MONASH University
Engineering

Neural Network Search for Facial Expression Recognition

ENG4701: Final Year Project - Progress Report

Author: Yuen Yee Mah (30213673)
Supervisor: Dr. Mohamed Hisham Jaward
Date of Submission: 21 Oct. 2022
Project type: Research

Contents

1 Introduction

Deep learning has performed exceptionally well 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 mainly attributed to the strong learning capabilities of deep learning methods, especially their capabilities of automatic feature extraction from unstructured data. Deep learning has shifted feature engineering for unstructured data from laborious manual work [10, 11] into automatic extraction [4, 12, 13], allowing researchers to have their manual work focus more on designing and engineering network components and neural architectures [14–16]. An example of the architecture of a modern deep learning neural network is shown in Fig. 2.

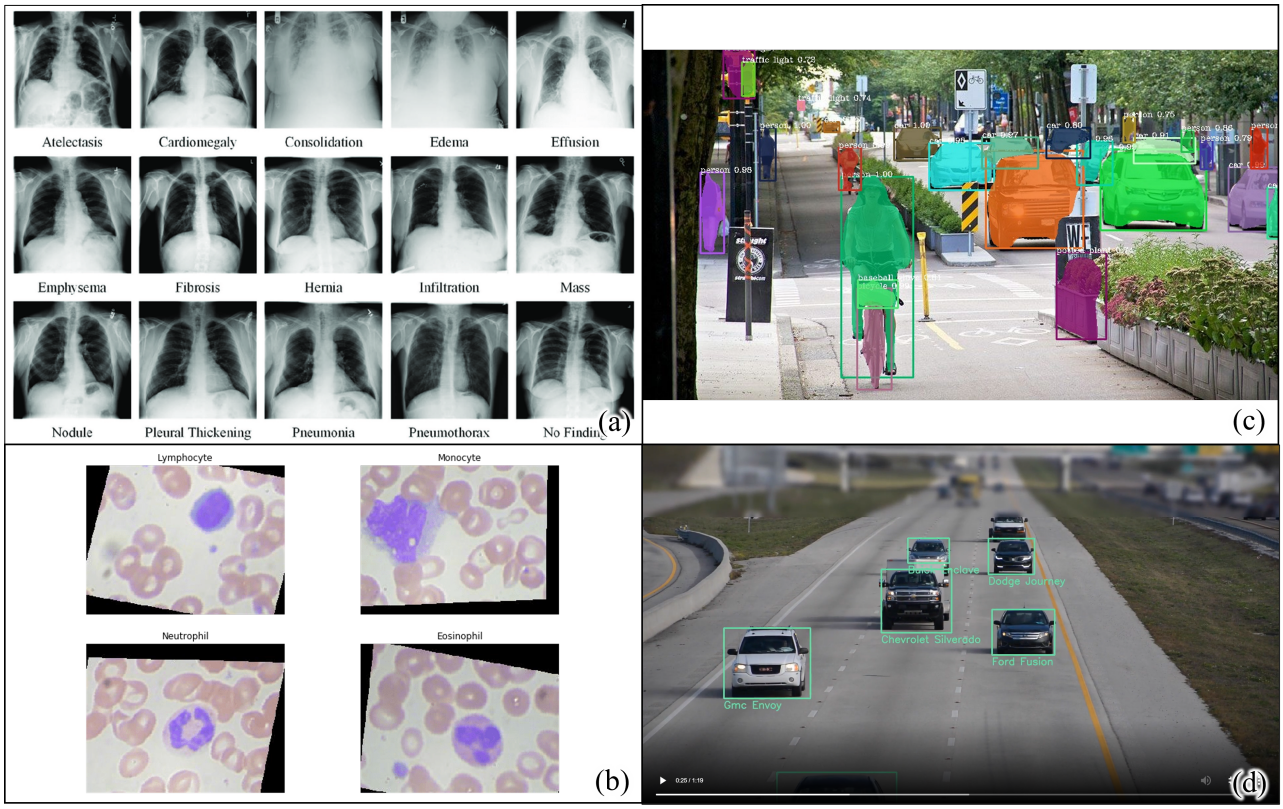
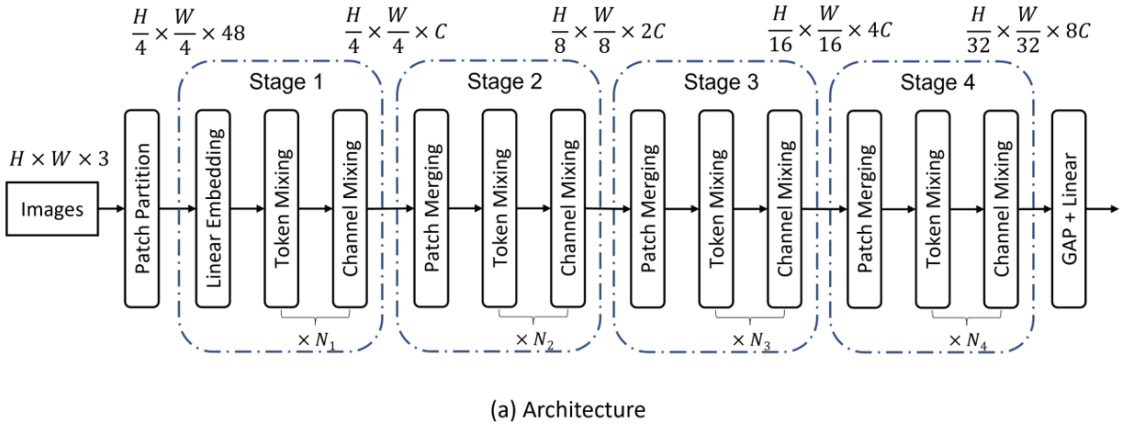


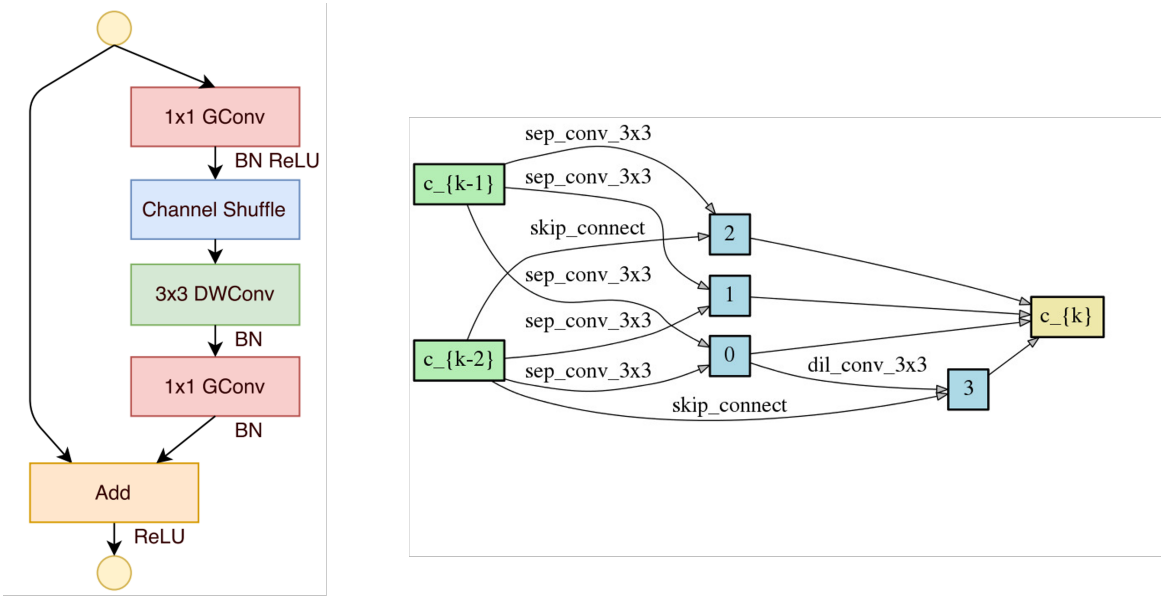
Fig. 1. Some examples of deep learning application. Image recognition for (a) thorax disease diagnosis [17, 18] and (b) blood cell classification [19]. Object detection for (c) instance segmentation of videos [20] and (d) vehicle detection and recognition [21].

While effective architecture alterations do improve the performance of deep learning methods, manual searching for enhanced architectures is itself an arduous, time-consuming, and error-prone task. In addition, manually designing neural architecture is not a beginner-friendly task, which relies heavily on the researchers' prior knowledge, expertise, and experience. As a result, it is difficult for newcomers or practitioners to amend emerging neural architectures per their actual needs [23]. Besides, heavy reliance on experts' prior knowledge can introduce human bias and fixed thinking paradigms to architecture design, partly limiting the discovery of novel architectures [24] as illustrated in Fig. 3. 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. The limitations of manual architecture search led to an insurgence in research efforts in recent years seeking to automate this search process. As a result, research on the problem of automatic search for an optimal architecture known as **Neural Architecture Search** (**NAS**) has grown in importance.



(a) Architecture

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 [22], 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. The operational details of the architecture are not important here.



(a) (b)

Fig. 3. (a) The micro-architecture of a cell type used in ShuffleNet [25], which was manually designed. (b) The micro-architecture of a cell type used in DARTS [26] 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 be subject to less fixed thinking paradigms and more creative. The operations used in the cells shown are not important here.

NAS mainly 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 [14] and MetaQNN [15]. 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 speed up the neural architecture search process [16, 30, 31]. Notably, Cai et al. [32] and Pham et al. [16] have managed to do so by leveraging the principle of reuse of learned model parameters. As search efficiency has improved, NAS methods have also been extensively applied in various fields. NAS has outperformed manually designed architectures on some tasks such as image classification [4–6], object detection [33–35], and semantic segmentation [36–38]. In other fields, such as data augmentation [39, 40], architectural scaling [41–43], speech recognition [44], and adversarial learning [45], NAS has achieved highly competitive performance comparable to state-of-the-art handcrafted architectures.

The automation of NAS is essentially a search problem over a set of decisions that define all different components of a neural network. The search space is defined implicitly by the set of feasible solutions to these decisions, while the optimizer defines the search algorithm.

This project focuses on further improving efficient NAS performance by integrating the latest algorithm developed for optimizing hyperparameters such as **Self-Tuning Network** (**STN**) [46] on image classification or recognition, especially **Facial Expression Recognition** (**FER**). Image classification field is focused on since it provides a challenging benchmark due to a great deal of manual architecture engineering devoted to this field [4–6]. On the other hand, as NAS still incorporates humans' prior knowledge on some parts of its search problem, especially the search space, it is relatively simple to define a suitable search space by utilizing knowledge from the vast source of manual engineering. Nevertheless, this, in turn, makes it less probable that NAS will discover architectures that perform significantly better in terms of test error rate than those existing since the discovered architectures cannot differ fundamentally without including foundational novelties such as new building blocks.

As most high-performing handcrafted architectures are designed based on general image classification datasets such as CIFAR-10 and ImageNet, they are transplanted for application on other datasets for other image classification tasks [47]. However, even though the transferred architectures demonstrate competitive results, the amount of inadaptability loss is unknown. Therefore, the less explored FER field is a focus of this project instead of only the general image classification datasets.

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 [14, 15]. Differentiable architecture search (DAS) [26] is the first search strategy that manages to reduce the search cost by three orders of magnitudes to a few GPU days while maintaining high test error performance by sharing weights between child architectures when searching (discussed in Section 3.3.4). DARTS [26] was developed as the first DAS-based model to search for the entire architecture instead of only some specific parts of the architecture [48, 49]. Thereafter, the field of DAS has been proliferating and is now one of the reliable NAS methods to achieve state-of-the-art model performance. Nevertheless, most recent DASs still use DARTS as the preliminary or foundational model [50–53], proving the effectiveness and fundamental importance of DARTS in the field of DAS. Therefore, this project of developing a new DAS-based model follows the prevailing and promising trend of using DARTS as the preliminary model.

DARTS [26] 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, but DARTS managed to achieve competitive test error performance with only four GPU days as, in practice, the optimization algorithm is able to reach a fixed point with a carefully chosen hyperparameter involved in the optimization algorithm (not hyperparameters related to building architecture) as discussed in Section 6.1.1.

2.1.2 Hypothesis

It is hypothesized that even if DARTS is able to accomplish competitive performance in terms of test error and computational resources, its greatly abbreviated optimization algorithm reduces itself to only converge to suboptimal hyperparameters during cell search. In other words, there is room for performance improvement in DARTS regarding test error and computational resources just by replacing the one-step unrolling with an inherently more efficient and stable hyperparameter optimization method.

2.2 Aims

This project aims to develop a new NAS network based on cutting-edge DAS-based networks by integrating the latest hyperparameter optimization algorithm, such as Self-Tuning Network (STN), into search strategy to improve DAS and possibly NAS performance on image classification tasks, especially for Facial Expression Recognition (FER).

2.3 Objectives

1. To understand, analyze, and evaluate different kinds of NAS networks by acquiring general and comprehensive knowledge all about NAS to construct a strong background and knowledge base.
2. To deeply understand, analyze, and evaluate a specialized field of NAS networks and its components by acquiring deep knowledge about the current state of theories, techniques, approaches, and models in the specialized field where this project focuses on: DAS-based networks which are NAS networks with modular search space, gradient-based search strategy tackling a bilevel optimization problem, and gradient descent-based performance estimation strategy.
3. To identify some latest promising and applicable gradient descent-based bilevel optimization solutions by the end of Final Year Project (FYP) A.
4. To apply and integrate the most promising optimization solution into the search strategy of a cutting-edge DAS-based NAS network.
5. To evaluate the newly developed DAS-based NAS network performance on image recognition tasks, especially for FER, against the best performance achieved by others [54].
6. To improve on the newly developed DAS-based NAS network if its performance is not desired by revamping the optimization solution or utilizing other optimization solutions.

