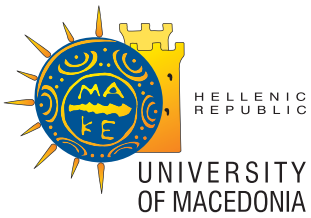


SCHOOL OF INFORMATION SCIENCES
DEPARTMENT OF APPLIED INFORMATICS
MSC IN ARTIFICIAL INTELLIGENCE AND DATA ANALYTICS

EVALUATING ACCELERATION TECHNIQUES FOR CANDIDATE EVALUATION IN GENETIC NEURAL ARCHITECTURE SEARCH

A dissertation

by

Foteini Dervisi

Thessaloniki, July 2022

EVALUATING ACCELERATION TECHNIQUES FOR CANDIDATE EVALUATION IN GENETIC NEURAL ARCHITECTURE SEARCH

Foteini Dervisi

Diploma (BSc & integrated MSc) in Electrical and Computer Engineering,
Aristotle University of Thessaloniki, 2019

Dissertation

Submitted in partial fulfilment of the requirements for
THE MSC IN ARTIFICIAL INTELLIGENCE AND DATA ANALYTICS

Supervisor Professor
Konstantinos Margaritis

Approved by the three-member Examination Committee on 04/07/2022

Konstantinos Margaritis	Ioannis Refanidis	Nikolaos Samaras
.....	.....	.....

Foteini Dervisi

.....

Evaluating Acceleration Techniques for Candidate Evaluation in Genetic Neural Architecture Search

Abstract

Deep learning has developed remarkably over the last few years due to the massive improvement of computational systems as well as the fact that the volume of available data for training deep neural networks has increased significantly. In order for deep learning techniques to succeed, appropriate neural network architectures need to be selected. The process of selecting a neural network architecture is typically performed by trial and error, which is a process that requires time, knowledge of the field and is likely to lead to errors when performed by humans. Furthermore, it may exclude architectures that have not yet been used for the desired task, but would have been capable of achieving great performance. The field of neural architecture search has therefore been developed as a means of tackling the aforementioned difficulties by automatically producing optimal neural network architectures. Neural architecture search requires the use of candidate evaluation techniques in order to assess the quality of the architectures that are generated during the search procedure and thus guide the algorithm to select the best architectures within the chosen search space. As the process of candidate evaluation is usually time-consuming and requires access to adequate computational power, there is increasing interest in reducing the time and computational requirements of candidate evaluation. This thesis focuses on the evaluation of acceleration techniques for evolutionary neural architecture search. The overall aim of this thesis is to examine the use of various acceleration techniques for candidate evaluation and assess their impact on the search procedure and on the quality of the produced architectures.

Keywords: Neural Networks, Deep Learning, Neural Architecture Search, Candidate Evaluation, Evolutionary Algorithms

Περίληψη

Η βαθιά μηχανική μάθηση έχει γνωρίσει σημαντική ανάπτυξη τα τελευταία χρόνια λόγω της εξέλιξης των υπολογιστικών συστημάτων και του γεγονότος ότι ο όγκος των διαθέσιμων δεδομένων για εκπαίδευση των βαθιών νευρωνικών δικτύων έχει αυξηθεί σημαντικά. Προκειμένου οι τεχνικές βαθιάς μηχανικής μάθησης να επιτύχουν τους στόχους τους και να κατορθώσουν να λύσουν τα προβλήματα για τα οποία έχουν σχεδιαστεί, απαιτείται η επιλογή κατάλληλων αρχιτεκτονικών νευρωνικών δικτύων. Η επιλογή κατάλληλων αρχιτεκτονικών νευρωνικών δικτύων είναι χρονοβόρα διαδικασία και πραγματοποιείται συνήθως μέσα από μία διαδικασία δοκιμής και πλάνης η οποία απαιτεί γνώση του πεδίου εφαρμογής. Επίσης, μια τέτοια διαδικάσια είναι πιθανό να οδηγήσει σε λάθη καθώς και να αποκλείσει αρχιτεκτονικές που δεν είχαν έως τώρα χρησιμοποιηθεί για κάποιο συγκεκριμένο πρόβλημα, όμως έχουν τη δυνατότητα να ανταποκριθούν σε αυτό και να το επιλύσουν αποτελεσματικά. Για το λόγο αυτό αναπτύχθηκε το ερευνητικό πεδίο της αυτόματης αναζήτησης αρχιτεκτονικών νευρωνικών δικτύων, το οποίο στοχεύει στην ανάπτυξη τεχνικών για την ανακάλυψη βέλτιστων αρχιτεκτονικών νευρωνικών δικτύων. Οι τεχνικές αναζήτησης αρχιτεκτονικών νευρωνικών δικτύων απαιτούν τη χρήση κάποιων μεθόδων για την αξιολόγηση των παραγόμενων αρχιτεκτονικών κατά τη διάρκεια της διαδικασίας της αναζήτησης προκειμένου ο μηχανισμός αναζήτησης να καθοδηγηθεί προς τις αρχιτεκτονικές με την καλύτερη συμπεριφορά εντός του χώρου αναζήτησης που έχει επιλεχθεί. Η διαδικασία της αξιολόγησης των παραγόμενων αρχιτεκτονικών χρειάζεται συνήθως αρκετό χρόνο και υπολογιστικούς πόρους, καθώς βασίζεται στην εκπαίδευση των παραγόμενων αρχιτεκτονικών πάνω σε ένα προκαθορισμένο σύνολο δεδομένων. Προκειμένου η διαδικασία της αναζήτησης να επιταχυνθεί, η ερευνητική κοινότητα έχει στραφεί προς τη χρήση τεχνικών για την επιτάχυνση της αξιολόγησης των παραγόμενων αρχιτεκτονικών. Η διπλωματική αυτή εστιάζει στη χρήση τεχνικών επιτάχυνσης της αξιολόγησης των αρχιτεκτονικών που παράγονται κατά τη διαδικασία της αυτόματης αναζήτησης και έχει ως στόχο την αξιολόγηση διάφορων τεχνικών επιτάχυνσης και την εκτίμηση του αντικτύπου που έχουν αυτές στη διαδικασία της αναζήτησης και στην ποιότητα των παραγόμενων αρχιτεκτονικών.

Λέξεις Κλειδιά: Νευρωνικά δίκτυα, Αναζήτηση αρχιτεκτονικών νευρωνικών δικτύων, Εξελικτικοί αλγόριθμοι, Αξιολόγηση υποψήφιων αρχιτεκτονικών

Acknowledgements

I would like to express my gratitude to George Kyriakides, a PhD candidate at the Department of Applied Informatics of the University of Macedonia, whose guidance and support have been invaluable for the completion of this thesis. I would also like to thank Professor Konstantinos Margaritis, my thesis supervisor, for giving me the opportunity to work on such an interesting project as well as for his patience, encouragement and valuable feedback.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning is a subfield of machine learning that employs multi-layer artificial neural networks in order to automate feature engineering and produce results based on perceiving the world as a hierarchy of concepts and attempting to learn complex concepts with the use of many simpler concepts that are combined to produce the final results [2]. The recent increase in the available data and computational power has led to the rapid growth of deep learning techniques, which have led to important breakthroughs in fields such as computer vision and natural language processing. For example, deep learning models have significantly contributed to the solution of problems such as image classification, object detection and face recognition, as well as problems such as machine translation, question answering and speech recognition.

As the success of deep learning methods is dependent on the neural network architecture that is chosen to tackle the task at hand, neural architecture search has emerged as a subfield of automated machine learning in order to enable researchers and machine learning practitioners to automatically discover optimal neural network architectures. Neural architecture search aims to discover optimal neural network architectures within a chosen search space. It employs optimisation methods such as evolutionary algorithms, reinforcement learning and Bayesian optimisation in order to explore the search space and uses candidate evaluation to assess the quality of the produced architectures and guide the search towards producing high-performing neural network architectures. Candidate evaluation is typically performed by training and evaluating each candidate architecture on a specific dataset, which is a time-consuming process that also requires access to adequate computational resources.

This thesis focuses on accelerating the candidate evaluation step of neural architecture search processes. It employs different techniques for the acceleration of candidate evaluation and aims to assess their impact on the neural architecture search procedure as well as on the quality of the discovered neural network architectures. The optimisation algorithms that have been explored are a simple genetic algorithm and a genetic algorithm with the use of fitness approximation. Two different approximation models have been

considered: an XGBoost model and a graph convolutional network model. Furthermore, different options regarding the fitness measure have been explored. The standard fitness measure in neural architecture search procedures is the validation accuracy of the trained candidate architectures. However, in order to reduce the time needed for the search procedure, an alternative fitness measure that calculates the fitness of individuals based on their initial state has been considered. This measure is called "Neural Architecture Search Without Training" score and has been proposed in order to cheaply estimate a network's trained accuracy without requiring any training.

The structure of this thesis is the following:

- Chapter 2 contains a theoretical background on artificial neural networks and neural architecture search.
- Chapter 3 introduces the reader to evolutionary algorithms and highlights their contribution in neural architecture search.
- Chapter 4 expands on the candidate evaluation step of the neural architecture search procedure and presents the acceleration methods that have been explored in the experiments that were conducted as part of this thesis.
- Chapter 5 presents the experiments that were performed and describes all the details related to the experimental setup.
- Chapter 6 presents the results of the experiments and draws conclusions as to the efficiency of the examined acceleration methods.
- Chapter 7 contains a summary and overall conclusion and also mentions potential future directions that are in line with the research conducted within this thesis.
- The accompanying code can be found in https://github.com/foteinidd/genetic_nas_acceleration.

Chapter 2

Artificial Neural Networks

Artificial neural networks are mathematical functions that resemble the function of the human brain in that they perform non-linear calculations with the use of small individual units that are called neurons. An artificial neural networks consists of layers, each of which consists of a large number of neurons, as well as weighted connections between neurons of different layers.

