control Effectiveness: _____ Control: Employ ergonomics software or refer to scientific ergonomic recommendations to take regular breaks for body	Medium				Medium

	and eye relaxation. Control Effectiveness: _____ Control: Print out journal papers for reading to reduce time staring at the laptop screen. Control Effectiveness:					
The laboratory has a very low temperature to resolve the heating issue of the computers since they almost always keep running without powering off. Working in the laboratory without enough clothing can lead to cold stress to the body.	Control: Use AnyDesk to run and test code on the on-campus computer remotely from my hostel. Control Effectiveness: _____ Control: Be dressed warm enough when inside the lab. Control Effectiveness:	Low				Low
Risk of infecting COVID-19 when working inside the on-campus laboratory.	Control: Use AnyDesk to run and test code on the on-campus computer remotely from my hostel. Control Effectiveness:	Low				Low
My laptop is susceptible to malfunction due to over usage for university work and unsafe environment inside my room such as potential drinking water spilling. Malfunctioning laptop can lead to data lost and significantly hinder the project progress. This can lead to psychological stress and distress.	Control: Backup all relevant data to cloud drive consistently. Control Effectiveness: _____ Control: Distance items that can potentially damage the laptop from my laptop when working inside my room. Control Effectiveness:	Medium				Medium
High workload due to the project and other university work can lead to psychological stress and distress. This can cause serious mental issues that need professional consultations, and significantly hinder the project progress.	Control: Employ time and project management skills effectively to ensure workload is distributed evenly and reasonably throughout the semester. Control Effectiveness: _____ Control: Seek professional help when facing any mental issues. Control Effectiveness:	Medium				Medium

9.2 Appendix B: Risk Management Plan

Risks can be categorized as Occupational Health and Safety (OHS) and non-OHS. The OHS risk assessment has been done in Safety and Risk Analysis Hub (SARAH) and approved by the project supervisor Dr. Mohamed Hisham Jaward. The non-OHS risk assessment is presented in Table 24.

Table 24: Risk assessment table for non-OHS project risks.

Project Risk	Risk	Likelihood	Consequence	Risk level	Mitigation	Residual Risk
Delayed completion of project	Deadlines may not be met due to slower progress than expected.	Possible	Catastrophic	H	Strictly follow the timeline designed and reduce the perfectionist tendency. Spend more time per day.	Tasks defined are too demanding or time-consuming to complete on time.
Experiment results	Hypothesis may not be supported by experiment results.	Possible	Serious	M	Discuss with the supervisor to identify alternative hypothesis or solution.	Follow an alternative approach may delay project delivery.
COVID-19 infection	Health (bodily and mental) conditions can be negatively affected by an infection resulting in low productivity.	Possible	Serious	M	Wear a mask in public areas and practice social distancing.	Chances of infection are reduced but not eliminated.
Work-life balance disrupted	Health (bodily and mental) conditions can be negatively affected by inadequate rest.	Almost Certain	Disastrous	E	Strictly follow the timeline designed and reduce the perfectionist tendency. Allocate time to rest.	Work-life balance may still be affected by high workload of other units.

9.3 Appendix C: Sustainability Plan

9.3.1 Sustainability Engineering

9.3.1.1 Sustainability

The United Nations Brundtland Commission defines sustainability as "meeting the needs of the present without compromising the ability of future generations to meet their own needs" [138]. Seager et al. [139] argue that the interpretation is confined to a narrow economic sense by defining the sustainability problem as a matter of natural resource stewardship and income distribution equity. They further propose that sustainability problem should incorporate environmental, economic, and social considerations in the context of complex global systems involving tangling cause-and-effect relationships, contrasting value judgements or cultural norms, and lack of streamlined collaboration methods between disciplines where necessary knowledge resides in. In other words, sustainability should be interpreted as an essentially contested concept [140] which can only be tackled by solutions with anticipation, adaptation, and resilience [139]. As technological innovation is crucial in enabling sustainability, engineering science and technology research and education should adapt to involve sustainability engineering to advance sustainability. There are 3 approaches to sustainability engineering discussed by Seager et al.: business-as-usual, systems engineering, and sustainable engineering science.