3 Literature Review

NAS design can be decomposed into three main dimensions: search space, search strategy, and performance estimation strategy, as illustrated in Fig. 4.

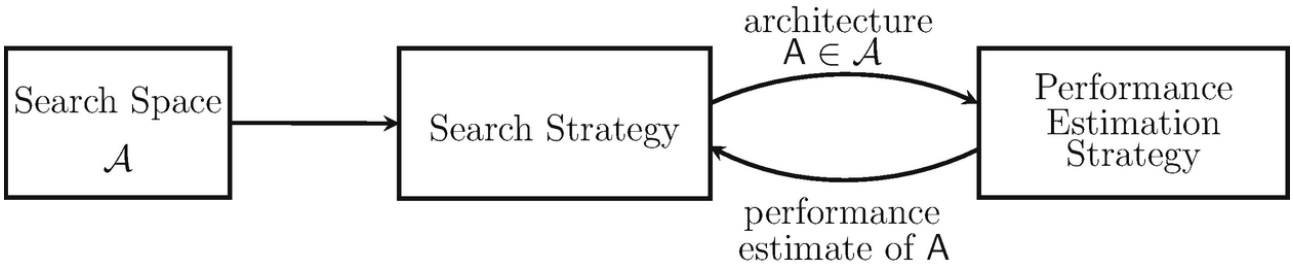


Fig. 4. An abstract illustration of NAS methods [55]. A search strategy chooses an architecture A from a predefined search space $\mathcal{A}$. A is then passed to a performance estimation strategy, which outputs the estimated performance of A to the search strategy.

- **Search Space.** The search space, which defines all possible discoverable architectures, is determined by the predefined operation set and constraints imposed on the architectures. 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.
- **Performance Estimation Strategy.** The goal of NAS is mainly to discover architectures that show excellent predictive performance on unseen data. Performance Estimation is the process of estimating this performance. Intuitively, the most direct approach is to perform standard training and validation of the architecture 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.

Almost all enormous successes achieved by deep learning in image recognition employ convolutional neural networks (CNNs) [4–6]. Therefore, NAS finding CNN-related architectures is the focus of this section. The review is divided into four main subsections: **search space**, **search strategy**, **performance estimation strategy**, and **bilevel optimization**, a detailed extension of gradient-based search strategy.

3.1 Search Space

In the language of computational graphs [56], a neural network is a directed acyclic graph (DAG) with a set of nodes Z as exemplified in Fig. 5. Each node $\boldsymbol{z}^{(k)}$ is a tensor (e.g., a feature map in CNNs) with an operation (directed edge) $o^{(k)} \in O$ on its set of parent nodes $I^{(k)}$. The input node is an exception that has no parent nodes. $\boldsymbol{x}$ is the input data while $\hat{\boldsymbol{y}}$ is the output prediction of a task defined. The computation at a node k is defined as:

$$\boldsymbol{z}^{(k)} = o^{(k)}(I^{(k)}) \tag{1}$$

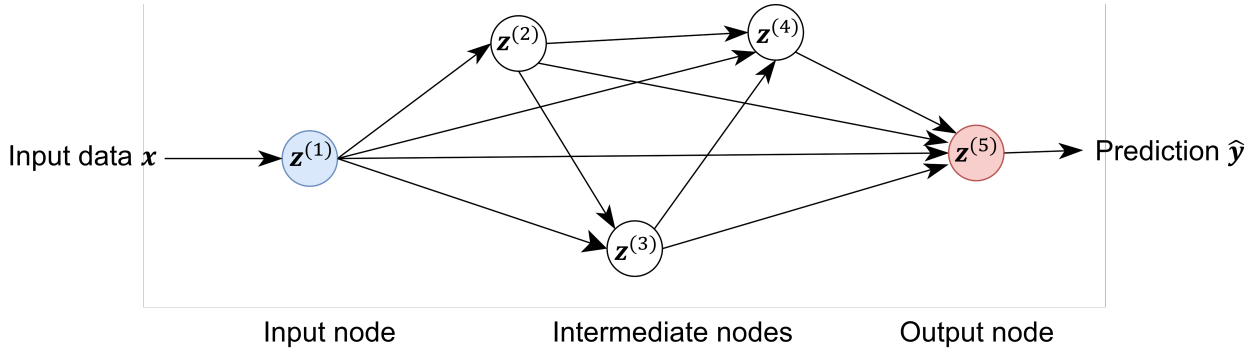


Fig. 5. An example of a DAG of 5 nodes with input and output. Node $\boldsymbol{z}^{(1)}$ is the input node while node $\boldsymbol{z}^{(5)}$ is the output node. Each node has incoming directed edges from all its predecessors, i.e., parent nodes. During training, training data is used as the input data $\boldsymbol{x}$ while the output prediction $\hat{\boldsymbol{y}}$, as a result of, e.g., a image recognition task, is used to compute the training loss. During validation, $\boldsymbol{x}$ is validation data when $\hat{\boldsymbol{y}}$ 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 includes unary operations such as pooling, convolutions, and activation functions or multivariate operations such as addition or concatenation. The network architecture is entirely defined by any representation that specifies the operation between each node and the set of parents. Search space, a subspace of this general definition of neural architectures, refers to the solution set of a neural architecture search strategy limited by its operation space and other imposed constraints. 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. 6 (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 convolutions [57] or depthwise separable convolutions [58];
3. hyperparameters depending on the operation such as number of kernels, kernel size, and strides for a convolutional layer [31, 32, 59].

Since 3. are 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 [60–62]. Skip connections [12], a notable example, allow the construction of complex, multi-branch architectures as shown in Fig. 6 (middle).

3.1.2 Modular Search Space

Motivated by effective handcrafted architectures that consist of repeated motifs [12, 64, 65], these motifs, also known as cells or building blocks, are searched for instead of the entire architecture [62, 66],

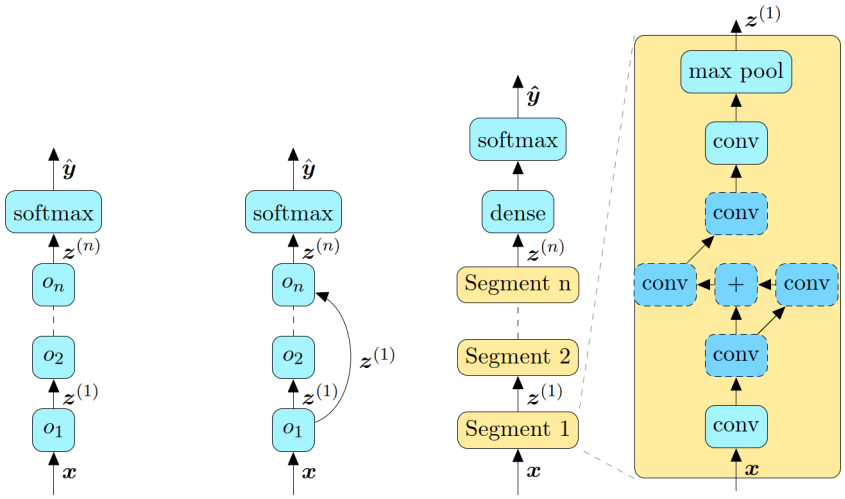


Fig. 6. Global search space: chain-structured (left) [31], with skips (middle) [14], and architecture template (right) [63].

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. The cells are then stacked in a predefined manner to form the final architecture, as exemplified in Fig. 7 (right).

3.1.3 Global vs. Modular Search Space

Global search space allows a greater degree of freedom on operation arrangement, which may be limited only by an architectural template, including admissible structural choices per the architecture definition and needs. A typical example, as illustrated in Fig. 6 (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. Despite 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. [62] 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 [51, 52, 62] as exemplified in Fig. 7.
3. Repeating building blocks have proved to be an effective design in handcrafted architectures for image recognition tasks [4, 12].

Due to the reduced computational cost and enhanced performance, modular search space for NAS is used in this project.

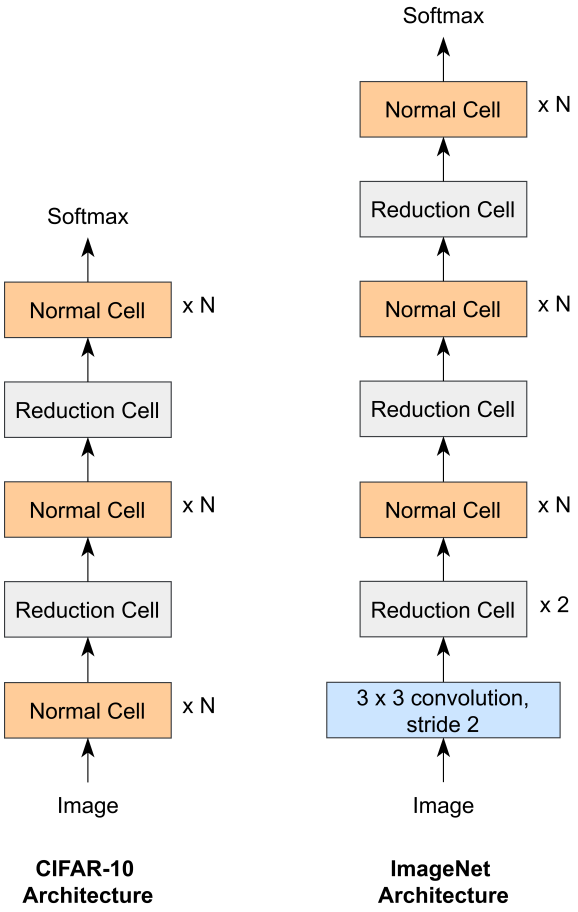


Fig. 7. Predefined macro-architectures stack searched cells to construct final architectures for image recognition on CIFAR-10 (left) and ImageNet (right) [62]. Normally, the CIFAR-10 architecture is used during **cell search** [14, 26] after which the optimal normal cell and the optimal reduction cell are extracted. The extracted two cells are then stacked as the ImageNet architecture for **architecture evaluation** and application on image recognition tasks [14, 26].

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. [14] 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 [14, 62] and Q-learning policy [15], 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 [67, 68], 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, 69], and multi-objective Pareto [70]), removing individuals from a population (e.g., the worst [29], the oldest [71], and nobody [72]), and generating offspring networks (e.g., random initialization [29], and Lamarckian inheritance [60]) are employed to develop NAS.

Bayesian optimization (**BO**) has been applied to NAS using classic Gaussian Process-based BO methods [73, 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 [26] is the pioneer in NAS to successfully relax discrete search strategy to become continuous, which makes the employment of **gradient-based** optimization for architecture search space possible. NAS networks based on gradient-based optimization 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 3.1. A DAG represents a cell here but a complete network there. Edges between each pair of nodes are the weighted sum parameterized by hyperparameters $\boldsymbol{\lambda}$ of all operations from the predefined operation set. The continuous relaxation of candidate operations evolves NAS into an optimization problem for an asset of continuous variables $\boldsymbol{\lambda}$ as illustrated in Fig. 8. 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. 7 (right). The detailed methods to realize DARTS are discussed in Section 4.2.

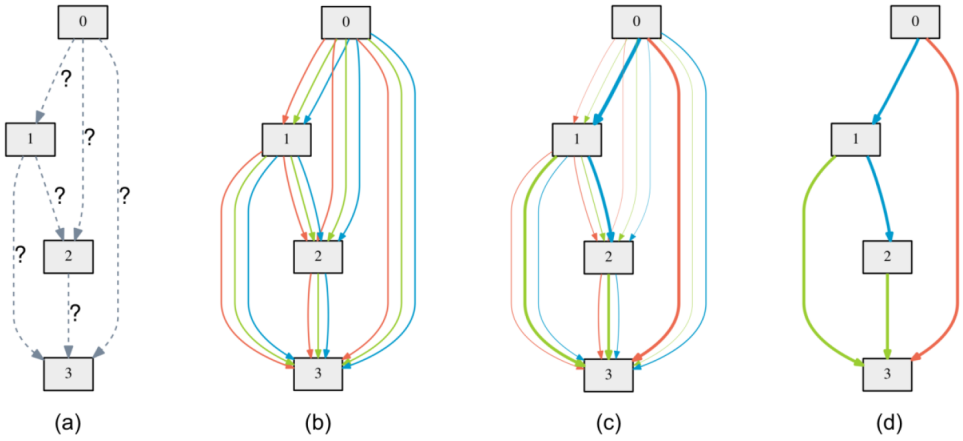


Fig. 8. Cell search strategy of DARTS [26]. (a) Operations on the edges are unknown at first. (b) Continuous relaxation of the search space by placing a mixture, i.e., weighted sum parameterized by $\boldsymbol{\lambda}$ of candidate operations on each edge. (c) Joint optimization of $\boldsymbol{\lambda}$ and the network weights as a bilevel optimization problem. (d) Discretization of final cell architecture based on learnt $\boldsymbol{\lambda}$. Note: Nodes are in rectangular grey boxes and edges are in line arrows. Each colour in line arrows represents a specific operation type from the predefined operation set, such as 3×3 separable convolutions.

Variants of DARTS have been developed to resolve shortcomings of DARTS. I-DARTS [51] associates each node with a softmax considering all input edges, as shown in Fig. 9 to reduce the locality of

DARTS. P-DARTS [80] 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. 7 (left) instead of the actual full-depth architecture as exemplified in Fig. 7 (right) during searching. GDAS [81] samples one subgraph in the cell DAG rather than the entire cell DAG in one iteration as illustrated in Fig. 10, improving search efficiency.