2.1 Neurons

Neurons are the basic building blocks of artificial neural networks and they are used in order to build neural network layers. Neurons are connected to neurons of different layers and each connection is assigned a weight. The weight between two neurons can be either positive or negative, according to the influence the one neuron has on the other. High weights between neurons signify that these two neurons have a great influence on one another. Each neuron receives a number of inputs (according to the neurons of previous layers to which it is connected) and a bias value. Its purpose is to calculate the weighted sum of the inputs and add the bias term to the result. The calculated term is then passed through a non-linear activation function, whose function is to take the linear output that has been calculated and perform a non-linear operation on it in order to perform regression or classification. The use of non-linear activations is invaluable for the solution of non-linearly separable problems. Common activation functions are listed in table 2.1.

2.2 Layer Types

There are different types of neural network layers, each of which consists of a set of neurons. In this section, some of the most common neural network layer types will be presented.

Table 2.1: Activation Functions

Activation function	Mathematical expression
Linear	$f(x) = x$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Hyberbolic tangent (tanh)	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
Rectified linear unit (ReLU)	$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
Leaky rectified linear unit (Leaky ReLU)	$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01x & \text{otherwise} \end{cases}$
Softplus	$f(x) = \ln(1 + e^{x})$
Softmax	$f(x) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$, $i = 1, 2, ..., N$

2.2.1 Fully Connected Layers

In fully connected layers, all neurons in one layer are connected to all neurons in the next layer. Fully connected layers may be used in all types of neural networks and they are the standard choice for neural network layers. However, the use of fully connected layers comes at a cost, as the computational power needed to train networks with fully connected layers is considerable, due to the fact that a large number of computations needs to be performed because each neuron is connected to all neurons in the previous as well as in the next layer. An example of a network with fully connected layers can be seen in figure 2.1. The edges have been coloured according to the edge weight values: positive weights are depicted in red, whereas negative weights are depicted in blue. Furthermore, the opacity of the edges is proportional to their weights.

2.2.2 Dropout Layers

Regularisation is used in order to tackle the issue of overfitting on the training data, which limits the generalisation ability of trained neural network models. A common regularisation approach is including dropout layers in the network. Dropout layers deactivate neurons with a probability equal to p during the training process. The value of p is non-trainable and is decided by the neural network architecture designer [3]. The use of dropout can be perceived as an ensemble learning technique in that as some neurons are randomly deactivated in each step of the training process, a set of sub-networks is trained in order to produce the final trained model.

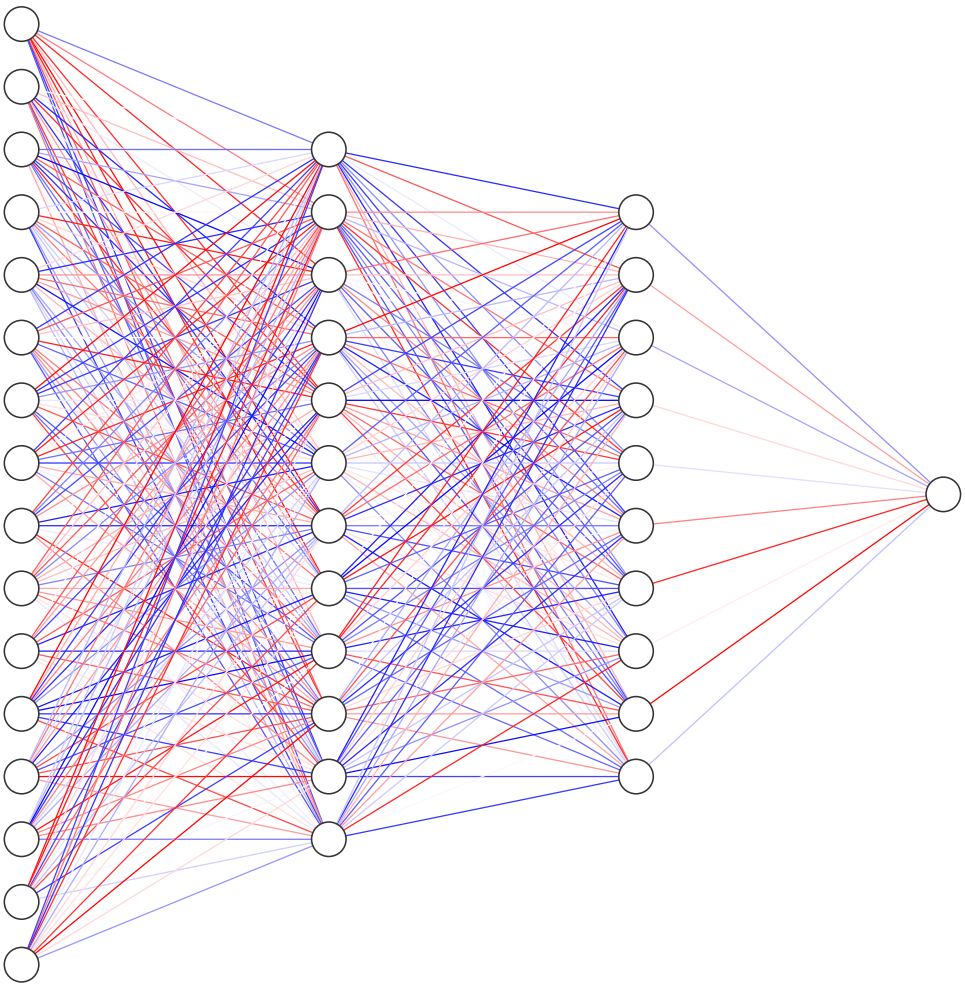


Figure 2.1: Fully Connected Neural Network

2.2.3 Convolutional Layers

Convolutional layers are suitable for the extraction of spatial information, which is why they are widely used in cases where feature extraction from images is required. These layers consist of filters that extract feature maps to summarise the discovered features. Filters are essentially $n \times n$ matrices that are multiplied element-wise with the layer's inputs. The elements of the matrix that occurs are then aggregated in order to provide a single number. Subsequently, the filter is moved m positions to the right of the layer's inputs and the same procedure is performed. The values of n and m are specified by the architecture designer [4]. Convolutional layers are the basic building blocks of convolutional neural networks (CNNs), which are particularly successful at performing image processing tasks. An example of a convolutional neural network can be seen in figure 2.2.

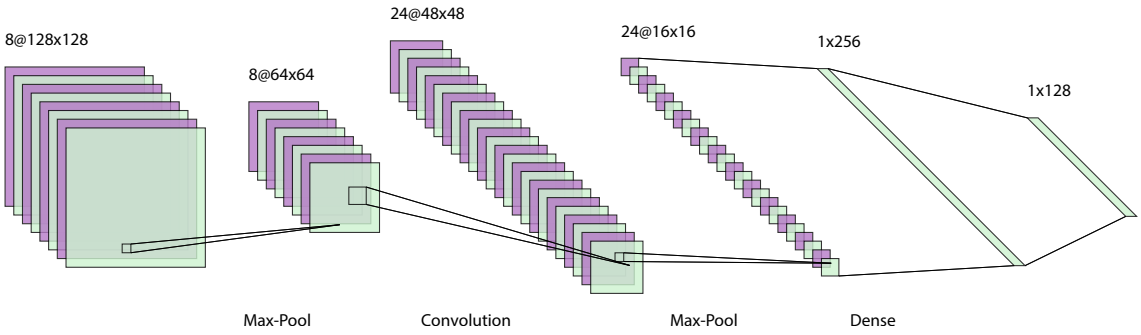


Figure 2.2: Convolutional Neural Network

2.2.4 Pooling Layers

Pooling layers are typically used after convolutional layers in convolutional neural networks and their purpose is to downsample the produced feature maps. They operate by aggregating input patches of a fixed size in order to produce an output with reduced dimensions. Common aggregation methods include averaging over the numbers in each patch (average pooling) and selecting the maximum element in each patch (max pooling).

2.2.5 Recurrent Layers

Recurrent layers are suitable for processing sequential data such as time series and text processing, as they operate recursively. They can be used to process the input data in such a way so as to provide an output whose type and dimension is different than the input's. These layers leverage the concept of memory, as they take both the data and the previous calculation's output as input in order to be able to remember the previous part of the sequence [5]. They are the building blocks of recurrent neural networks (RNNs), which have led to significant advancements in the field of sequence processing. Common types of recurrent neural networks include Long Short-Term Memory (LSTM) networks [6] and Gated Recurrent Unit (GRU) networks [7].

2.3 Neural Networks

Artificial neural networks consist of an input layer, one or more hidden layers and an output layer. Their purpose is to learn to find patterns in data, identify trends and make accurate predictions of the future. The learning process is performed by first performing a feed-forward operation, i.e. propagating the input data through the layers and producing an output and then optimising that output with the use of backpropagation, which is the process that is used during the learning phase of the algorithm in order to provide the network with feedback. This process consists of using a loss function in order to calculate

the difference between the produced output and the desired output and then using this difference in order to adjust the weights of the connections between neurons. The weight adjustment process is performed iteratively, starting from the output layer and going back layer by layer until the input layer is reached. The training process, which consists of the feedforward and backpropagation steps, is performed iteratively for a number of epochs, gradually leading to the minimisation of the difference between the network's output and the desired output. The weight optimisation process is performed with the use of an optimisation algorithm. Common optimisation algorithms that are used for weight optimisation while training neural networks include batch gradient descent, stochastic gradient descent (SGD), mini-batch gradient descent, stochastic gradient descent (SDG) with momentum, Nesterov accelerated gradient, Adaptive Moment Estimation (Adam), Adagrad, Adadelta and RMSProp [8].

2.4 Deep Neural Networks

Deep neural networks are artificial neural networks with more than one hidden layer. They contain a large number of neurons and are capable of automatically performing complicated tasks such as face recognition, speech synthesis and time series forecasting. The existence of multiple hidden layers enables the gradual discovery of information from the input data. The first few hidden layers extract low-level characteristics from the input data, whereas the next hidden layers try to infer high-level concepts that are useful for producing the final output. The field of deep learning has significantly evolved over the last few years, due to the increase of the available processing power, which has enabled breakthroughs in the field.