9.3.1.2 Business-As-Usual

As the most popular strategy, business-as-usual assumes that the introduction of new capabilities will inevitably improve environmental, social, and economic quality and is positive about technological advancement. In other words, this strategy suggests that science be conducted as normal, which involves intensive research in a single sub-discipline considering only professional ethics while ignoring more general contextual issues like the scale and efficiency of the resulting sustainable engineering solutions. This strategy is criticised for resulting in technological advancements that disproportionately benefit the wealthy [141] and reinforcing consumerism through the device paradigm, that is, by luring people into passivity and addictions to technological artefacts that separate means from ends and rob meaning [142].

9.3.1.3 Systems Engineering

Systems engineering includes two perspectives: engineering with sustainability constraints and with the triple bottom line of sustainability. The former optimizes engineering systems for conventional objectives like cost reduction or return rate maximisation under stringent constraints of environmental emissions standards and a growing stakeholder and public interest and involvement. The latter broadens the design objectives to include the sustainability triple bottom line: economy, environment, and society. Environmental and social goals are included in design objectives for the evaluation of trade-offs with other solutions, not just as constraints to be subject to. The two perspectives presented here, however, continue to disregard issues of scale: cost-effective manufacturing of commodities leads to unsustainable demand growth while neglecting the associated external costs, negative prices, and marginalisation of public goods [143].

9.3.1.4 Sustainable Engineering Science

Sustainable engineering science is aware that current scientific paradigms are prone to myopia and have left behind a legacy of challenging social, environmental, and economic issues. As a result, this strategy aims to fundamentally alter the viewpoint and scientific methodology that gave rise to complicated problems like hazardous waste or climate change. Three key differences separate sustainable engineering science from other approaches [139]: (1) it is based on the recognition that the macro-ethical requirements of technology development go beyond just research or professional ethics; (2) it moves away from risk-based systems optimization and toward anticipatory and systems resilience perspectives; and (3) it moves forward with an awareness of the need to intentionally develop the interactional expertise required to conduct integrative, cross-disciplinary scientific research.

9.3.1.5 Summary

The author agrees with the practice of sustainable engineering science the most. The first two approaches have intrinsic shortcomings such as contradicting interests between myopic design objectives and sustainability, ignoring sustainability issues brought on by scale and efficiency, and a lack of interactional expertise. In order to address sustainability as a wicked problem, sustainable engineering science should be used for its paradigm shift in problem framing and constrained solutions. The problem formulation shifts from a narrow definition to a continuous responsive cycle of anticipation and adaptation as fresh knowledge concerning feedback effects and unintended consequences is learned. The solutions offered by sustainable engineering science involve interactional expertise considering the intricate interactions between the resulting engineered system and the adaptive natural and social systems.

9.3.2 Assessment and Analysis of Sustainability Performance of the Project

The project's sustainability performance is assessed following the metrics proposed by [144]: quality attributes (development-, usage-, and process-related attributes) and economic, social, and environmental benefits.

9.3.2.1 Quality Attributes

The code for the NAS network developed in this project should have high modifiability and reusability by writing the network as a class in Python from which an instance can be easily created using a line of code. The class should include comprehensive input arguments in the calling method so the network can be customized for usage without modifying the code inside the class. All methods inside the class should be modularized according to their functionalities to improve the modifiability of the network code. The code should be documented with clear and concise comments so that it can be reused without understanding difficulties. For network training and evaluation, a graphics processing unit (GPU) and methods that improve hardware efficiencies, such as parallel computing and optimized libraries (Pytorch), should be used to improve the training and evaluation efficiency.

9.3.2.2 Economic, Social, and Environmental Benefits

Modifiability minimizes development and support costs in terms of time taken and energy consumed. Reusability minimizes environmental impact through less effort in improving the network in the future if needed. With detailed documentations, the code can be accessed and understood by a larger audience to reduce the learning costs and provide more equal opportunities in deep learning research and applications. Efficiency minimizes the effort waste with an optimized use of environmental resources such as electrical energy. As an attempt to improve the image recognition capability of deep learning in terms of accuracy performance and environmental resources, this project helps to make applications of image classification more accessible. Image classification can be used for security applications, e.g., as a part of a surveillance system to ensure security to advance social welfare. Moreover, healthcare industries can employ image classification for practitioners to diagnose diseases more accurately [10, 11], improving the efficiency and effects of the healthcare system. In addition, image classification can be used to aid the process of automated inspection and quality control in the manufacturing industry to reduce the repetitive work needed, improving the availability of psycho-social-spiritual sustenance [142].