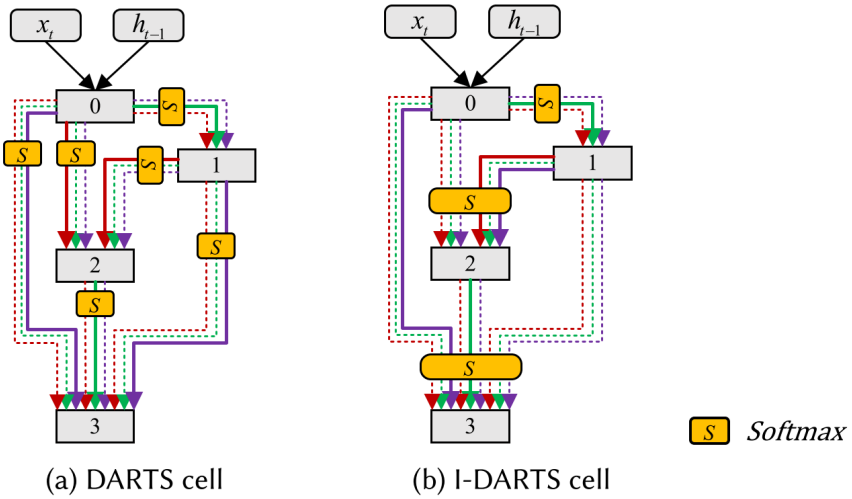


Fig. 9. Search strategy comparison between DARTS and I-DARTS [51]: (a) Softmax is performed on each outgoing edge. (b) Softmax is performed on all incoming edges of a node. Note: Nodes are in rectangular grey boxes and edges are in line arrows. $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$ represent the cell outputs of the previous two layers as explained in Section 4.2.

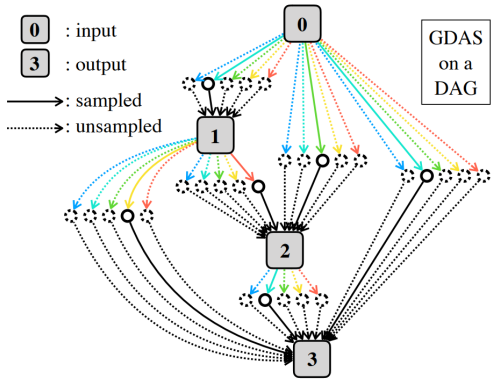


Fig. 10. Only one, i.e., the sampled subgraph in the cell DAG, as indicated by solid connections is trained in one iteration in GDAS [81] instead of the entire cell DAG that takes place in DARTS as illustrated in Fig. 8.

3.2.3 Discrete vs. Continuous Search Strategy

Discrete search strategy demands high computational resources [62] or complex (not end-to-end) NAS design [14]. The computational inefficiency is hard to resolve using discrete search strategies 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 [26, 51, 80, 81].

A discrete search strategy typically involves several functioning blocks, e.g., an additional controller for RL frameworks [14, 62], as illustrated in Fig. 11, and EAs for architecture optimization but gradient

descent for weight optimization [63, 82], i.e., two different methods to solve the outer and inner optimization tasks in the bilevel optimization problem. 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 [26, 51, 80, 81]. Therefore, a gradient-based search strategy is used in this project.

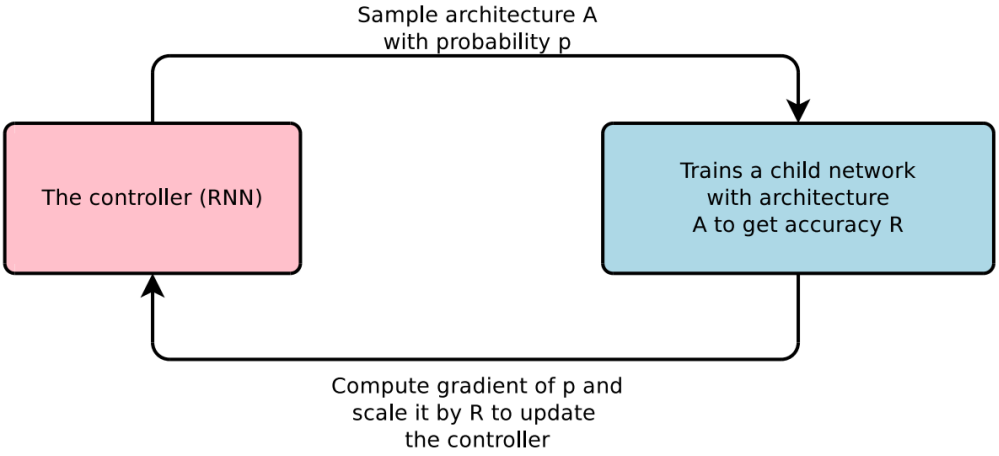


Fig. 11. Framing NAS as a RL problem requires an additional functioning block, i.e., a recurrent neural network (RNN) controller [14].

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. 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, 71]. 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 [62, 83], training on data subset [84], on lower-resolution images [85], fewer filters per layer and fewer cells [62], and fewer backpropagation inner steps for weight updates [26]. 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 [73, 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, 60]. This approach allows search spaces with no intrinsic upper bound on the architecture's size [55].

3.3.4 Weight Sharing / One-Shot Models

One-Shot Architecture Search involves a supergraph (one-shot model) 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) 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 [16] 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 [26], and SNAS [52], resulting in competitive performances.

Nevertheless, this method introduces bias like the lower fidelity estimates method for the same reason (see 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 for improving search efficiency [31, 32, 92, 93]. However, weight inheritance relies on a parent network and implies a system of a complex structure such as Auto-Keras [94] is needed to train a NAS network. Since weight sharing can be integrated into an simpler end-to-end DAS-based NAS method such as DARTS [26], this project uses weight sharing.

3.4 Bilevel Optimization

As a subset of hyperparameter optimization, NAS can be cast as a **bilevel optimization** (**BLO**) problem when gradient-based search strategy is used [26]. BLO contains two levels of optimization tasks, where the inner (lower-level) task in eq. (3) is nested within the outer (upper-level) task in eq. (2) [95, 96]. In NAS, it is usually formulated as:

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

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

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 neural network (child architecture). $\boldsymbol{w}^*$ 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 outer objective $\mathcal{L}_{out}$ defined in terms of the optimal solution to an inner objective $\mathcal{L}_{in}$. $\mathcal{L}_{out}$ is usually validation cost function $\mathcal{L}_{val}$ while $\mathcal{L}_{in}$ 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 gradient descent-based algorithm [26, 52] which involves computations of total gradients $d\mathcal{L}_{val}/d\boldsymbol{\lambda}$ and $d\mathcal{L}_{train}/d\boldsymbol{w}$. $d\mathcal{L}_{val}/d\boldsymbol{\lambda}$ 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 usual backpropagation through the computation graph for gradient descent of outer objective with respect to weights as shown in Fig. 12 and the response gradient in eq. (9).

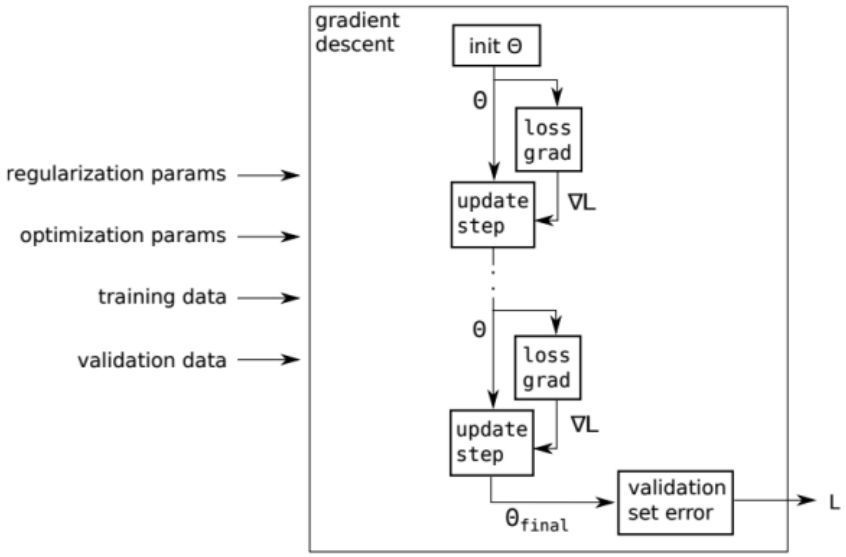


Fig. 12. Computation graph for gradient descent [98]. θs represent weights.

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

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) = \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) + \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}_T}\frac{\partial \boldsymbol{w}_T}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}_T}\frac{\partial \boldsymbol{w}_T}{\partial \boldsymbol{w}_{T-1}}\frac{\partial \boldsymbol{w}_{T-1}}{\partial \boldsymbol{\lambda}} + ... + \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}_T}\frac{\partial \boldsymbol{w}_T}{\partial \boldsymbol{w}_{T-1}}...\frac{\partial \boldsymbol{w}_1}{\partial \boldsymbol{\lambda}} \quad (9)$$

where T is the number of inner steps for weight optimization (or unrolling). The hypergradient is estimated by propagating derivatives backwards through the entire inner optimization (T is determined by when the weight optimization converges) [99, 100], through truncated backpropagation which unrolls T 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 [26].

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 child network 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

In this approach, a network (hypernetwork) generates the weights for another network (main or elementary network) [104, 105], as illustrated in Fig. 13.

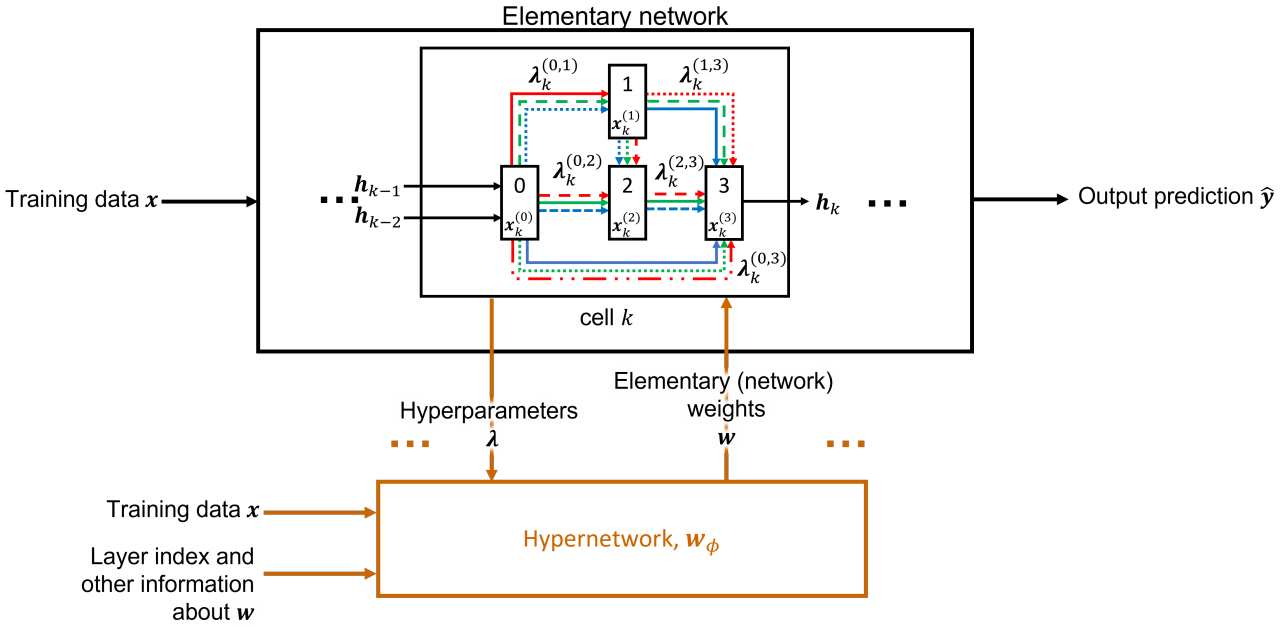


Fig. 13. The hypernetwork (brown) takes in hyperparameters $\boldsymbol{\lambda}$, training data $\boldsymbol{x}$, and other information about elementary weights $\boldsymbol{w}$ as input. It outputs the elementary (network) weights $\boldsymbol{w}$ after every outer step of $\boldsymbol{w}_{\boldsymbol{\phi}}$ updates. $\boldsymbol{h}_k$ is the output of cell k. $\boldsymbol{h}_{k-1}$ and $\boldsymbol{h}_{k-2}$ are the outputs of the previous two cells. A refined figure of the cell structure is in Fig. 14 and a more detailed discussion of the cell structure is in Section 4.2. Note that during **cell search**, cells are actually stacked as a small architecture in a predefined repeating pattern as exemplified in Fig. 7 (left). After searching, the optimal normal cell and the optimal reduction cell are extracted and stacked as a large architecture as exemplified in Fig. 7 (right) for training and **architecture evaluation**.

In simple terms, the hypernetwork learns the function ($\boldsymbol{r}(\boldsymbol{\lambda}) = \arg\min_w \mathcal{L}_{train}(\boldsymbol{\lambda}, \boldsymbol{w})$) in eq. (6) by inputting hyperparameters $\boldsymbol{\lambda}$ which is a function of training data $\boldsymbol{x}$, training data $\boldsymbol{x}$, and other information about elementary (network) weights $\boldsymbol{w}$. The hypernetwork outputs optimal elementary network weights $\boldsymbol{w}^*$ at convergence.