2.5 Neural Architecture Search

The task of selecting an appropriate neural network architecture for a given task is still a non-trivial problem that typically requires a time-consuming trial-and-error process, which is why attempts to automate this process have recently been made. The process of automating the design of neural network architectures is called neural architecture search. Neural architecture search is a subfield of automated machine learning that employs optimisation algorithms in order to automatically discover optimal neural network architectures within a given search space. It has led to the discovery of state-of-the-art architectures that are able to perform various tasks better than previous manually-designed architectures and use minimal computational resources [9–13]. The basic components of a neural architecture search algorithm are the search space, the optimisation method and the candidate evaluation method [14].

The search space defines the set of neural network architectures that are eligible for selection by the algorithm. The choice of search space is usually made according to prior experience, which however may introduce bias to the process, as in this case the architectures that will be considered during the search procedure may be similar to architectures that have previously been used to tackle problems similar to the one the designed architecture is intended to address, thus excluding different architectures that may have been capable of performing better [15]. The optimisation method provides the means to explore the search space and discover high-performing neural network architectures within it. Common choices of optimisation methods in neural architecture search processes include evolutionary algorithms, reinforcement learning and Bayesian optimisation, whereas some one-shot methods have also been proposed [16]. Finally, the candidate evaluation method is the process that is used for the evaluation of the candidate neural network architectures that come up as the algorithm is running. Candidate evaluation serves as a means of guiding the search towards discovering high-performing neural network architectures. The typical process that is followed for candidate evaluation is performing a standard training and validation procedure for each candidate architecture on a specific dataset and evaluating the architecture based on the performance statistics that are calculated during the validation procedure. However, this is a time-consuming process, as it requires training and evaluating a large number of architectures, which is why alternative approaches have been proposed to speed up this step of the neural architecture search process. Some approaches for the reduction of the time needed for candidate evaluation include reducing the number of training epochs [17], performing the training and validation procedure on less computationally demanding datasets and then transferring the results to the desired dataset, reducing the number of cells or filters used, as well as using predictive models to estimate the performance of candidate architectures and guide the search based on these predictions [18].

2.5.1 Benchmark Datasets

Neural architecture search benchmark datasets have been developed in order to provide a quick and easy way to test new ideas in the field and compare different approaches, as the process of candidate evaluation typically requires a time-consuming and computationally demanding training and validation procedure for each of the generated architectures, which leads to the inability of researchers without access to adequate computational resources to reproduce published results and perform novel experiments in the field. Benchmark datasets contain precomputed performance statistics of complete search spaces, thus allowing for the rapid evaluation and comparison of neural architecture search algorithms.

2.5.1.1 NAS-Bench-101

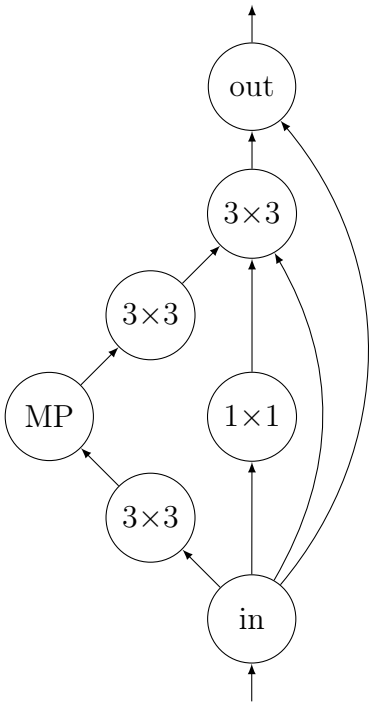
NAS-Bench-101 [19] is the first publicly released neural architecture search benchmark dataset. It consists of 423,624 convolutional neural network architectures that have been trained on the CIFAR-10 dataset and are intended to be used for the task of image classification. Each architecture has been trained with three different initialisations and the performance statistics have been computed after training each architecture for 4, 12, 36 and 108 epochs. Thus, the final dataset contains over 5 million architectures. NAS-Bench-101 architectures are built by defining a cell and using a predefined structure (figure 2.4) to construct the network's architecture by repeatedly stacking identical cells. Each architecture's cell contains a maximum of 7 layers including the input and output layer and a maximum of 9 connections between layers. The cell is depicted in the form of a directed acyclic graph, where nodes represent layers and edges represent the connections between them. The hidden layers are selected from the available operations, which are 3×3 convolution, 1×1 convolution and 3×3 max pooling. An example of a NAS-Bench-101 cell is displayed in figure 2.3a. The predefined skeleton of NAS-Bench-101 architectures consists of an initial 3×3 convolutional layer, three cell stacks (each of which consists of three identical cells) and a max pooling layer between each stack. The network's final layers are a global average pooling layer and a dense layer, which is used to calculate and output the final result. The best architecture in NAS-Bench-101 achieves a mean test accuracy of 94.32% on the CIFAR-10 dataset.

2.5.1.2 NAS-Bench-201

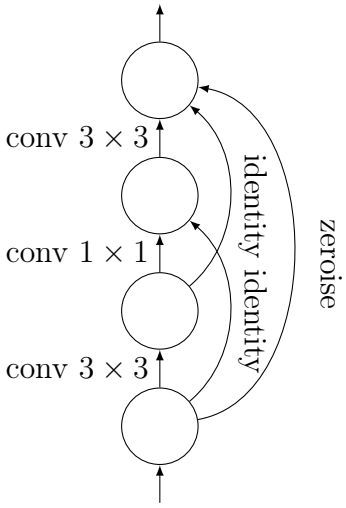
NAS-Bench-201 [20] also contains convolutional neural network architectures that are constructed with the use of a predefined skeleton of multiple stacked cells. The total number of architectures within the NAS-Bench-201 search space is 15,625. The architectures it contains are trained on the CIFAR-10, CIFAR-10 and ImageNet-16-120 datasets, and a manual training, test and validation split is performed for each of the three datasets. The architectures are trained once by using the training set for training and the validation set for evaluation and once by using the training and validation sets for training and the test set for evaluation. The skeleton is the same that is used by NAS-Bench-101 (figure 2.4), with the only difference being the number of cells in each stack. The neural architecture search process defines the architecture of the cell, which is depicted with the use of a directed acyclic graph with 4 nodes and 6 edges, as each node is connected to all subsequent layers. The edges here represent the operations, which are chosen from 5 possible types of layers: 1×1 convolution, 3×3 convolution, 3×3 average pooling, zeroise and identity. An example of a NAS-Bench-201 cell can be seen in figure 2.3b.

2.5.1.3 NATS-Bench

NATS-Bench [21] architectures are also constructed with the same predefined skeleton (figure 2.4) used by NAS-Bench-101 and NAS-Bench-201. NATS-Bench contains two distinct search spaces, a topology search space (TSS) and a size search space (SSS). The topology search space is identical to the search space of NAS-Bench-201 (i.e. the different networks in the search space use different types of operations), whereas the size search space offers different choices regarding the number of channels in convolutional layers but consists of networks with identical types of operations, which are selected by exploring the topology search space and identifying the architecture that performs best on CIFAR-100. The possible numbers of channels in the size search space are 8, 16, 24, 32, 40, 48, 56 and 64. Thus, the overall size of the topology search space is $15,625$ networks, whereas the size of the size search space is $32,768$ networks.



(a) NAS-Bench-101 Cell Example



(b) NAS-Bench-201/NATS-Bench Cell Example

Figure 2.3: NAS Benchmark Cell Examples

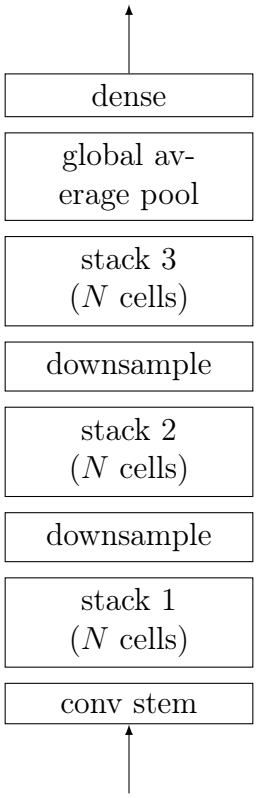


Figure 2.4: NAS-Bench-101 ($N = 3$)/NAS-Bench-201 ($N = 5$)/NATS-Bench ($N = 5$) Skeleton

Chapter 3

Evolutionary Algorithms

Evolutionary algorithms are population-based metaheuristics inspired by natural evolution. They use mechanisms such as mutation, crossover, selection, inversion and reproduction with an aim to discover high-quality solutions to optimisation problems. These algorithms use an iterative procedure where in each iteration a population consisting of the candidate solutions to the given optimisation problem is built and a fitness function is employed to determine each candidate solution's quality. The final solution is discovered at the end of the iterative procedure, which resembles the natural selection process in that high-quality solutions are favoured and retained as the evolutionary algorithm is running whereas solutions of inferior quality are discarded.

3.1 Surrogate-Assisted Evolutionary Algorithms

The evolutionary process succeeds in finding appropriate solutions to a large range of optimisation problems, however the time and computational power needed for the algorithm to run can be considerable, mainly due to the fact that the calculation of the fitness of each candidate solution is required. Due to the fact that the evaluation of the fitness function for each individual in the population can be time-consuming and complex, fitness approximation models have been introduced. Fitness approximation models attempt to overcome the problem of the computational complexity of fitness function evaluation by employing predictive models estimate the quality of candidate solutions and guide the evolutionary algorithm's search. These predictive models are called surrogate models or meta-models, whereas when such models are employed, the evolutionary procedure is called surrogate-assisted evolutionary algorithm [22]. The fitness approximation models are usually machine learning based. Typical models that are used for this purpose include regression models, support vector machines (SVMs) as well as different types of neural networks.