9.4 Appendix D: Generative AI Statement

MONASH University

Yuen Mah <ymah0006@student.monash.edu>

Generative AI use in FYP B (ENG4702)

Google Forms <forms-receipts-noreply@google.com>
To: ymah0006@student.monash.edu

26 May 2023 at 18:11

Thanks for filling in Generative AI use in FYP B (ENG4702)

Here's what was received.

Edit response

Generative AI use in FYP B (ENG4702)

The responses to this form will need to be copied and put into an appendix in your Final Report.

Email *

ymah0006@student.monash.edu

Name *

Mah Yuen Yee

Campus

- ○ Clayton
- ◉ Malaysia

Host Department

- [ ] Chemical and Biological Engineering
- [ ] Civil Engineering
- [x] Electrical and Computer Systems Engineeing
- [ ] Materials Science Engineering
- [ ] Mechanical and Aerospace Engineering
- [ ] Software Engineering
- [ ] Robotics and Mechatronics Engineering

Supervisor

Dr Liang Shiuan-Ni

This project has been conducted using AI tools *

- [ ] In this assessment, there will be no use of generative artificial intelligence (AI). All content in relation to the assessment task has been produced by the authors.
- [x] In this assessment, the following generative AI will be used for the purposes nominated in part 2. (Please note: any use of generative AI must be appropriately acknowledged - see Learn HQ)
- [ ] In this assessment, AI writing assistants (e.g., Grammarly, Writesonic, Quillbot, Microsoft Editor) will be the only form of Generative AI used.
- [ ] This project involves the development or authoring of Unique Generative AI, Unique operation of commercially available Generative AI OR Unique non-generative AI (Machine Learning, Artificial Neural Network, Logistic Regression, etc.)

AI Tools used

In question 1, you answered you were using generative AI for the purposes nominated in part 2. Insert names of AI tools, or types of tools (e.g. image generators/text generators) you used.
(Please note: any use of generative AI must be appropriately acknowledged - see Learn HQ)

ChatGPT-3.5, Grammarly

How has the technology be used?

How did you use this technology?

*

- [ ] As a fundamental aspect of the study (i.e., this is a study centred on generative AI, human interaction, associated ethics, etc.)
- [ ] Audio Transcription
- [ ] Coding/Scripting
- [ ] For the operation of robotics
- [ ] Generation of novel content - Datasets
- [ ] Generation of novel content - Graphics/Images
- [ ] Generation of novel content - Video
- [ ] Generation of novel content - Writing
- [ ] Idea generation
- [ ] Initial research
- [ ] Machine Language Translation
- [ ] Mathematics
- [x] Paraphrasing
- [ ] Proofreading
- [ ] Text Analytics
- [ ] Text Summarisation
- [ ] Thematic analysis
- [ ] Visualisation (of data)
- [x] Writing assistance

Other:

How was the Generative AI response validated?

Generative AI is used for paraphrasing and writing assistance in this project. Generative AI has not been used for any other purposes in this project. Therefore, the output messages should bear the same meaning as the input messages with improved grammar. The output message is read and modified by the author to ensure the meaning conveyed is the same as the input message.

Permissions

The use of Generative AI has been discussed with and approved by my academic supervisor. *

- (●) Yes
- ( ) No

End

Thank you for completing this form - your responses will be emailed to you for your Progress Report

9.5 Appendix E: Common Terms

Table 25: Common terms used in this report and their meanings.

Term	Meaning
$\boldsymbol{\lambda}$	Hyperparameters/ Operation weightages/ Architecture parameters
$\boldsymbol{w}$	Supernet weights = weights of all candidate architectures
ϕ	Hypernetwork
$\boldsymbol{w}_{\phi}$	Hypernetwork weights
Hypergradient	The gradient of L_{val} with respect to $\boldsymbol{\lambda}$.