The hypernet is a function $\boldsymbol{w}_{\boldsymbol{\phi}}$ with hypernetwork weights $\boldsymbol{\phi}$:

$$\boldsymbol{w} = \boldsymbol{w}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}) \tag{10}$$

At convergence,

$$\boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda}) = \boldsymbol{w}^* = \boldsymbol{r}(\boldsymbol{\lambda}) \tag{11}$$

[104] has proved theoretically that

1. if $\boldsymbol{w}_{\boldsymbol{\phi}}$ 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.

In other words,

$$\mathcal{L}_{val}(\boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda})) = \mathcal{L}_{val}(\boldsymbol{w}^*(\boldsymbol{\lambda})) \tag{12}$$

As a result, 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

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{r}(\boldsymbol{\lambda})) = \frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda})) = \underbrace{\frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda}, \boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda}))}_{\text{direct gradient}} + \underbrace{\overbrace{\left(\frac{\partial \boldsymbol{w}_{\boldsymbol{\phi}^*}}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda})\right)^{\top}}^{\text{response Jacobian}} \frac{\partial \mathcal{L}_{val}}{\partial \boldsymbol{w}_{\boldsymbol{\phi}}}(\boldsymbol{\lambda}, \boldsymbol{w}_{\boldsymbol{\phi}^*}(\boldsymbol{\lambda}))}_{\text{response gradient}} \tag{13}$$

The response gradient is now just the derivative of validation loss with respect to the input of the hypernet which can be computed easily via ordinary backpropagation through the hypernet.

This hypernet-based method enables amortizing the cost of computing the hypergradient without the inherent trade-off dilemma of implicit differentiation and unrolling discussed in Section 3.4.1. This is because only $\boldsymbol{w}_{\boldsymbol{\phi}}$, the weights of the hypernet, has to be optimized instead of the weights of all the child architectures during cell search phase. Nevertheless, the hypernetwork is an exceptionally high-dimensional mapping resulting in colossal memory requirement and a long amortizing process as it can have several times as many parameters as the elementary model. For instance, a fully-connected layer of H units to output D parameters (weights) needs at least $D \times H$ hypernetwork parameters for $\boldsymbol{\phi}$. Therefore, **Self-Tuning Network** (**STN**) [105], a scalable hypernetwork-based approach to BLO, was developed to resolve this issue by representing hypernet compactly and efficiently.

3.4.3 Self-Tuning Network

Self-Tuning Network (**STN**) [105] is a hypernet whose architecture computes activations of the elementary network and includes a correction term dependent on $\boldsymbol{\lambda}$. This construction allows compact approximation of $\boldsymbol{w}^*$ by modeling the best-response of each row in a layer's weight matrix as a rank-one affine transformation of the hyperparameters. The approximation has also been justified by applying it to a simple model and comparing it against its calculated $\boldsymbol{w}^*$ [105]. An extended discussion on STN is in Section 4.3. STNs have shown better generalization results than competing approaches in hyperparameter optimization [105]. To the best of the author's knowledge, STNs have not been applied to NAS problems. Therefore, it is decided to be tested on NAS problems in this project.

4 Methodology and Methods

NAS network design consists of three main components: search space, search strategy, and performance estimation strategy. DARTS is used as the foundational model in this project due to its effectiveness and elegance in exploiting gradient information. Its search strategy uses unrolling with finite difference approximation to estimate the hypergradient for BLO. This project plans to replace the unrolling method with an STN in an attempt to improve image recognition, especially FER results.

This section first discusses the **datasets** used. Then the technical details of **DARTS** and **STN** are discussed. After that, the training procedure of **DARTS-STN** is presented.

4.1 Datasets

For general image recognition, two datasets are used: Canadian Institute For Advanced Research (CIFAR-10) [106] and ImageNet [107]. CIFAR-10 consists of 60000 32 $\times$32 colour images equally distributed in 10 different classes of vehicles and animals. Imagenet consists of 1.2 million 469 $\times$ 387 colour images distributed in 1000 classes of animals, vehicles, food, appliances, buildings, etc.. Each class has at least 1000 images.

For FER, three datasets are used: Facial Expression Recognition 2013 Dataset (FER2013) [108], Extended Cohn-Kanade (CK+) [109], and Japanese Female Facial Expression (JAFFE) [110] are used. FER2013 consists of 30000 48 $\times$ 48 grayscale images. CK+ consists of 593 640 $\times$ 480 colour video sequences recorded at 30 frames per second. JAFFE consists of 213 256 $\times$ 256 graysclae images. All datasets use the same 7 classes of facial expressions include anger, disgust, fear, happiness, sadness, surprise and neutral expression.

4.2 Preliminary – DARTS

4.2.1 Search Space

As discussed in Sections 3.1.2 and 3.2.2, two types of cells are found in the modular search space: a normal cell and a reduction cell. A cell is a DAG consisting of an ordered sequence of N nodes (2 input nodes and 1 output node) with one directed edge between every two nodes as illustrated in Fig. 8(a). Each node $\boldsymbol{x}^{(j)}$ is a latent representation (a feature map in CNN) and each directed edge $\Theta^{(i,j)}$ (directed from $\boldsymbol{x}^{(i)}$ to $\boldsymbol{x}^{(j)}$) is a weighted sum of operations $o \in O$ that transforms $\boldsymbol{x}^{(i)}$. Fig. 8 shows an example of the DAG during cell search. In the example, the predefined operation set has a dimension of 4, each color in line arrows representing each operation. Each directed edge, as a weighted sum of the 4 distinct operations, is shown by 4 lines of distinct colors. Fig. 8 is a simplified figure for explanation, DARTS has a higher dimensional predefined operation set.

Fig. 3(b) shows a searched normal cell in DARTS while Fig. 14 shows the general structure of a cell in DARTS. The 2 input nodes are cell outputs of previous two layers (cells are stacked into a **small**, predefined architecture as shown in Fig. 7 (left) for architecture search). The output node is the depthwise concatenation of all intermediate nodes. Each intermediate node is a function of all its predecessors:

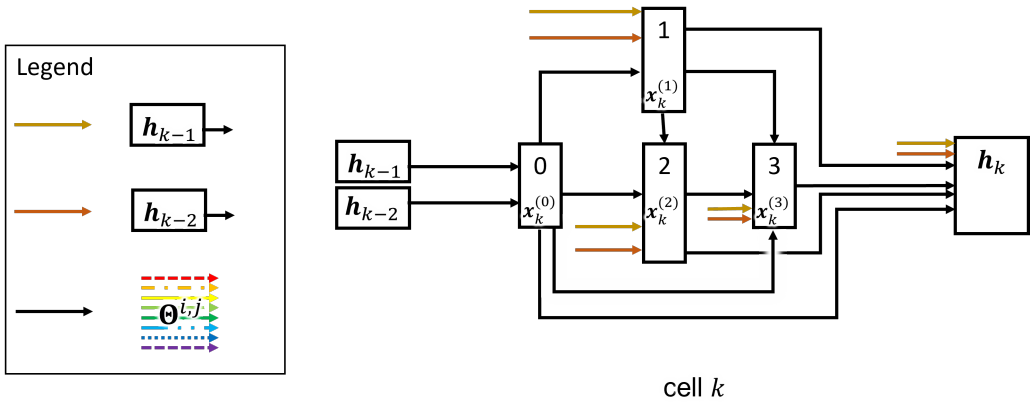


Fig. 14. A cell in DARTS has two inputs h_{k-1} and h_{k-2} from the previous two cells and one output node h_k. There are four intermediate nodes in between. The predefined operation set used in DARTS has a dimension of 8, representing by the 8 rainbow-coloured arrows that form the directed edge $\Theta^{(i,j)}$ where the directed edge points from node i to node j.

$$\boldsymbol{x}^{(j)} = \sum_{i<j} \Theta^{(i,j)} \left(\boldsymbol{x}^{(i)} \right) \tag{14}$$

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 $\Theta^{(i,j)}$:

$$\Theta^{(i,j)}(\boldsymbol{x}) = \sum_{o \in O} \frac{exp(\lambda_o^{(i,j)})}{\sum_{o' \in O} exp(\lambda_{o'}^{(i,j)})} o(\boldsymbol{x}) \tag{15}$$

where $\boldsymbol{\lambda}^{(i,j)}$ is a $|O|$-dimensional vector representing operation weights (hyperparameters) for a pair of nodes (i, j). The architecture search task thus reduces to learning a set of continuous variables (can be seen as a matrix) $\boldsymbol{\lambda} = \{\lambda^{(i,j)}\}$. 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 $o^{(i,j)}$:

$$o^{(i,j)} = \arg\max_{o \in O} \boldsymbol{\lambda}^{(i,j)} \tag{16}$$

$\boldsymbol{\lambda}$ is the hyperparameters and the upper-level variable in the BLO problem (eq. (5) and eq. (6)) of DARTS. The BLO question is now defined. Section 4.2.2 talks about how DARTS solves it while Section 4.3 talks about how STN solves it.

4.2.2 Unrolling as Search Strategy and Performance Estimation Strategy

This sub-subsection explains the details of **unrolling** methods used by DARTS [26]. DARTS algorithm [26] is as follows:

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**
  Update architecture $\boldsymbol{\lambda}$ by descending $\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{w} - \xi\frac{\partial\mathcal{L}_{train}}{\partial\boldsymbol{w}}(\boldsymbol{w},\boldsymbol{\lambda}),\boldsymbol{\lambda})$
  Update weights $\boldsymbol{w}$ by descending $\frac{\partial\mathcal{L}_{train}}{\partial\boldsymbol{w}}(\boldsymbol{w},\boldsymbol{\lambda})$
Derive the final architecture based on the learned $\boldsymbol{\lambda}$

The $\boldsymbol{\lambda}$ update happens every outer step while the $\boldsymbol{w}$ update happens every inner step. DARTS approximates hypergradient as follows:

$$\frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{w}^*(\boldsymbol{\lambda}),\boldsymbol{\lambda}) \approx \frac{d\mathcal{L}_{val}}{d\boldsymbol{\lambda}}(\boldsymbol{w} - \xi\frac{\partial\mathcal{L}_{train}}{\partial\boldsymbol{w}}(\boldsymbol{w},\boldsymbol{\lambda}),\boldsymbol{\lambda}) \tag{17}$$

where $\boldsymbol{w}$ is the weights at the specific outer step and ξ is set to be the same as the learning rate of the inner step as a simple working strategy. Only one inner step is performed when descending $\boldsymbol{\lambda}$ and $\boldsymbol{w}$ within an outer step. Applying chain rule to the hypergradient approximation (right-hand side of eq. (17)) results in:

$$\frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{\lambda}}(\boldsymbol{w}',\boldsymbol{\lambda}) - \xi\frac{\partial\mathcal{L}^2_{train}}{\partial\boldsymbol{\lambda}\partial\boldsymbol{w}}(\boldsymbol{w},\boldsymbol{\lambda})\frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{w}'}(\boldsymbol{w}',\boldsymbol{\lambda}) \tag{18}$$

where $\boldsymbol{w}' = \boldsymbol{w} - \xi\frac{\partial\mathcal{L}_{train}}{\partial\boldsymbol{w}}(\boldsymbol{w},\boldsymbol{\lambda})$. The second term in eq. (18) is approximated using finite difference method to avoid the expensive computation of the matrix-vector product lying inside the term:

$$\frac{\partial\mathcal{L}^2_{train}}{\partial\boldsymbol{\lambda}\partial\boldsymbol{w}}(\boldsymbol{w},\boldsymbol{\lambda})\frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{w}'}(\boldsymbol{w}',\boldsymbol{\lambda}) \approx \frac{\frac{\partial\mathcal{L}_{train}}{\partial\boldsymbol{\lambda}}(\boldsymbol{w}^+,\boldsymbol{\lambda}) - \frac{\partial\mathcal{L}_{train}}{\partial\boldsymbol{\lambda}}(\boldsymbol{w}^-,\boldsymbol{\lambda})}{2\epsilon} \tag{19}$$

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.

DARTS employs two approximation methods: first-order approximation which sets $\xi = 0$ and second-order approximation which sets $\xi > 0$. The first-order approximation is the same as unrolling with $T = K = 1$ discussed in Section 3.4.1. The second-order approximation achieves a better test error performance with a higher number of GPU days [26]. For both these two methods, there is no theoretical guarantee for convergence. In practice, DARTS chooses ϵ carefully to be $0.01/\left\|\frac{\partial\mathcal{L}_{val}}{\partial\boldsymbol{w}'}(w',\alpha)\right\|_2$ to reach convergence for $\boldsymbol{\lambda}$.

4.3 Self-Tuning Network as Search Strategy and Performance Estimation Strategy

As discussed in Section 3.4, the architecture $\boldsymbol{\lambda}$ and network weights $\boldsymbol{w}$ can be jointly optimized during search phase as a BLO problem. As mentioned in Section 3.4.2, a BLO problem can be efficiently solved using **Self-Tuning Network** (**STN**). STN is constructed based on a compact approximation of best response optimal weights $\boldsymbol{w}^*$ by modeling the best-response of each row in an (elementary network) layer's weight matrix $\boldsymbol{W}$ as a rank-one affine transformation of the hyperparameters $\boldsymbol{\lambda}$. The detailed derivation and justification are in [105], and only the method is presented here. In this subsection, the best-response approximation function $\hat{\boldsymbol{w}}_{\boldsymbol{\phi}}$ that is memory efficient and scalable is first constructed. Then, the method to automatically adjust the scale of the training neighborhood where

the weights of the STN $\boldsymbol{\phi}$ trained on is discussed. Finally, the training algorithm is presented. Noting sometimes $\boldsymbol{W}$ instead of $\boldsymbol{w}$ is used in the following to emphasize network weights as a matrix but both notations are equivalent in this proposal.