3.2 Genetic Algorithms

Genetic algorithms are a type of evolutionary algorithms where the initial population of randomly generated candidate solutions is gradually evolved to lead to the discovery of high-quality solutions with the use of an iterative process of multiple generations. The parameters that need to be defined in order for the genetic algorithm to operate are the representation of individuals and the fitness function, which is typically the objective function of the optimisation problem the algorithm is intended to tackle. The process that is performed in each generation of the algorithm is the following: First, each candidate solution in the current population is evaluated with the use of the fitness function. Once all individuals have been assessed, a stochastic process is employed with an ultimate aim to retain high-quality solutions in the population and discard low-quality ones. Typical selection strategies include:

- **random selection**,
- **tournament selection**, which is the process of selecting the best individuals from randomly sampled subsets of the population,
- **linear ranking selection**, which involves ranking the candidate solutions in descending order based on their fitness and then assigning each candidate a selection probability based on its rank and
- **roulette wheel selection**, where the probability of selection of each candidate is proportional to its fitness, thus making it more probable for the higher-performing candidates to be chosen.

These individuals are then subjected to a procedure of modification, which involves operations such as mutation or crossover, in order to generate a novel population with characteristics that are similar to the ones of the best solutions found up until that point. The new population is then selected. Typical strategies for the selection of the new population are:

- **replace oldest**: the replacement of all parents with the new offsprings that have been generated,
- **survival of the best individuals**: performing a ranking operation on all parents and offsprings based on the fitness measure and retaining the N individuals with the best performance,
- **replace worst**: using offsprings to replace the worst performing individuals,
- **replace random**: randomly replacing some of the parents with generated offsprings.

The iterative process is terminated when one of the following conditions is satisfied:

- the predefined **maximum number of iterations** has been reached,
- the algorithm has succeeded in discovering individuals whose fitness is deemed satisfactory, meaning that some of the individuals that have been produced offer a **satisfactory solution** to the optimisation problem at hand,
- **the best individual has remained the same** for a number of iterations,
- **no member of the population has changed** over the last iteration.

3.3 Evolutionary Algorithms in Neural Architecture Search

Evolutionary algorithms are a common choice for the optimisation strategy of neural architecture search procedures, meaning that they are used as a means of exploring the chosen search space. In the context of neural architecture search, evolutionary algorithms can help with selecting layer types, parameters and connections between layers [23]. Search space exploration with the use of evolutionary algorithms is performed as follows: First, a number of architectures are randomly sampled from the search space to create the initial population. Then, each architecture in the population is evaluated with the help of the fitness function. Fitness measures are usually the networks' performance statistics that are relevant to the problem that the produced neural network architecture is intended to solve. For example, a typical fitness measure is the validation accuracy in cases where the network is intended to solve a classification problem, whereas in cases where the network is designed to tackle regression problems a metric such as the mean squared error (MSE) or the mean absolute error (MAE) on the validation data may be used as a fitness measure of candidate solutions. Once all solutions have been evaluated, a stochastic strategy is employed for the selection of the best architectures and these architectures are slightly altered with the use of an operation such as mutation or crossover. Common modifications include changing the layer types or the connections between layers with a predefined probability. The new population is then formed and the iterative procedure continues until the selected termination criterion is satisfied.

3.3.1 Evolutionary Neural Architecture Search with Fitness Approximation

NAS-EA-FA V2 [24] is an evolutionary algorithm that can be used to reduce the time needed for the search procedure by using a fitness approximation model to estimate the

fitness of individual architectures instead of performing a training and validation procedure for each candidate architecture. This algorithm uses a selection scheme in order to train and evaluate a small number of architectures in each iteration, as the performance statistics of a number of architectures are required to provide training data for the fitness approximation model, and then trains the fitness approximation model, which provides an estimation of the fitness of the remaining architectures. The selection scheme works as follows: First, the architectures in the current population are sorted in descending order based on their fitness score. The algorithm then selects the K architectures with the best performance as well as the H architectures with the largest distance from all the previously trained architectures. The H architectures are included in the fitness approximation model's training set in order to enhance diversity by encouraging the algorithm to explore more areas of the search space, thus contributing to the improvement of the algorithm's stability. Moreover, in an attempt to enhance the trained fitness approximation model's predictive ability, the training dataset is augmented by including the isomorphic sequences of the selected $K + H$ architectures.

Chapter 4

Candidate Evaluation

Candidate evaluation is the process of evaluating the quality of candidate architectures during the neural architecture search procedure and is used in order to discover the best performing architectures based on a predefined criterion. A typical approach is evaluating candidates based on the candidate architectures' performance statistics, which are calculated after a full training and validation procedure. This is a computationally expensive process as it requires the training and validation of a large number of architectures.

However, the task that candidate evaluation methods should be able to accomplish is not the accurate calculation of each candidate architecture's absolute performance but rather the determination of each candidate's performance in comparison to other candidates. Based on this realisation, alternative approaches have been proposed to speed up the candidate evaluation process, such as training candidate networks for just a few epochs, which is an approach that has been shown to produce satisfactory results given that the difference in the number of epochs used for the candidate evaluation and the number of epochs the produced architectures are intended to be trained for is relatively small. Another approach considers the use of hyper-networks or network morphisms to provide a starting point for candidate evaluation. Furthermore, a transfer learning approach has been proposed, which performs candidate evaluation on a less computationally demanding dataset and then transferring the results to the intended dataset. Another proposed approach suggests the reduction of the number of filters or the number of cells that each building block of the architecture consists of. Apart from the aforementioned methods, fitness approximation models are becoming an increasingly popular approach for the acceleration of the candidate evaluation process in neural architecture search. This approach leverages surrogate predictive models to estimate the performance of candidate architectures. However, the training and evaluation of a small number of architectures is still required to provide training data for the surrogate model to be trained. Finally, another acceleration approach involves the use of weight-sharing amongst candidate neural network architectures.

4.1 Fitness Approximation

This thesis focuses on designing efficient neural architecture search algorithms while at the same time reducing the candidate evaluation step's time and computational cost. The particular aim of this thesis is to evaluate the use of acceleration techniques for candidate evaluation in neural architecture search. Fitness approximation is a popular acceleration method for candidate evaluation, as it allows for the reduction of the neural architecture search process by quickly providing estimations of the quality of candidate networks. Common choices for fitness approximation in neural architecture search algorithms include ensemble learning techniques, such as XGBoost, and graph convolutional networks.

4.1.1 Ensemble Learning

Ensemble learning is the process of combining multiple machine learning models in order to obtain an overall better result [25]. It is an approach that can be implemented either in parallel or sequentially.

4.1.1.1 Bootstrap Aggregation

Parallel ensemble learners use bootstrap aggregation (also known as bagging), which is based on building bootstrapped datasets from the initial dataset, i.e. creating datasets with a number of samples equal to that of the initial dataset but which are created by randomly selecting samples with replacement from the initial dataset. Furthermore, for each bootstrapped dataset a set containing the initial dataset's samples that have not been included in the bootstrapped dataset is stored in order to be used for validation purposes. Each bootstrapped dataset is then fed as training data into a machine learning model instance. The idea is to build a set of multiple models that have been trained on slightly different data, thus producing different results. Common approaches to improve the performance of the model instances include the selection of a subset of features to make predictions as well as the randomisation of bootstrapped datasets. Once all model instances have been trained, an aggregation technique is employed in order to determine the ensemble model's prediction. Bagging is used by random forests, which are essentially ensemble learners consisting of multiple decision trees.

4.1.1.2 Boosting

Sequential ensemble learners are built with the use of the boosting technique, which creates the overall model incrementally by assigning higher weights to training data samples for which the model instances' prediction was inaccurate. Common boosting algorithms include Adaptive Boosting (AdaBoost), Gradient Boosting and eXtreme Gradient Boosting (XGBoost).

AdaBoost is a boosting algorithm that improves its performance by assigning a weight probability to each training sample according to the previous model instance's performance. In the first iteration, all samples in the training dataset are assigned equal weights and the training dataset is fed into the first model instance. The training samples for which the model's prediction was inaccurate are then assigned a higher weight compared to the samples for which the model's prediction was accurate and the new dataset is given as input to the next model instance. Finally, the individual models' predictions are combined with the use of voting in cases of classification problems and averaging in cases of regression problems.

Gradient boosting works by using a sequential procedure which in each step tries to optimise the previous model instance's loss function in order to produce a new model instance whose predictions are more accurate than the previous instance's. The optimisation is performed in a way that is similar to neural network training, as it leverages the gradient descent algorithm, i.e. the process of minimising the loss gradient to fit the model. It is a technique that is particularly successful in problems with tabular data and is suitable for classification as well as for regression. The components of a gradient boosting algorithm are the loss function, the weak learners which are combined to provide the overall model and the additive model whose purpose is to add a weak learner in each step of the process and perform a gradient descent procedure in order to minimise the loss induced by adding a new model instance. The additive model modifies the parameters of the new model instances in order to ensure that the added weak learners contribute to reducing the loss function value.

XGBoost [26] was introduced in order to reduce the gradient boosting algorithms' requirements for time and computational resources by optimising the algorithm's speed and memory utilisation. It supports the use of parallel and distributed computing methods, while also using cache optimisation to optimise memory usage. XGBoost operates by attempting to address the weaknesses of gradient boosting trees and it succeeds in enhancing the predictive power of the models it produces by taking the distribution of features across the datapoints in a decision tree's leaf and reducing the size of the search space in order to facilitate the decisions of creating new branches within decision trees. This approach is particularly helpful in cases where a large number of features is provided as input to the algorithm, as in this case the number of potential splits is large, making the decision process time-consuming. It also which uses L1 and L2 regularisation in order to prevent overfitting.

4.1.2 Graph Convolutional Networks

Graph convolutional networks have been developed to provide a way for building predictive models for whole graphs as well as for nodes or edges within graphs. These networks

aim to tackle such tasks by learning embeddings to extract information regarding the structure of the graph. Graph convolutional networks use convolutional layers that are similar to the ones used in convolutional neural networks. Convolutional neural networks make use of trainable filters in order to learn from objects with grid structures and are particularly successful at dealing with tensor data where local patterns can be observed. Graph data do not fall in this category, as their structure does not resemble a grid. However, such data can still benefit from models that are able to identify local patterns, which is why graph convolutional networks have been developed. Graph convolutional networks are a generalised version of convolutional neural networks, as they are able to deal with graph data, where the nodes are unordered and the number of edges may be different in each graph. They employ local filters in order to aggregate and process information from neighbouring nodes. They can be split into spatial graph convolutional networks and spectral graph convolutional networks. Spatial graph convolutional networks [27] exploit spatial features in order to extract useful knowledge from graphs. They aggregate neighbour node information by using spatial characteristics, thus having a closer relationship with convolutional neural networks, where the position of each element in the grid-like input influences the result. Spectral graph convolutional networks [28] leverage spectral analysis, which is extensively used in network analysis for unsupervised tasks such as community detection and graph clustering, and combine it with the principles of convolutional neural networks in order to produce models that can be trained in a supervised manner. These networks leverage spectral decomposition, i.e. the eigendecomposition of the Laplacian matrix that represents the graph.

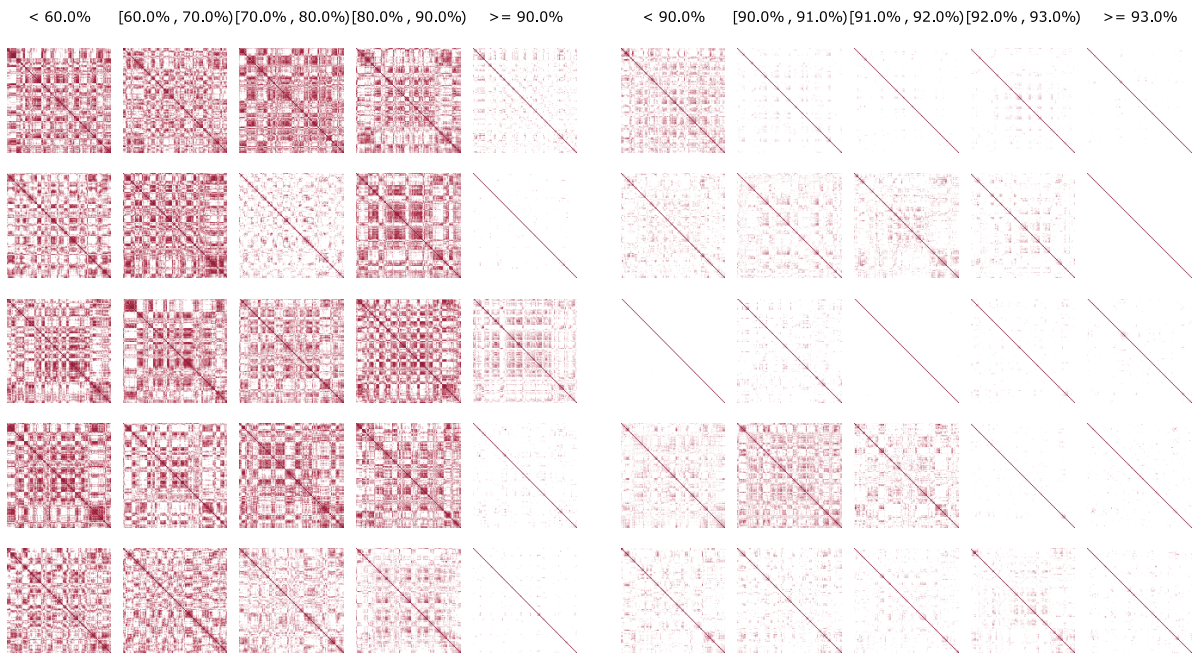
Graph convolutional applications can be useful in applications in various domains. First of all, they can be employed in cases where the processing of graph data is needed for tasks such as graph classification, node classification, link prediction and node clustering. However, graph convolutional networks can also be used to process other kinds of data. For example, they can be leveraged to solve problems in computer vision, such as image classification, visual question answering, image captioning and activity recognition. Furthermore, they can be employed to tackle problems related to natural language processing, such as text classification, information extraction, semantic role labelling and neural machine translation. Moreover, graph convolutional networks can also be used in social network analysis for tasks such as community detection, link prediction, retweet count forecasting, fake news detection and recommendation systems. As graph structures are commonly observed in various scientific domains, such as physics, chemistry, biology and materials science, graph convolutional networks can also be used to tackle scientific problems [29]. Finally, graph convolutional networks have efficiently been used in tasks related to neural architecture search [30, 31], which is why we have chosen to explore them as an option for performing fitness approximation.

4.2 Neural Architecture Search Without Training

The Neural Architecture Search Without Training (NASWT) score is an alternative approach for the acceleration of the candidate evaluation step, as it is able to assess the quality of candidate neural network architectures from their initial state [1]. The NASWT score can be calculated for networks that use the rectified linear unit (ReLU) activation function, as it looks at the overlap of these activations between a batch of data points in order to partially predict a network's performance after training. The score has a positive correlation with the trained networks' accuracy and its calculation only requires a few seconds on a single GPU, meaning that it is an approach that has the potential to accelerate candidate evaluation considerably.

A network's NASWT score is calculated by looking at the rectified linear unit (ReLU) activations of the network at its untrained state and forming a binary code based on which ReLU activations are positive and which are zero. Thus, inputs with similar codes correspond to the same linear region within a network and are therefore more difficult to disentangle, whereas inputs with different codes are easier to separate. The Hamming distance between pairs of binary codes of inputs is used in order to determine the level of dissimilarity of each pair of inputs and construct the kernel matrix to capture the difference between binary codes of a whole mini batch of input data. The kernel matrix is calculated by subtracting the distance between pairs of datapoints from the total number of activations in the network and the final NASWT score is obtained by using the log determinant of the kernel matrix. The intuition is that if the cross-correlation between two images is high, that means they're hard to separate and therefore it's harder to train the network, whereas if the cross-correlation is low, i.e. if the Hamming distance between the binary codes of a pair of inputs is high, then the two inputs have a large distance in the network's space thus making it easier for the network to be trained. That is why the kernel matrix is generally diagonal for the most high-performing networks, as the cross-correlation between different inputs should be low, allowing only for the diagonal entries of the matrix to have a high value. The kernel matrices of various untrained NAS-Bench-201 and NDS-DARTS architectures for a batch of 128 CIFAR-10 images are displayed in figure 4.1. The NASWT score doesn't take labels into account, but rather depicts a network's flexibility and gives an estimate of which networks are easier to train based on how easy it is for them to separate the different input datapoints.

Figure 4.2 displays the linear regions that are created based on the binary codes of ReLU activations. In step 1, each ReLU activation unit splits the input into two regions based on which nodes are positive and which are zero, thus creating an active and an inactive region within the input. Step 2 shows the intersections between the active and inactive regions of all ReLU activation nodes in the first layer (layer A). Step 3 shows the intersections between all ReLU activations in layers A and each activation in layer B.



(a) NAS-Bench-201 (b) NDS-DARTS

Figure 4.1: NAS Without Training Score - Kernel Matrices of Untrained Architectures (as displayed in the original paper [1])

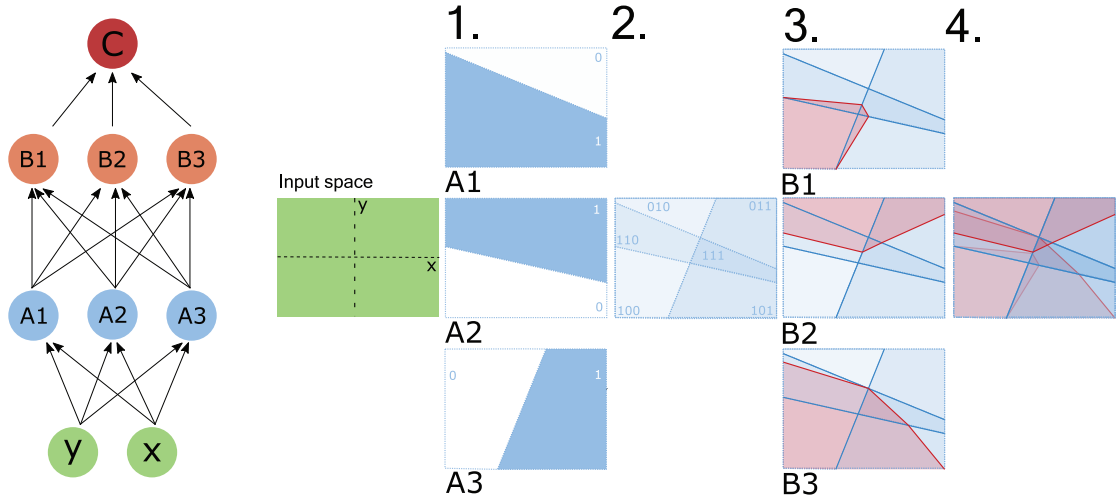


Figure 4.2: NAS Without Training Score - ReLU Activation Binary Codes (as displayed in the original paper [1])

Finally, step 4 shows the intersections between the ReLU activations of all nodes in layers A and B, thus creating unique identification patterns for each linear region.

Chapter 5

Experiments

In order to evaluate acceleration techniques for candidate evaluation in neural architecture search, we performed two different sets of experiments. In the first set of experiments, we use a vanilla genetic algorithm as the neural architecture search optimisation method in order to provide a baseline for comparison, whereas in the second experiment we use an enhanced evolutionary algorithm (NAS-EA-FA V2) that employs fitness approximation techniques to reduce the algorithm running time and produce high-performing neural network architectures. Each experiment is performed on the NAS-Bench-101 dataset and on the NATS-Bench topology search space dataset.