4.3.1 An Efficient Best-Response Approximation for Neural Networks

The best-response (what the STN should output at convergence) for a given elementary layer's weight matrix $\boldsymbol{W} \in \mathbb{R}^{D_{out} \times D_{in}}$ and bias $\boldsymbol{b} \in \mathbb{R}^{D_{out}}$ can be approximated as an affine transformation of the hyperparameters $\boldsymbol{\lambda}$:

$$\hat{\boldsymbol{W}}_{\phi}(\boldsymbol{\lambda}) = \boldsymbol{W}_{\text{elem}} + (\boldsymbol{V}\boldsymbol{\lambda}) \odot_{\text{row}} \boldsymbol{W}_{\text{hyper}}, \quad \hat{\boldsymbol{b}}_{\phi}(\boldsymbol{\lambda}) = \boldsymbol{b}_{\text{elem}} + (\boldsymbol{C}\boldsymbol{\lambda}) \odot \boldsymbol{b}_{\text{hyper}} \tag{20}$$

where $\odot$ is elementwise multiplication, $\odot_{\text{row}}$ is row-wise rescaling, and $\boldsymbol{V}$ and $\boldsymbol{C}$ are scaling factors. The architecture sums up the usual elementary weight/bias and an additional weight/bias scaled by linearly transformed hyperparameters. Alternatively, the architecture can be seen as operating directly on the pre-activations of the elementary layer, with an added correction to account for the hyperparameters (STN has another input $\boldsymbol{x}$):

$$\hat{\boldsymbol{W}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda})\boldsymbol{x} + \hat{\boldsymbol{b}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda}) = [\boldsymbol{W}_{\text{elem}}\, \boldsymbol{x} + \boldsymbol{b}_{\text{elem}}] + \left[(\boldsymbol{V}\boldsymbol{\lambda}) \odot \left(\boldsymbol{W}_{\text{hyper}}\, \boldsymbol{x}\right) + (\boldsymbol{C}\boldsymbol{\lambda}) \odot \boldsymbol{b}_{\text{hyper}}\right] \tag{21}$$

This best-response architecture is memory-efficient: $\hat{\boldsymbol{W}}_{\boldsymbol{\phi}}$ has $D_{out}(2D_{in} + n)$ parameters and $\hat{\boldsymbol{b}}_{\boldsymbol{\phi}}$ has $D_{out}(2 + n)$, where n is the number of hyperparameters. The number of hypernetwork parameters $\boldsymbol{\phi}$ is dependant on only the elementary weight size and the number of hyperparameters, causing the hypernetwork to have only about two times the size as the elementary network.

4.3.2 Adapting the Hyperparameter Distribution

Lorraine et al. [104] (the original hypernet) locally approximates $\boldsymbol{w}^*$ in a neighbourhood around the current upper-level parameter $\boldsymbol{\lambda}$ for $p(\boldsymbol{\lambda})$ to have enough support. They set the hyperparameter distribution to be a factorized Gaussian noise distribution, i.e., $p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})$ with a fixed scale parameter $\boldsymbol{\sigma} \in \mathbb{R}^n_+$ and $\boldsymbol{\epsilon}$ as perturbation. $\boldsymbol{\phi}$ is found by minimizing the (inner) objective:

$$\mathbb{E}_{\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})} \left[f\left(\boldsymbol{\lambda} + \boldsymbol{\epsilon}, \hat{\mathbf{w}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda} + \boldsymbol{\epsilon})\right)\right] \tag{22}$$

Intuitively, the lower-level parameter $\hat{\mathbf{w}}_{\boldsymbol{\phi}}$ learns how to respond only when the upper-level parameter $\boldsymbol{\lambda}$ is perturbed by a small amount $\boldsymbol{\epsilon}$. $\boldsymbol{\phi}$ is updated to minimize eq. (22) while $\boldsymbol{\lambda}$ is updated to minimize the outer objective eq. (23).

$$\min_{\boldsymbol{\lambda} \in \mathbb{R}^n} \mathcal{L}_{val}\left(\boldsymbol{\lambda}, \hat{\mathbf{w}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda})\right) \tag{23}$$

As illustrated in Fig. 15, the size of entries of $\boldsymbol{\sigma}$ control the scale of the hyperparameter distribution on which $\boldsymbol{\phi}$ is trained. The entries must remain large enough relative to the loss landscape to ensure $\hat{\mathbf{w}}_{\boldsymbol{\phi}}$ can capture the local shape around the current hyperparameter values. This is an issue in the

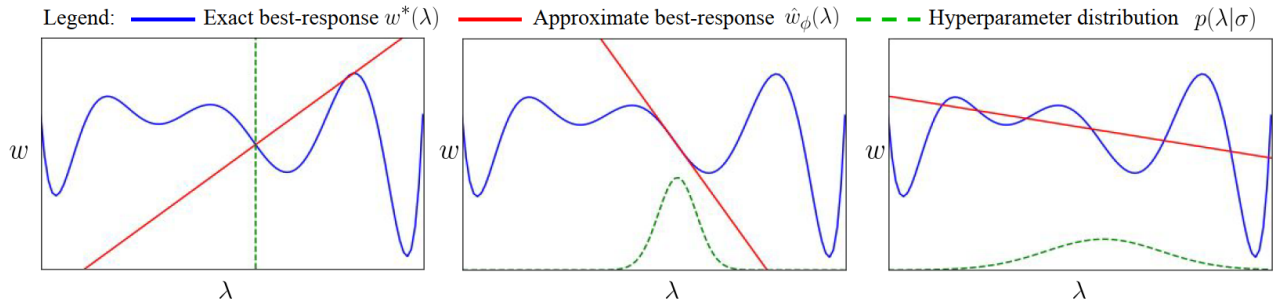


Fig. 15. The effect of sampled neighbourhood size. Left: Too small the sampled neighbourhood cannot guarantee approximation learnt has its gradient matches that of the best-response. Middle: Neither too small nor too wide the sampled neighbourhood guarantees approximation learnt has its gradient matches that of the best-response. Right: Too wide the sampled neighbourhood cannot match the gradient of the approximation learnt to that of the best-response due to insufficient modelling flexibility.

original hypernet which sets $\boldsymbol{\sigma}$ fixed. These issues can be resolved by keep adjusting $\boldsymbol{\sigma}$ during searching based on how sensitive the upper-level objective is with respect to the sampled hyperparameters [105]. Therefore, an entropy term $\mathbb{H}$ weighted by $\tau \in \mathbb{R}_+$ is included to enlarge the entries of $\boldsymbol{\sigma}$. The resulting objective is:

$$\mathbb{E}_{\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}|\boldsymbol{\sigma})} \left[\mathcal{L}_{val} \left(\boldsymbol{\lambda} + \boldsymbol{\epsilon}, \hat{\mathbf{w}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda} + \boldsymbol{\epsilon}) \right) \right] - \tau \mathbb{H}[p(\boldsymbol{\epsilon} \mid \boldsymbol{\sigma})] \tag{24}$$

Similar objectives have been used for variational inference for representation learning [111] and for better training [112]. In summary, for this project, the BLO problem is defined in Section 4.2 following [26], $\boldsymbol{\phi}$ is updated to minimize eq. (22) [104] while $\boldsymbol{\lambda}$ is updated to minimize eq. (24) [105].

4.4 DARTS-STN

DARTS with STN (DARTS-STN) discussed encompasses the search space, search strategy, and performance estimation strategy used for cell search in this project. Since direct cell search on the large and high-resolution ImageNet is costly, cell search is performed on CIFAR-10 from which cells are extracted for architecture evaluation on ImageNet. The training procedure of DARTS-STN network is as follows.

First, two types of cells: normal and reduction, are stacked as a small architecture in a predefined repeating pattern as shown in Fig. 7 (left) to build the network for cell search training. The small network is then fed into the DARTS-STN cell search training algorithm to be trained on CIFAR-10. After the cell search training, the two types of cells are extracted and stacked as a large architecture defined in Fig. 7 (right). The architecture is then trained on ImageNet for performance evaluation (or architecture evaluation) on general image recognition. DARTS is trained the same way for performance comparison. For FER, the small network is trained on FER2013, while the large network will be trained on all three datasets for architecture evaluation.

4.4.1 Cell Search Training Algorithm

The gradient descent algorithm trains the STNs by alternating between updating $\boldsymbol{\phi}$ for T_{train} steps to minimize eq. (22) and updating $\boldsymbol{\lambda}$ and $\boldsymbol{\sigma}$ for T_{val} steps to minimize eq. (24). The complete DARTS-STN algorithm is shown as Algorithm 2.

Algorithm 2 DARTS-STN Training Algorithm

Initialize: Best-response approximation parameters $\boldsymbol{\phi}$, hyperparameters $\boldsymbol{\lambda}$, learning rates $\{\alpha_i\}_{i=1}^{3}$
  while not converged **do**
    for $t = 1, ..., T_{train}$ **do**
      $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon} \mid \boldsymbol{\sigma})$
      $\phi \leftarrow \boldsymbol{\phi} - \alpha_1 \frac{\partial}{\partial \phi} f\left(\boldsymbol{\lambda} + \boldsymbol{\epsilon}, \hat{\mathbf{w}}_{\phi}(\boldsymbol{\lambda} + \boldsymbol{\epsilon})\right)$
    for $t = 1, ..., T_{val}$ **do**
      $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon} \mid \boldsymbol{\sigma})$
      $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \alpha_2 \frac{\partial}{\partial \boldsymbol{\lambda}} \left(F\left(\boldsymbol{\lambda} + \boldsymbol{\epsilon}, \hat{\mathbf{w}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda} + \boldsymbol{\epsilon})\right) - \tau \mathbb{H}[p(\boldsymbol{\epsilon} \mid \boldsymbol{\sigma})]\right)$
      $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} - \alpha_3 \frac{\partial}{\partial \boldsymbol{\sigma}} \left(F\left(\boldsymbol{\lambda} + \boldsymbol{\epsilon}, \hat{\mathbf{w}}_{\boldsymbol{\phi}}(\boldsymbol{\lambda} + \boldsymbol{\epsilon})\right) - \tau \mathbb{H}[p(\boldsymbol{\epsilon} \mid \boldsymbol{\sigma})]\right)$

4.4.2 Performance Evaluation / Architecture Evaluation

There will be four final neural networks based on methods used and datasets searched on: DARTS, DARTS-STN, DARTS-FER, and DARTS-STN-FER. The performance of the four models will be evaluated as discussed in Section 4.4. The metrics for general image recognition models and FER models will be the same, including test error, number of parameters, number of flops, and search cost in GPU days.

If the newly-developed NAS solution performs better than DARTS, the next phase of the project can be started, i.e., combining STN with a newer DAS model that performs better than DARTS to achieve state-of-the-art results on image recognition and FER tasks.

5 Scope, Project Plan & Timeline

5.1 Project Scope

This project's scope is to utilize one of the latest and improved gradient-based bilevel optimization solutions that has never been used on NAS, on NAS by integrating it into existing NAS solutions.

The following is in-scope:

1. All coding of the proposed new NAS solutions.
2. Compare the performance of the revamped NAS solutions with the chosen benchmark NAS solutions by testing on ImageNet, CK+, and JAFFE datasets.

The following is out-of-scope:

1. Mathematical justification of the chosen gradient-based bilevel optimization solution.
2. Mathematical analysis of the proposed NAS solutions.
3. Improving the other parts of the chosen existing NAS solutions.

5.2 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 1.
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 STN).
8. Identify a latest gradient-based bilevel optimization solution (STN was chosen).
9. Code a simple BLO solution (e.g., unrolling) to test on a toy problem.
10. Understand STN thoroughly.
11. Understand the code of STN.
12. Code STN to solve a toy problem.

13. Code STN to solve a HO problem.

Tasks for FYP B in sequence:

1. Integrate STN into DARTS in code.
2. Test DARTS and DARTS-STN on chosen datasets.
3. Compare and analyze the performance of DARTS and DARTS-STN.
4. Identify and learn a more competitive existing gradient-based NAS solution as reference model 2 (RM2).
5. Code RM2 out and test it on its original experiments.
6. Integrate STN into RM2 in code.
7. Test RM2 and RM2-STN on chosen datasets.
8. Compare and analyze the performance of DARTS, DARTS-STN, RM2, and RM2-STN.

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. Reference Model 1,
5. Reference Model 2, and
6. Project Documentation.

Fig. 16 shows the Gantt Chart for the project timeline. The black line arrow indicates the critical path. In Fig. 16, 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. 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, tasks that need a continuous flow of work such as understanding code and testing of models will be continued.