5.1 Vanilla Genetic Algorithm

In this section, the genetic algorithm that has been used in our experiments is described. An iterative procedure for a predefined number of iterations (T) is used. In the first iteration, a fixed number of architectures is randomly sampled from the search space to create the initial population. Each architecture is depicted with the use of a binary sequence that contains information regarding the types of layers of each architecture's cell as well as the connections between these layers. Due to the fact that the architectures have not been evaluated yet, the fitness of each of the sampled architectures is set to zero. In each iteration, the parents are chosen with the use of a tournament selection scheme with a fixed tournament size, whereas the offsprings are derived from the parents with the use of bitwise mutation in order to alter some of the types of layers as well as some of the connections between them. The layer mutation rate and the connection mutation rate are constant and defined at the start of the procedure. The new population is created with the use of the "replace oldest" strategy, i.e. by replacing all parents with the generated offsprings.

Two different sets of experiments have been conducted; the first set of experiments uses the validation accuracy as the architectures' fitness measure, which is the standard choice for evolutionary neural architecture search approaches. The use of the

validation accuracy as fitness measure means that each architecture in the population needs to be trained and evaluated on a specific dataset in order for the algorithm to assess its quality, which is not convenient due to time and computational power requirements. For this reason, a second set of experiments was conducted, where the NASWT score was used as fitness measure for the architectures in the population. The use of the NASWT score considerably reduces the time required for the search procedure, as it is able to evaluate each architecture within a few seconds on a single GPU.

5.2 NAS-EA-FA V2

NAS-EA-FA V2 is used as a means of accelerating the neural architecture search procedure, as it operates with the use of a fitness approximation model to estimate the fitness of candidate architectures, thus requiring the candidate evaluation of only a small number of architectures in each iteration to provide training data for the fitness approximation model. The fitness of the remaining architectures in each iteration is instantly estimated with the use of the fitness approximation model, which is trained in each iteration of the algorithm with the use of the available training data.

The algorithm performs an iterative procedure for a predefined number of iterations (T), which is described below: In the first iteration, a number of architectures is randomly sampled from the search space. The fitness of all architectures in the population is initialised with a value of zero, as in the beginning of the search process there are no performance statistics for the architectures. In each iteration, the architectures are sorted in descending order based on their fitness and the K architectures with the highest fitness as well as the H architectures with the largest distance from the previously evaluated architectures are selected to be used as training data for the fitness approximation model. These $K + H$ architectures are then trained and evaluated, whereas their isomorphisms are also identified. The training set for the current iteration's fitness approximation model is built by appending the $K + H$ architectures and their isomorphisms to the previous iteration's training set. In this way, the training set gets larger in each iteration, providing the model with the opportunity to achieve better predictive performance. The process of training the fitness approximation model is then performed by using the aforementioned data as input. Then, a genetic algorithm is used which runs for a predefined number of generations (G). The genetic algorithm works similarly to the vanilla genetic algorithm used in the first set of experiments, i.e. by selecting parents with the use of tournament selection and offsprings with the use of bitwise mutation, whereas the "replace oldest" strategy is used in each iteration for the creation of the next iteration's population. However, this genetic algorithm does not require a training and validation procedure for the generated offsprings, as their performance is estimated with the use of the trained fitness approximation model. It is expected that the first few iterations of the algorithm will pro-

duce mediocre results, as the training data provided to the fitness approximation model will be fairly limited. As the search process progresses, it is expected that the search algorithm's performance will improve, as the fitness approximation model's predictive power will increase due to the availability of a larger training dataset.

Again, two different options were explored regarding the fitness measure: the use of the validation accuracy and the use of the NASWT score. In the case where the validation accuracy is used to assess the produced architectures, the need for training and evaluating networks is not completely eliminated as $K + H$ architectures need to be trained in each iteration, but the amount of time that is saved is considerable in comparison to the vanilla genetic algorithm. In the case where the NASWT score is used as the fitness score, there is no need to train any architectures, which means that the time needed for the search process is significantly reduced, but at the same time some some noise is introduced due to the fact that the score does not provide an absolute measure of a network's performance. This noise is added to the noise already present in NAS-EA-FA V2 due to the use of the fitness approximation model, which estimates the performance of architectures instead of training them to determine their real performance, meaning that the results that are produced may not be that accurate.

Furthermore, two different fitness approximation models were considered: an XGBoost model, which is the model that is used in the original implementation of NAS-EA-FA V2, and a graph convolutional network that consists of five GraphSAGE layers with LSTM aggregators, a max pooling layer and two linear layers. The network was trained with the use of an Adam optimizer and a Mean Squared Error (MSE) loss function, as the problem that the network is intended to solve is a regression problem. As the use of a fitness approximation model aims to accelerate the neural architecture search procedure, the graph convolutional network is trained by using 30% of the dataset as training data and the remaining 70% as validation data. The summary of the graph convolutional network model used is displayed in listing 5.1.

Listing 5.1: Graph Convolutional Network Summary

```
GCN(
  (sage_lstm_h_1): SAGEConv(
    (feat_drop): Dropout(p=0.0, inplace=False)
    (lstm): LSTM(6, 6, batch_first=True)
    (fc_self): Linear(in_features=6, out_features=150, bias=False)
    (fc_neigh): Linear(in_features=6, out_features=150, bias=False)
  )
  (sage_lstm_h_2): SAGEConv(
    (feat_drop): Dropout(p=0.0, inplace=False)
    (lstm): LSTM(150, 150, batch_first=True)
```

```
    (fc_self): Linear(in_features=150, out_features=75, bias=False)
    (fc_neigh): Linear(in_features=150, out_features=75, bias=False)
  )
  (sage_lstm_h_3): SAGEConv(
    (feat_drop): Dropout(p=0.0, inplace=False)
    (lstm): LSTM(75, 75, batch_first=True)
    (fc_self): Linear(in_features=75, out_features=37, bias=False)
    (fc_neigh): Linear(in_features=75, out_features=37, bias=False)
  )
  (sage_lstm_h_4): SAGEConv(
    (feat_drop): Dropout(p=0.0, inplace=False)
    (lstm): LSTM(37, 37, batch_first=True)
    (fc_self): Linear(in_features=37, out_features=18, bias=False)
    (fc_neigh): Linear(in_features=37, out_features=18, bias=False)
  )
  (sage_lstm_h_5): SAGEConv(
    (feat_drop): Dropout(p=0.0, inplace=False)
    (lstm): LSTM(18, 18, batch_first=True)
    (fc_self): Linear(in_features=18, out_features=9, bias=False)
    (fc_neigh): Linear(in_features=18, out_features=9, bias=False)
  )
  (pool_1_m): MaxPooling()
  (fc1): Linear(in_features=9, out_features=4, bias=True)
  (fc2): Linear(in_features=4, out_features=1, bias=True)
)
```

5.3 Experimental Setup

5.3.1 The NORD Framework

Neural Operations Research and Development (NORD) [32] is a Python framework that has been released in order to aid the design and development of neural architecture search pipelines by providing a means of decoupling the process of designing and implementing such algorithms. The NORD framework also provides an easy way to test neural architecture search approaches on both benchmark datasets and custom datasets, thus paving the way for the fair comparison of different ideas concerning the search process.

NORD's basic functionality includes the neural descriptor class, which takes graph topologies as input and builds the equivalent neural network architectures, and the neural evaluator class, which includes everything that is needed for the candidate evalu-

ation process (i.e. functions for performing training and validation as well as functions to retrieve precomputed performance statistics from benchmark datasets). The neural descriptor class contains functions to add layers, add layers sequentially, form and remove connections between layers and convert neural descriptor objects to NetworkX objects. The neural evaluator contains functions to load data, set the device that will be used for training (i.e. CPU or GPU), convert neural descriptor objects to PyTorch neural network models, set the optimizer, train and test the network, provide information regarding the training procedure after each epoch, calculate the network's loss and performance statistics and evaluate both neural descriptor objects and neural network models. NORD has been used in all experiments conducted within the scope of this thesis in order to easily evaluate and compare the generated architectures. Furthermore, the functionality of the neural evaluator class has been extended to support the calculation of the NASWT score by taking as input the training dataset and the desired size of the batch of data that will be used for the calculation of the score.

5.3.2 Random Architecture Sampling

In the beginning of the evolutionary neural architecture search procedures that are being explored in this thesis, the initial population is derived by randomly sampling architectures from the given search space. The architectures are generated by randomly choosing from the available types of layers and randomly forming connections between the chosen layers. In the graph that is generated for each of the sampled architectures, the nodes depict the layers, whereas the edges represent the connections between layers. During the generation process it is possible to generate architectures that contain nodes that have either no input or no output connections, leading to the creation of inconsistent architectures. In order to resolve this issue, all generated architectures are being checked in order to identify such cases and the nodes without input or output connections are pruned, meaning that the resulting population contains architectures with different numbers of layers.

5.3.3 Parameter Values

The parameter settings used for the experiments that were conducted are reported in table 5.1. The values of the parameters generally follow the original implementation of NAS-EA-FA V2.

Table 5.1: Parameter Values

Algorithm	Parameter	Value
Vanilla genetic algorithm	Population size	100
	Tournament size	20%
	Layer mutation rate	$1/5$
	Connection mutation rate	$1/21$
	Number of generations	100
NAS-EA-FA V2	Population size	100
	Tournament size	20%
	Layer mutation rate	$1/5$
	Connection mutation rate	$1/21$
	Number of iterations (T)	10
	Number of generations (G) in each iteration	10
	K	30
	H	20
	XGBoost learning rate	0.1
Graph convolutional network	Train percentage	30%
	Train batch size	64
	Validation batch size	64
	Number of filters	150
	Adam learning rate	0.001
	Number of epochs	100

Chapter 6

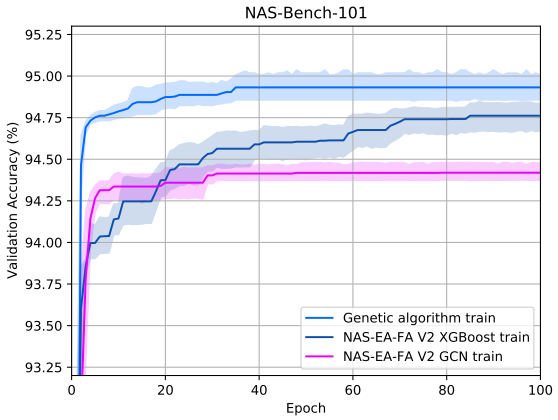
Results

In this chapter, the results of the experiments that were conducted as part of this thesis are reported. The experiments' aim was to assess the effect of various acceleration techniques for the candidate evaluation step of neural architecture search procedures, as well as to compare different acceleration techniques. All experiments were conducted on the NAS-Bench-101 dataset and on the topology search space of the NATS-Bench dataset.

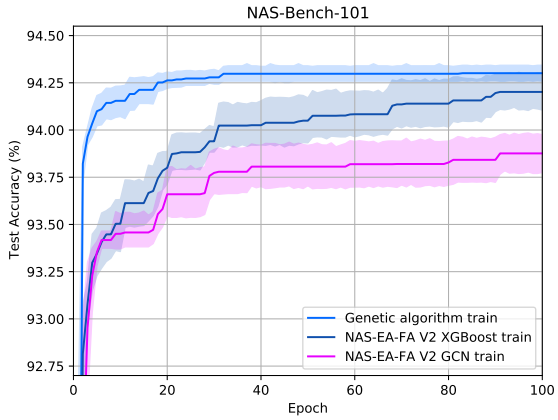
6.1 Performance Statistics

The first experiment focuses on comparing the performance of a simple genetic algorithm, the original NAS-EA-FA V2 algorithm where an XGBoost model is used for fitness approximation and the NAS-EA-FA V2 algorithm where a graph convolutional network is used for fitness approximation. In this experiment, all algorithms use the validation accuracy as the fitness score of candidate architectures. The simple genetic algorithm requires conducting a training and validation procedure on all the produced architectures, whereas NAS-EA-FA V2 aims to speed up the process with the use of fitness approximation and thus requires training and evaluating a only a small number of architectures in each iteration in order to provide training data for the fitness approximation model. In figure 6.1, the best accuracies of 10 runs are observed, with the mean and 95% confidence interval displayed. The conclusion that can be drawn is that the simple genetic algorithm produces better architectures than NAS-EA-FA V2, as NAS-EA-FA V2 seems to restrict the search space, probably due to the fitness approximation training data selection process, which selects the best performing architectures in each iteration and the architectures with the largest distance from the previously trained architectures. The difference in the performance of the algorithms is more apparent in the case of NATS-Bench than in the case of NAS-Bench-101. Furthermore, it seems that the simple genetic algorithm is more stable in the sense that the accuracies of the best architectures it produces do not change as much as the accuracies of the best architectures produced by NAS-EA-FA V2, as the simple genetic algorithm's 95% confidence interval is narrower. Concerning the compari-

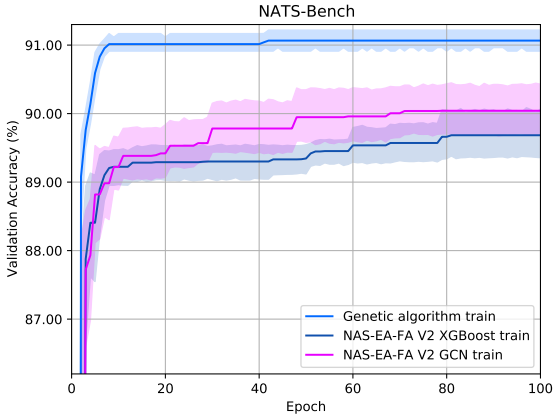
son between XGBoost and the graph convolutional network, the XGBoost model seems to perform better in the case of NAS-Bench-101, whereas the graph convolutional network achieves better performance in the case of NATS-Bench, thus a definitive conclusion as to the best choice of fitness approximation model cannot be reached.



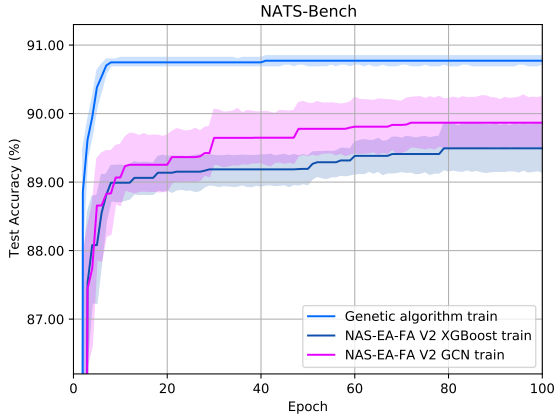
(a) NAS-Bench-101 - Best Validation Accuracy



(b) NAS-Bench-101 - Best Test Accuracy



(c) NATS-Bench - Best Validation Accuracy

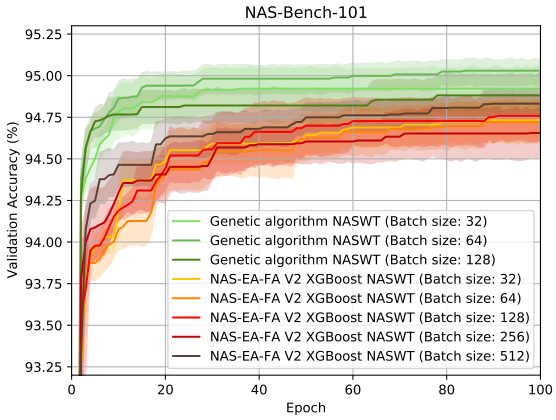


(d) NATS-Bench - Best Test Accuracy

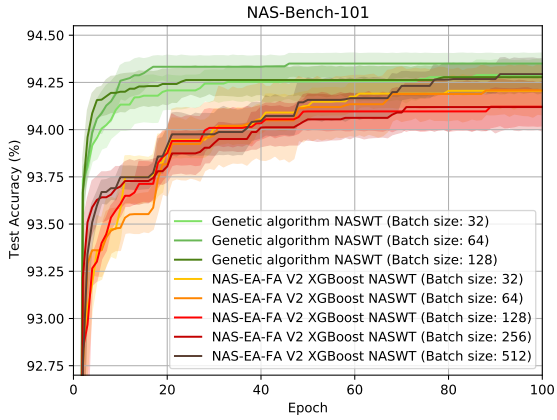
Figure 6.1: Genetic Algorithm Train - NAS-EA-FA V2 XGBoost Train - NAS-EA-FA V2 GCN Train - Best Accuracy of 10 Runs (mean and 95% confidence interval displayed)

The second experiment aims to assess the effectiveness of the NASWT score as fitness measure. We use a simple genetic algorithm and NAS-EA-FA V2 as optimisation methods for the neural architecture search procedure and experiment with the NASWT score as fitness measure by using different data batch sizes for its calculation. The observation that can be made is that the simple genetic algorithm produces better architectures than NAS-EA-FA V2 as can be seen in figures 6.2a, 6.2b, 6.3a and 6.3b, but is worse at identifying the best-performing architectures as can been in figures 6.2c, 6.2d, 6.3c and 6.3d. Thus, in the case where the NASWT score is used as the fitness score, NAS-EA-FA V2 seems to be a better choice than the simple genetic algorithm. Regarding the batch size of the NASWT score, it seems that a larger batch size doesn't always guarantee better

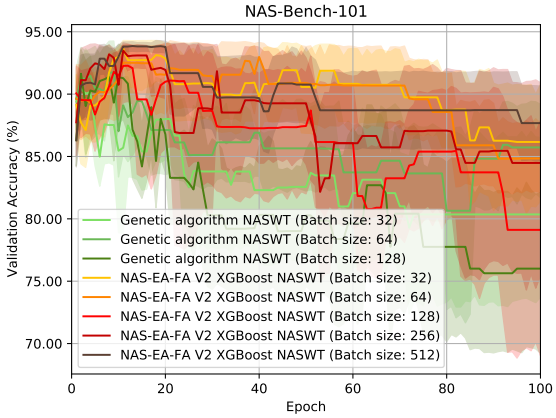
performance. By looking at the results we can conclude that in the case of NAS-Bench-101, the simple genetic algorithm performs better when the NASWT score is calculated with the use of a batch size of 128, whereas in the case of NAS-EA-FA V2, the NASWT score with a batch size of 512 yields the best results. In the case on NATS-Bench, it seems that the best performance is reached with the use of the NASWT score with a batch size of 128 for both the vanilla genetic algorithm and NAS-EA-FA V2.



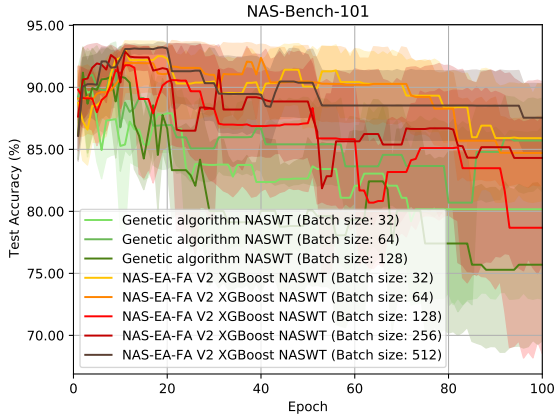
(a) NAS-Bench-101 - Best Validation Accuracy



(b) NAS-Bench-101 - Best Test Accuracy



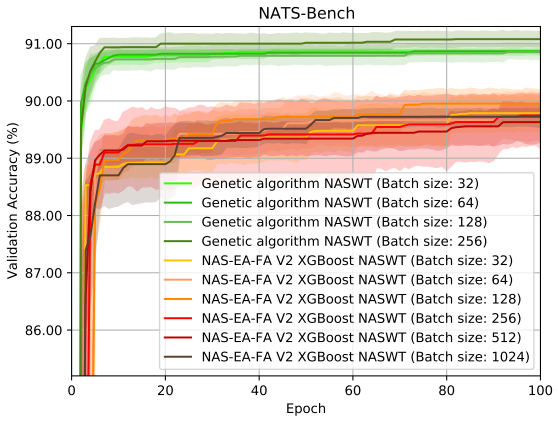
(c) NAS-Bench-101 - Best Validation Accuracy Based on NASWT Score