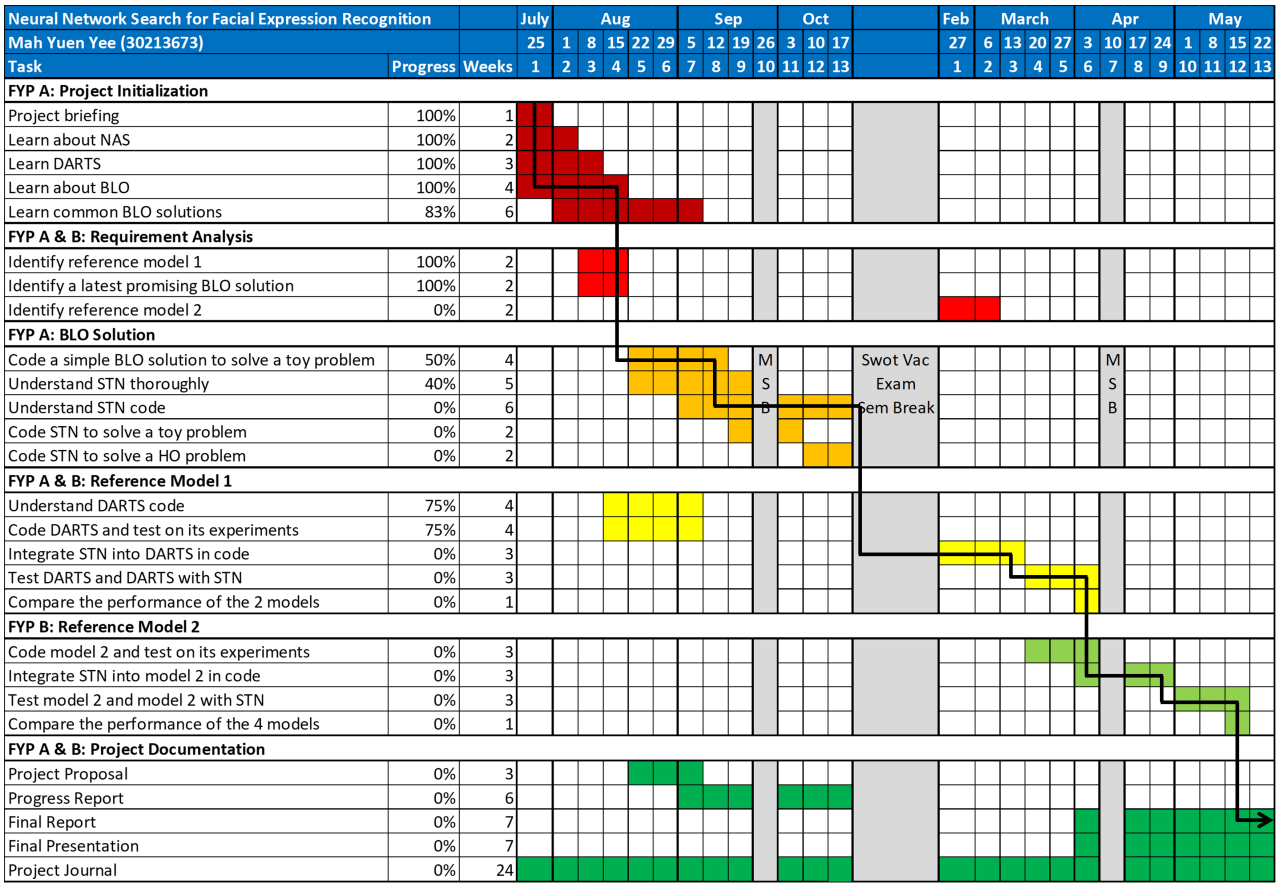


Fig. 16. The Gantt Chart for the project. MSB stands for Mid-Sememster Break.

6 Project Progress

6.1 Preliminary Results and Discussion

6.1.1 Unrolling in DARTS

As hypothesized in Section 2.1, the unrolling algorithm employed by DARTS has a high probability of reducing DARTS to only converge to suboptimal $\boldsymbol{\lambda}$ during cell search outlined in Algorithm 2. An experiment is conducted to investigate the convergence performance of DARTS unrolling algorithm and to validate if the hypothesis is valid in this aspect. Vectors are not bold in this section for simplicity and clarity.

6.1.1.1 Toy Problem

A toy problem, reminiscent of [101] is used for the experiment. The simple problem is framed as, for $\lambda, w \in \mathbb{R}^2$:

$$\begin{aligned} &\min_{\lambda} \|\hat{w}^*\|^2 + 10\,\|\sin(\hat{w}^*)\|^2 =: f(\hat{w}^*, \lambda) \\ &\text{s.t. } \hat{w}^* \approx \arg\min \frac{1}{2}(w-\lambda)^\top G(w-\lambda) =: g(w, \lambda) \end{aligned} \tag{25}$$

where $\|\cdot\|$ is the ℓ_2 norm, sine is applied elementwise, $G = \operatorname{diag}\left(1, \frac{1}{2}\right)$, and $\hat{w}^*$ is defined as the result of T steps of unrolling and K steps of backpropagating on g with inner learning rate and $\xi = 0.1$, initialized at $w_0 = (2, 2)$ and $\lambda_0 = (1, 1)$. Graph of f is shown in Fig. 17.

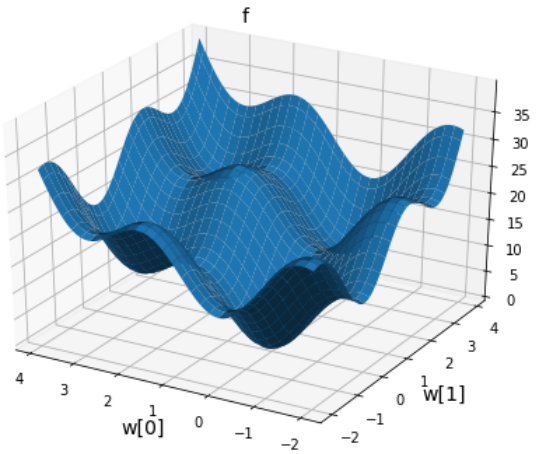


Fig. 17. Graph of f.

The optimal λ is $(0, 0)$. Four unrolling settings are experimented: full backpropagation ($T = K = 100$), truncated backpropagation ($T = 100, K = 1$), second-order approximation and first-order approximation ($T = 1, K = 1$) used by DARTS.

6.1.1.2 Search Cost

The results of λ convergence are shown in Fig. 18. Consistent with [26], the first-order approximation method diverges and cannot even handle this simple toy problem. The total time taken for each of the other unrolling methods to convergence is shown in Table I.

TABLE I
THE TOTAL TIME TAKEN FOR THE UNROLLING METHODS TO CONVERGENCE.

Unrolling method	Number of outer steps	Total time taken (s)
Full	3	0.9421
Truncated	9	0.4797
Second-order	24	0.1074

Even if the full and truncated backpropagations take much less outer steps for λ to reach convergence, they take 9 and 4 times more time than the second-order approximation respectively. This clearly shows that the full backpropagation and truncated backpropagation cannot be used to replace the second-order approximation in DARTS in order to preserve the low search cost (in GPU days) DARTS consumes.

6.1.1.3 Suboptimal Convergence

The inner learning rate and ξ are changed to test whether the second-order approximation method can converge to the optimal λ under different conditions. When the inner learning rate and ξ are incremented from 0.1 to 0.13, the second-order approximation method fails to converge to the optimal λ when the full and back propagations still manage to do so. The results are shown in Fig. 19. This shows that the second-order approximation method is very sensitive to the working setting even in solving a toy problem and has a high probability of converging to suboptimal λ during cell search in DARTS since the search space has a much higher dimension and complexity compared to the toy problem.

6.1.1.4 Summary

In conclusion, the unrolling approximation methods used in DARTS have a high chance of converging λ to suboptimal values during cell search. The full and truncated backpropagations cannot be used in order to preserve the low search cost of DARTS. In other words, different unrolling methods have different vulnerabilities or shortcomings for application in NAS problems. Therefore, the hypergradient approximation component in DARTS has a room for performance improvement by replacing the unrolling method with other inherently more efficient hypergradient approximation method such as STN [105].

6.1.2 DARTS

Since DARTS [26] is used as the baseline model in this project, an experiment is conducted to replicate the original preliminary results. A DARTS cell search is ran in the experiment for 40 epochs to search for the optimal normal and reduction cells following all the original DARTS settings. The cell search training and validation results are shown in Fig. 20. The searched cells are shown in Fig. 21 and Fig.

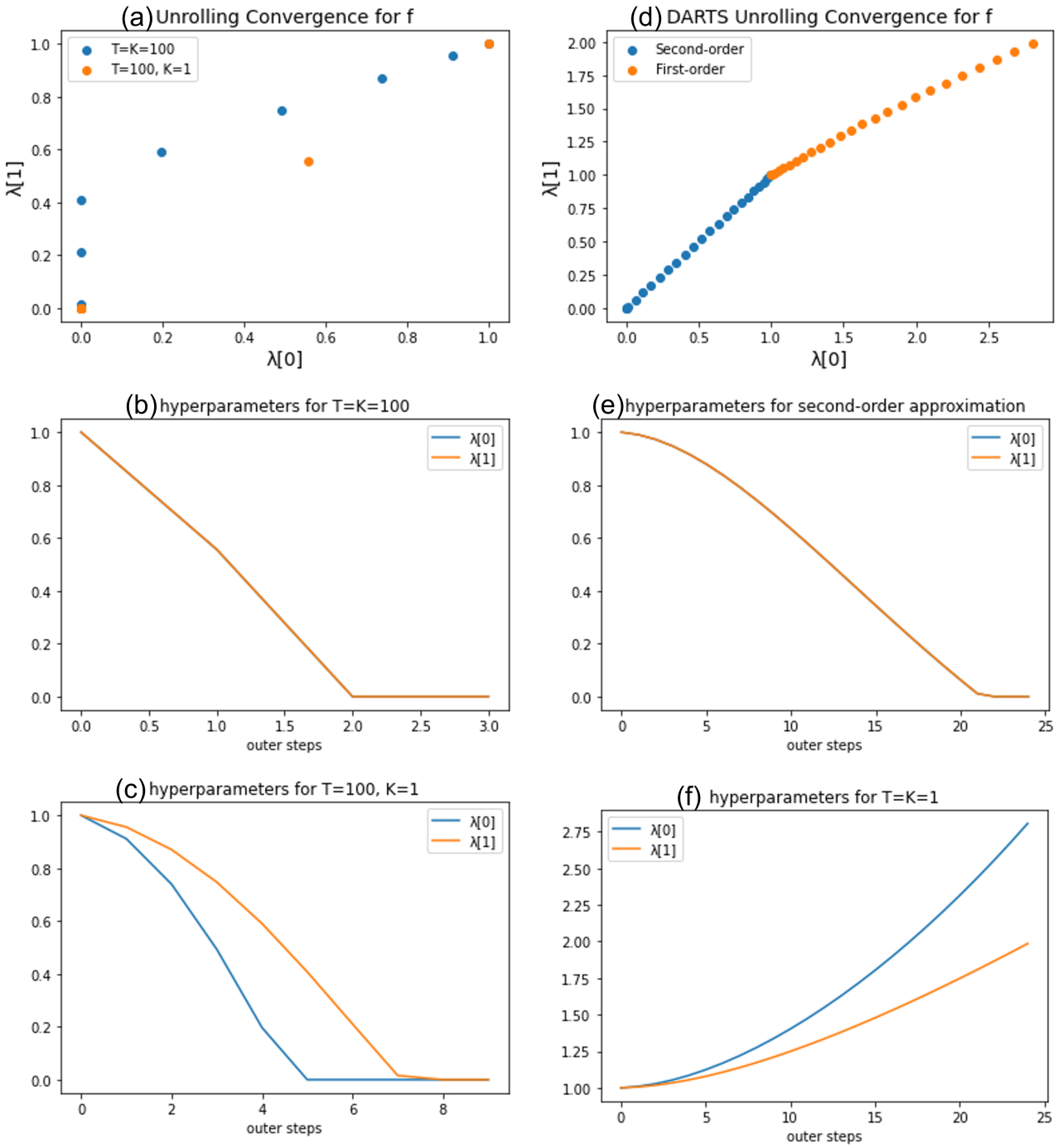


Fig. 18. (a) λ convergence for f using full backpropagation ($T = K = 100$) and truncated backpropagation ($T = 100, K = 1$). λ convergence using (b) full backpropagation ($T = K = 100$) and (c) truncated backpropagation ($T = 100, K = 1$) with respect to the number of outer steps. (d) λ convergence for f using unrolling methods, i.e., second- and first-order approximation employed in DARTS. λ convergence using (e) second-order approximation ($T = K = 1$ with $\xi > 0$) and (f) first-order approximation ($T = K = 1$) with respect to the number of outer steps.

22. The best-performing searched cells from the original DARTS paper are also shown in Fig. 23 for cell comparison.

The final validation accuracy of the searched micro-architecture on CIFAR-10 is 88.1240% which is consistent with the range of around 88% to 89.5% in the original DARTS paper. The searched cells in the experiment are slightly different from the original DARTS paper. This is as expected since

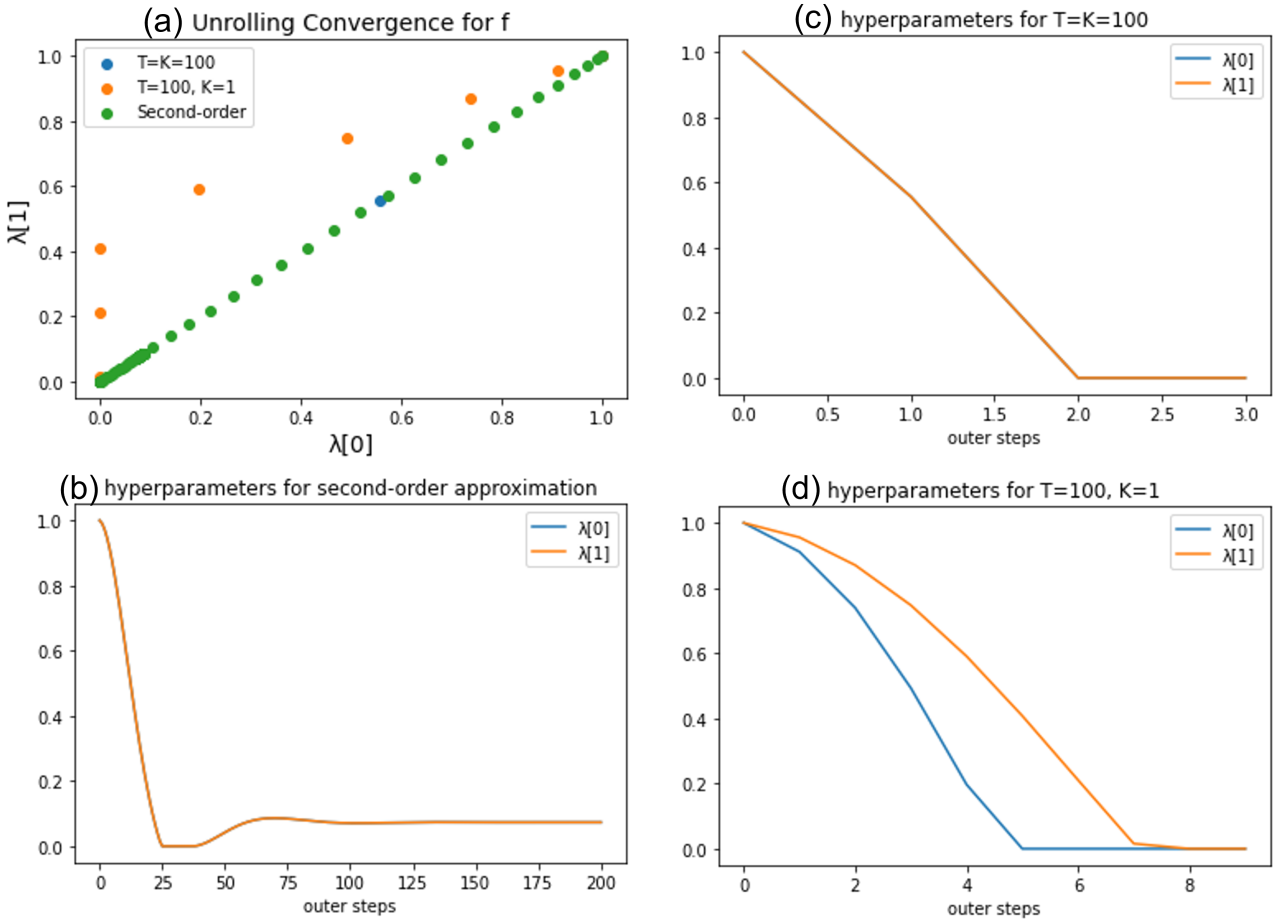


Fig. 19. (a) λ convergence for f with inner learning rate and ξ being 0.13. λ convergence using (b) second-order approximation, (c) full backpropagation ($T = K = 100$) and (d) truncated backpropagation ($T = 100, K = 1$) with respect to the number of outer steps.

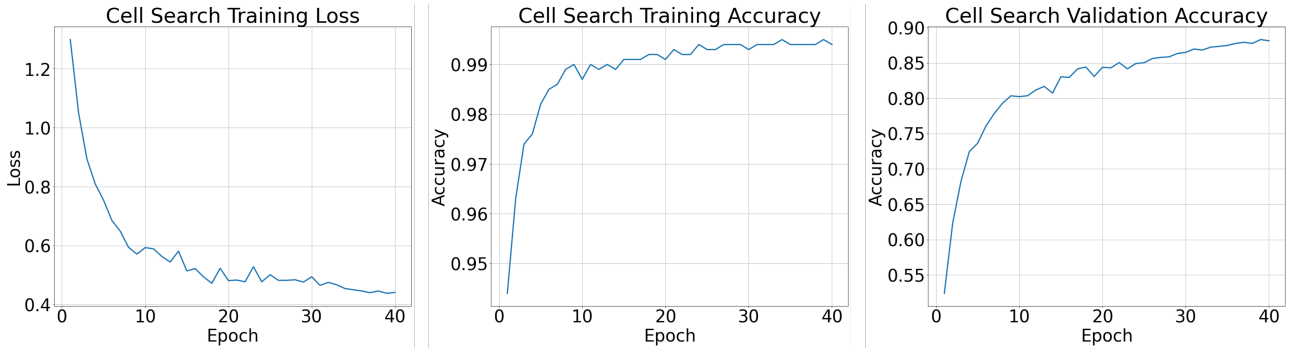


Fig. 20. Training loss, training accuracy and validation accuracy of the micro-architecture (small network) using DARTS on CIFAR-10 during cell search.

the searched cells are dependant on the random seeds used. In addition, the original DARTS paper trains the micro-architecture for 4 times and shows only the cells searched from the best-performing attempt in terms of validation accuracy. In contrast, only 1 cell search is performed in the experiment for preliminary results and it certainly does not perform as good as the best-performing cells from the original DARTS paper proving by the validation accuracy deviation.

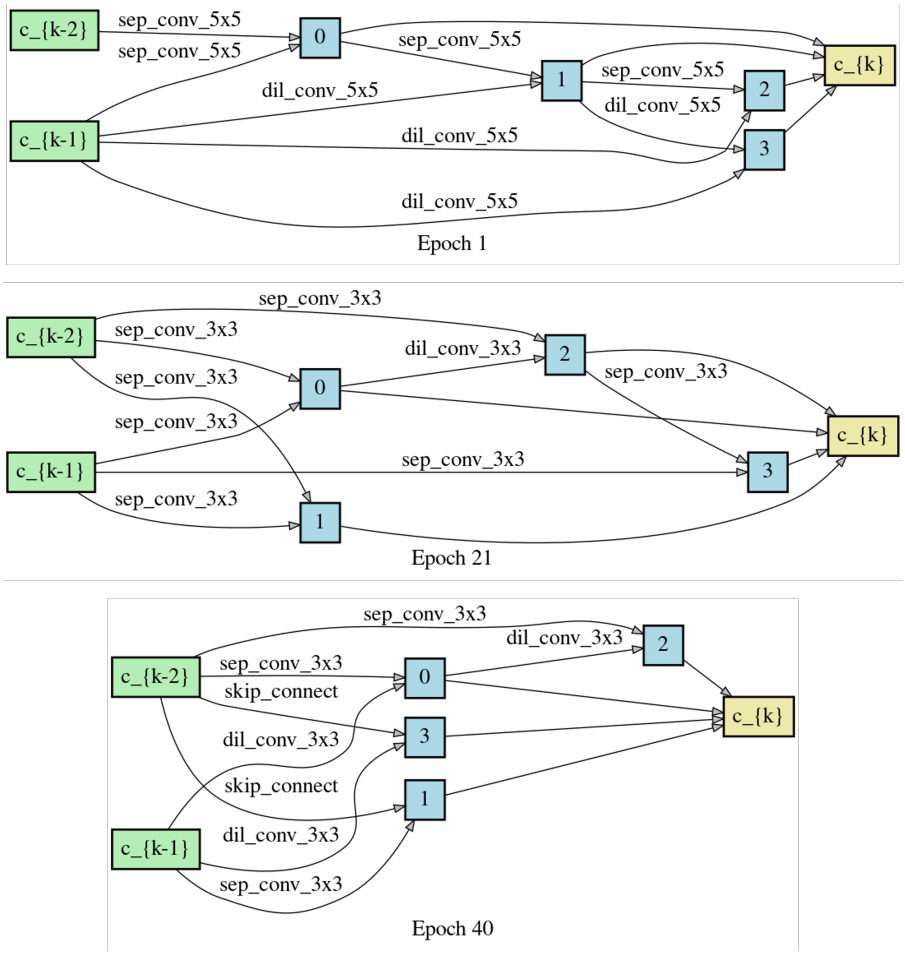


Fig. 21. Searched normal cell after 1st (top), 21th (middle) and 40th (bottom) epoch.

6.1.3 Self-Tuning Networks

The hyperparameter optimization capability of STNs is tested on two datasets: MNIST and FashionMNIST. The former is a dataset of handwritten digits while the latter is a dataset of Zalando's (a online retailer of beauty, fashion, and shoes) article images. Both has 10 classes for image classification.

For MNIST, a multilayer perceptron (MLP) with 2 hidden layers is trained. Each hidden layer is a fully-connected layer following by Rectified Linear Unit (ReLU) activation. The hyperparameters to be optimized by the STN is the 3 dropout rates applied to the input and the per-layer activations (output of each hidden layer). For FashionMNIST, a CNN with 2 convolutional layers with 32 and 64 kernels and 2 fully-connected layers is trained. There are 6 hyperparameters to be optimized by the STN: input dropout, per-layer dropouts and Cutout holes and length [113]. The results are shown in Fig. 24.

Overall, STNs are able to perform low-dimensional hyperparameter optimization resulting in high accuracies. Further analysis requires further experiments which is discussed in Section 6.2.

6.2 Limitations and Future Work

6.2.1 Limitations of DARTS

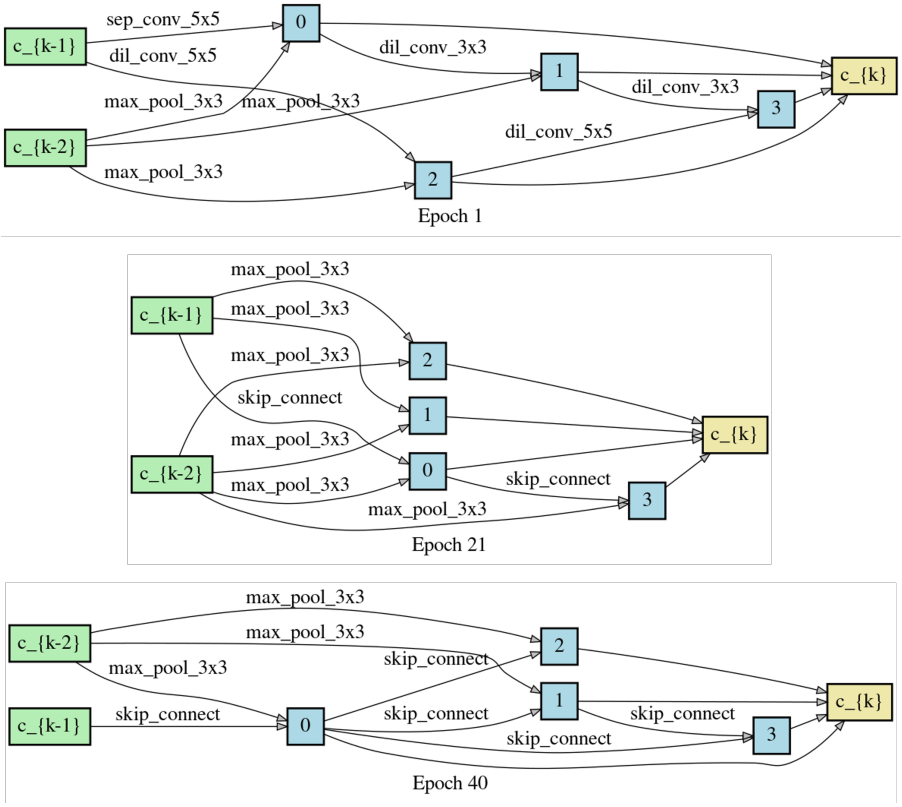


Fig. 22. Searched reduction cell after 1st (top), 21th (middle) and 40th (bottom) epoch.

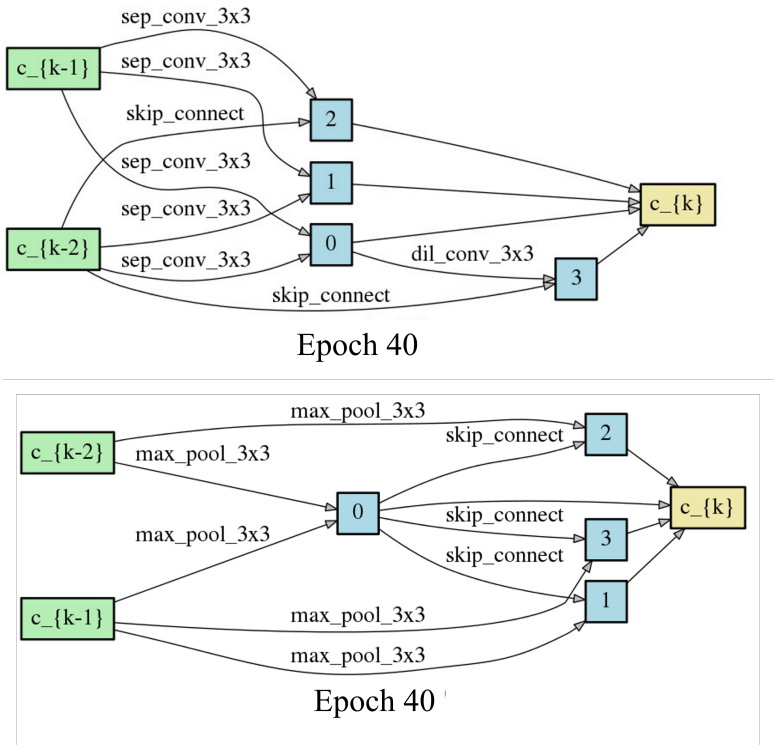


Fig. 23. The best-performing searched normal (top) and reduction (bottom) cell after 40th epoch from the original DARTS paper [26].

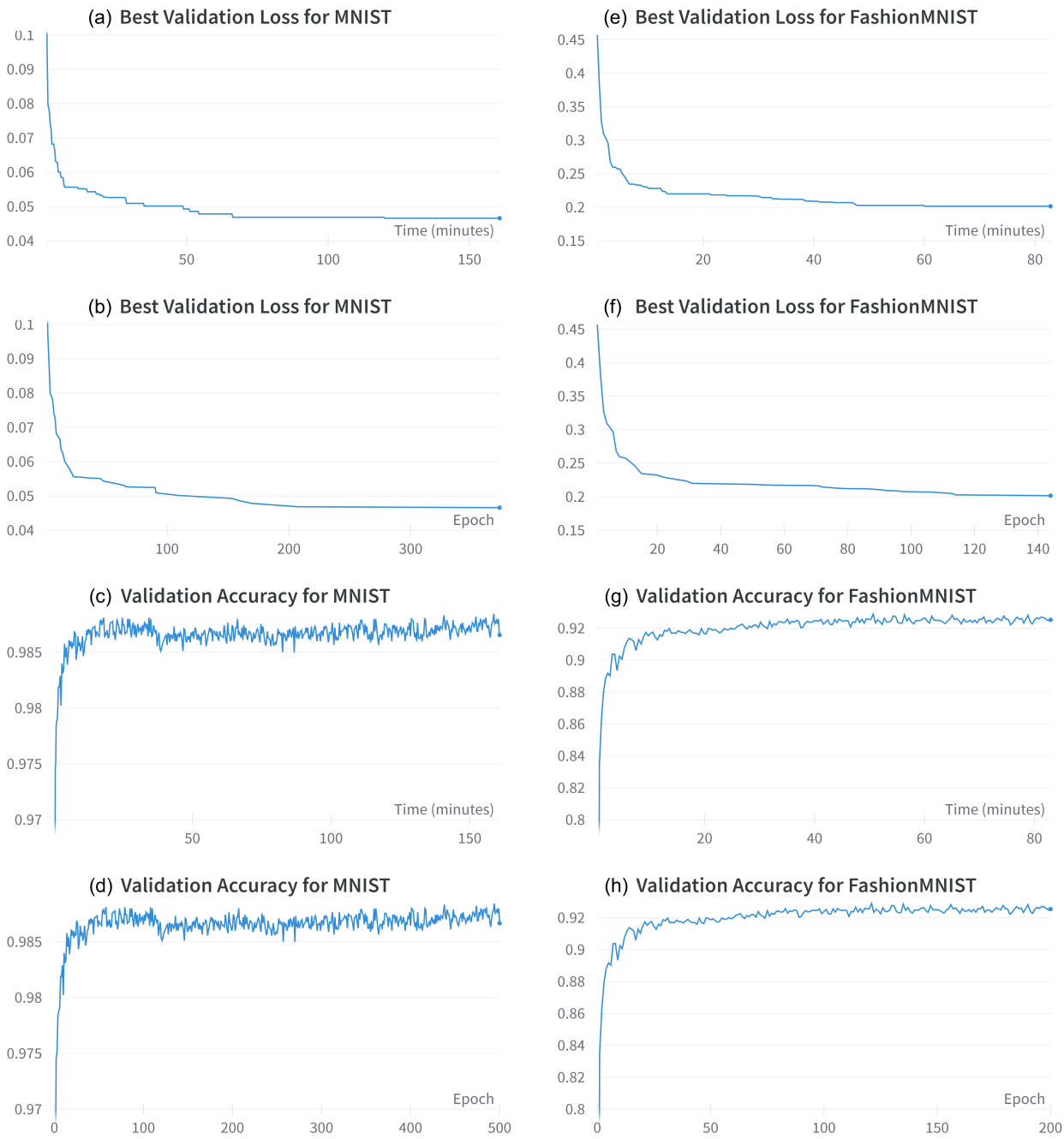


Fig. 24. Best validation loss for MNIST with respect to (a) time taken and (b) number of epochs using STNs. Validation accuracy for MNIST with respect to (c) time taken and (d) number of epochs. Best validation loss for FashionMNIST with respect to (e) time taken and (f) number of epochs. Validation accuracy for FashionMNIST with respect to (g) time taken and (h) number of epochs using STNs.

6.2.1.1 Performance Inferiority

DARTS is the pioneer and has a powerful influence in the field of differentiable architecture search. However, its performance for image classification in terms of accuracy and computational resources has been surpassed by many other later gradient-based and non-gradient-based NAS algorithms. For numeric reference, DARTS has a 73.1% top-1 accuracy on ImageNet, with the search cost being 4 GPU days.

Among the better-performed algorithms, the gradient-based NAS algorithms include GDAS, which samples one sub-graph instead of the entire DAG in one training iteration [81], and PC-DARTS that performs operation search only in a subset of channels [50]. On the other hand, the non-gradient-based NAS algorithms include EA-based CARS-I [114], zero-shot based Zen-NAS [115], training-free TE-NAS [116], and semi-supervised SemiNAS [117]. All these methods achieve better test error rates while consuming similar or several times lower GPU days compared to DARTS.

6.2.1.2 Gradient-based NAS Algorithms

The gradient-based NAS algorithms that perform better usually still employ hyperparameter optimization methods of similar performance compared to unrolling used in DARTS. Therefore, STNs can be applied to these models in an attempt to achieve the best performance among the gradient-based NAS algorithms. Nevertheless, this can be done in this project only if DARTS-STN does perform better than DARTS and there is sufficient time after integrating DARTS-STN.

6.2.1.3 Non-Gradient-based NAS Algorithms

However, the performance superiority achieved by the non-gradient-based NAS algorithms 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 [115] while keeping the search cost below 4 GPU days. For reference, 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. [53] 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.

6.2.1.4 Summary

As mentioned in Section 5.1, the improvement of other parts besides the bilevel optimization (functionally identical to hyperparameter optimization or hypergradient approximation in this project) element of DARTS is out of the scope of the project. Therefore, the primary product of this project: DARTS-STN, should have a better performance than DARTS but has a high chance of inferior performance compared to other more modern gradient-based and non-gradient-based NAS algorithms for the general image classification tasks. STNs can be applied to the current best-performing gradient-based NAS algorithm in an attempt to reach the highest accuracy among the gradient-based NAS algorithms, provided there is sufficient time in this project. However, integrating STN into any gradient-based NAS algorithm has a low chance of outperforming the latest non-gradient-based NAS algorithms due to the inherent weight-sharing issue lying in gradient-based NAS algorithms.

6.2.2 Limitations of STNs

STN is, in fact, not the most promising hypernetwork-based hyperparameter optimization method to the author's knowledge. There is an improved variant of STN known as Δ-STN [118] that uses a centered parameterization of the hypernetwork with a modified training objective. Δ-STNs perform better than STNs in terms of accuracy and search cost in all the experiments conducted in the paper [118].

There are two reasons for proposing STNs instead of Δ-STNs that perform better as the hyperparameter optimization method used in this project. First, the author is not familiar with the field of hyperparameter optimization (or even any optimization) before starting this project. Due to this project's short time frame and the author's high workload, the simpler version of the promising hyperparameter optimization methods is preferred for learning and implementation. Nevertheless, Δ-STN can be implemented in this project as an additional improvement, provided there is sufficient time. Second, Δ-STN is an STN variant instead of a whole-new hypernetwork-based hyperparameter optimization method. In other words, STN is the foundation of Δ-STN. For the author to build a solid knowledge foundation in this field, it is always advisable to start with the foundational model rather than its variant.

6.2.3 Future Work

6.2.3.1 Minimum Requirements - DARTS Phase

The performance evaluation of STNs has not been completed. The preliminary results of implementing STNs to solve some hyperparameter optimization problems are shown in Section 6.1.3. However, the efficiency and optimality of STNs can only be evaluated by applying other hyperparameter optimization methods, such as RS and BO, to the same problems for time and accuracy comparison. The experiment was tested on dropout rates, but unrolling cannot be performed to optimize hyperparameters that result in discrete changes such as dropout rates [99]. Therefore, another experiment has to be designed for time and accuracy comparison between STNs and unrolling. In addition, STNs should also be tested on high-dimensional hyperparameter optimization problems to validate their scalability, as NAS problems tend to have high-dimensional *lambda*.

Following the project plan and timeline in Section 5.2, after validating the performance superiority of STNs over other common hyperparameter optimization methods, the author should start to integrate STN into DARTS in code. First, the unrolling code of DARTS has to be replaced by that of STN. The integrated DARTS-STN will then be trained and tested on datasets chosen as discussed in Section 4.1. Meanwhile, DARTS should also be trained and tested on the same datasets for later performance comparison between DARTS and DARTS-STN. DARTS-STN is expected to perform better in terms of both accuracy and search cost as tested in other hyperparameter optimization problems [104, 105, 118].

The performance comparison between DARTS and DARTS-STN, which must have a better performance on all datasets chosen, is the minimum requirement to finish this project. Assuming the worst-case scenario of the author's productivity (inconsistent with the normal project progress assumed in Section 5.2), the performance evaluation of STNs has to be finished in 3 weeks after starting FYP B, and the integration coding should take less than 6 weeks. The remaining weeks are left for running code on all the datasets and performance evaluation. Provided there is no great deviation in model performance from expectations, the minimum requirements should be able to be met.

6.2.3.2 Project Completeness - RM2 Phase

Due to the limitations of DARTS as discussed in Section 6.2.1, DARTS-STN cannot be the final milestone to achieve the best image recognition performance among all gradient-based NAS algorithms. Provided the minimum requirements can be met in the first-half period of FYP B, an attempt to integrate STN into the best-performing gradient-based NAS algorithm (referred to as RM2 in this report) should be performed. This should be started during the training and testing period of DARTS-STN after the DARTS-STN coding is done.

The third quarter of FYP B should be used to code RM2-STN, while the last quarter should be used for performance evaluation between RM2 and RM2-STN. This RM2 phase can only start if DARTS-STN coding can be completed on time, i.e., by the end of Week 6. Due to the general difficulties of concept and code understanding, coding, and debugging, it is not very likely that DARTS-STN coding can be completed in the midst of FYP B. As a result, there is a chance of starting the RM2 phase but the chance of completing the RM2 phase with satisfactory results is arguable.

7 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 II.

TABLE II
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.

8 Sustainability Plan

8.1 Sustainability Engineering

8.1.1 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" [119]. Seager et al. [120] 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 [121] which can only be tackled by solutions with anticipation, adaptation, and resilience [120]. 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.

8.1.2 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 [122] 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 [123].

8.1.3 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 [124].

8.1.4 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 [120]: (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.

8.1.5 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.

8.2 Assessment and Analysis of Sustainability Performance of the Project

The project's sustainability performance is assessed following the metrics proposed by [125]: quality attributes (development-, usage-, and process-related attributes) and economic, social, and environmental benefits.

8.2.1 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.

8.2.2 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 [17, 18], 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 [123].

9 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] 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.

[11] 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.

[12] 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.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

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

[15] 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

[16] 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

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

[18] 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

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

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

[21] 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

[22] 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

[23] 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.

[24] 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.

[25] 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.

[26] 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

[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] 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

[34] 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.

[35] 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.

[36] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. L. Yuille, and L. Fei-Fei, "Auto-Deeplab: Hierarchical neural architecture search for semantic image segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 82–92.

[37] Y. Zhang, Z. Qiu, J. Liu, T. Yao, D. Liu, and T. Mei, "Customizable architecture search for semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[38] V. Nekrasov, H. Chen, C. Shen, and I. Reid, "Fast neural architecture search of compact semantic segmentation models via auxiliary cells," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[39] 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.

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

[41] 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

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

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

[44] 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

[45] 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.

[46] 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

[47] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, "A comprehensive survey on transfer learning," *PROCEEDINGS OF THE IEEE*, vol. 109, no. 1, pp. 43–76, JAN 2021.

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

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

[50] 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.

[51] 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

[52] 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

[53] 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

[54] S. Li, W. Li, S. Wen, K. Shi, Y. Yang, P. Zhou, and T. Huang, "Auto-fernet: A facial expression recognition network with architecture search," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2213–2222, 2021.

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

[56] 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

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

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

[59] 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

[60] 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

[61] 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

[62] 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.

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

[64] 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.

[65] 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.

[66] 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.

[67] 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

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

[69] 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

[70] 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

[71] 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.

[72] 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-

[73] 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

[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] 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.

[81] 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.

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

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

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

[108] I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville, M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, Y. Zhou, C. Ramaiah, F. Feng, R. Li, X. Wang, D. Athanasakis, J. Shawe-Taylor, M. Milakov, J. Park, R. Ionescu, M. Popescu, C. Grozea, J. Bergstra, J. Xie, L. Romaszko, B. Xu, Z. Chuang, and Y. Bengio, "Challenges in representation learning: A report on three machine learning contests," in *Neural Information Processing*, M. Lee, A. Hirose, Z.-G. Hou, and R. M. Kil, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 117–124.

[109] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*, 2010, pp. 94–101.

[110] M. Lyons, M. Kamachi, and J. Gyoba, "The Japanese Female Facial Expression (JAFFE) Dataset," Apr. 1998, The images are provided at no cost for non- commercial scientific research only. If you agree to the conditions listed below, you may request access to download. [Online]. Available: https://doi.org/10.5281/zenodo.3451524

[111] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-VAE: Learning basic visual concepts with a constrained variational framework," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=Sy2fzU9gl

[112] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural networks," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 1613–1622.

[113] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[114] 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.

[115] 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.

[116] 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.

[117] 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.

[118] J. Bae and R. B. Grosse, "Delta-stn: Efficient bilevel optimization for neural networks using structured response jacobians," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21 725–21 737, 2020.

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

[120] 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.

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

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

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

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

[125] 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.

10 Appendix

10.1 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