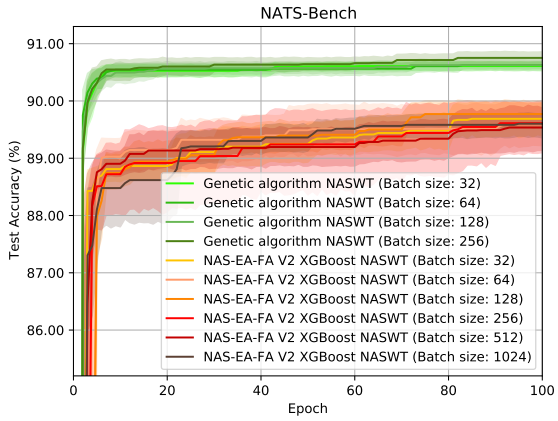
(d) NAS-Bench-101 - Best Test Accuracy Based on NASWT Score

Figure 6.2: Genetic Algorithm NASWT - NAS-EA-FA V2 NASWT - NAS-Bench-101 - Best Accuracy of 10 Runs (mean and 95% confidence interval displayed)

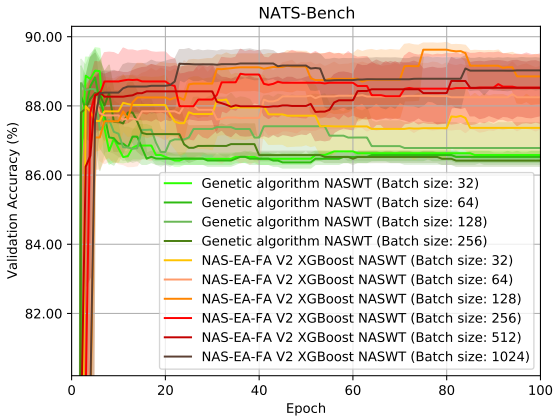
In order to draw our final conclusions, we also created some comparative diagrams between different approaches for the acceleration of candidate evaluation. Methods that use both the validation accuracy and the NASWT score are included, in order to provide a means of assessing the validity of each of the proposed fitness scores. In figures 6.4 and 6.5, three approaches that use the validation accuracy as fitness score and two approaches that use the NASWT score with a batch size of 32 are considered. The first three approaches include a simple genetic algorithm, NAS-EA-FA V2 with the use of an XGBoost model for the fitness approximation and NAS-EA-FA V2 with a fitness ap-



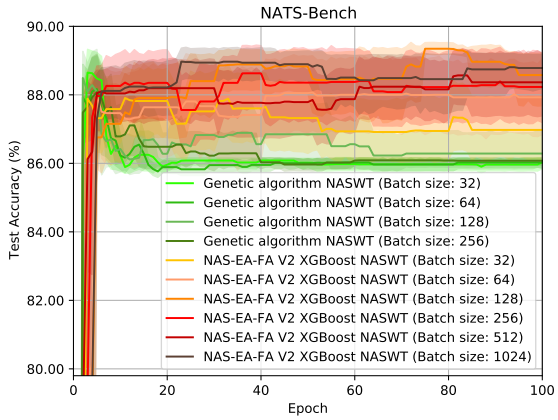
(a) NATS-Bench - Best Validation Accuracy



(b) NATS-Bench - Best Test Accuracy



(c) NATS-Bench - Best Validation Accuracy Based on NASWT Score

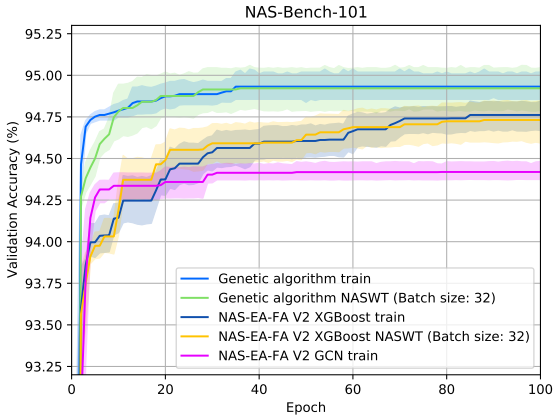


(d) NATS-Bench - Best Test Accuracy Based on NASWT Score

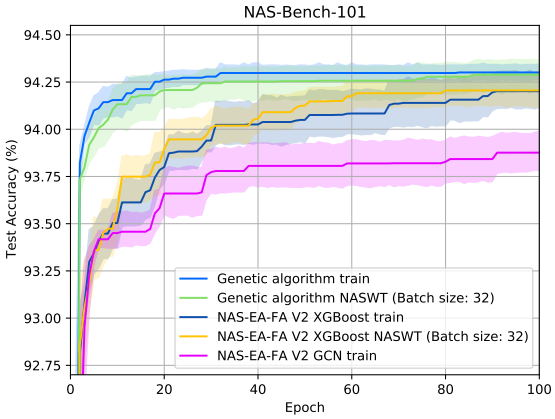
Figure 6.3: Genetic Algorithm NASWT - NAS-EA-FA V2 NASWT - NATS-Bench - Best Accuracy of 10 Runs (mean and 95% confidence interval displayed)

proximation graph convolutional network model. The remaining two approaches, which assess the quality of candidate architectures with the use of the NASWT score, are a simple genetic algorithm and NAS-EA-FA V2 with the use of an XGBoost model. It can be seen that the simple genetic algorithm generally produces better architectures than NAS-EA-FA V2 regardless of the chosen fitness measure for reasons that have been mentioned above. NAS-EA-FA V2 with the use of XGBoost and the validation accuracy as the fitness measure seems to produce similar architectures with NAS-EA-FA V2 with the use of XGBoost and NASWT score as the fitness measure, whereas the approach where NAS-EA-FA V2 uses a graph convolutional network as fitness approximator seems to produce slightly worse architectures than the two aforementioned approaches in NAS-Bench-101 and slightly better architectures than the two aforementioned approaches in NATS-Bench. However, it is apparent that the NASWT score is not capable of accurately distinguishing the best-performing architectures, as can be seen in figures 6.4c, 6.4d, 6.5c and 6.5d. Therefore, it seems that NAS-EA-FA V2 with the use of the validation accu-

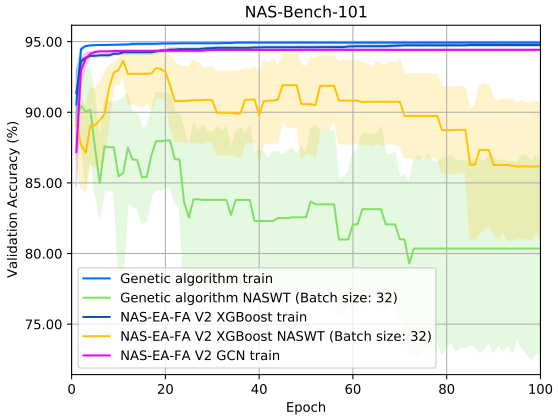
racy as fitness score is preferred over the use of the NASWT score for the acceleration of the neural architecture search process, even though NAS-EA-FA V2 requires the training and evaluation of some architectures to provide training data for the fitness approximation model. In the case of NAS-Bench-101, the best-performing acceleration technique is NAS-EA-FA V2 with the use of XGBoost, whereas in the case of NATS-Bench, the graph convolutional network as fitness approximation model produces better results than XGBoost.



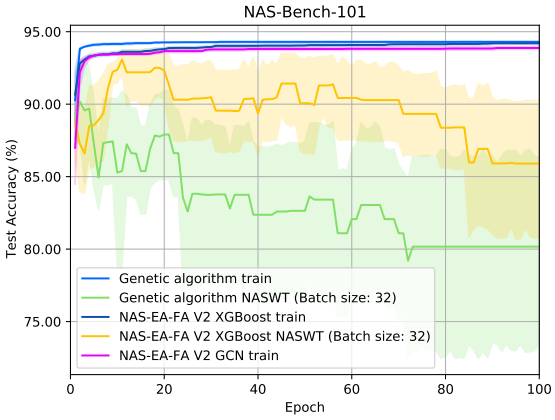
(a) NAS-Bench-101 - Best Validation Accuracy



(b) NAS-Bench-101 - Best Test Accuracy



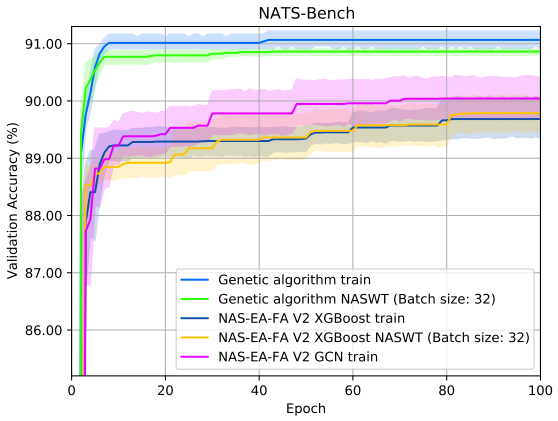
(c) NAS-Bench-101 - Best Validation Accuracy Based on NASWT Score



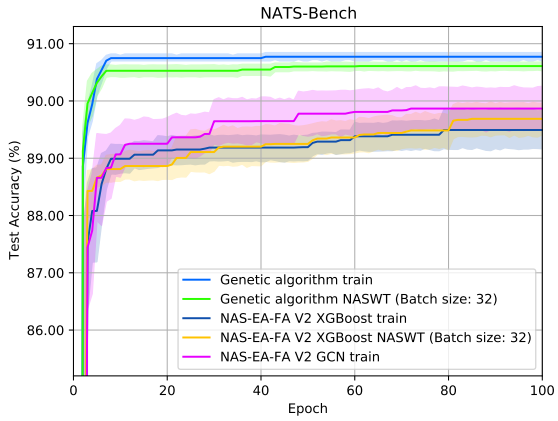
(d) NAS-Bench-101 - Best Test Accuracy Based on NASWT Score

Figure 6.4: Comparison 1 - NAS-Bench-101 - Best Accuracy of 10 Runs (mean and 95% confidence interval displayed)

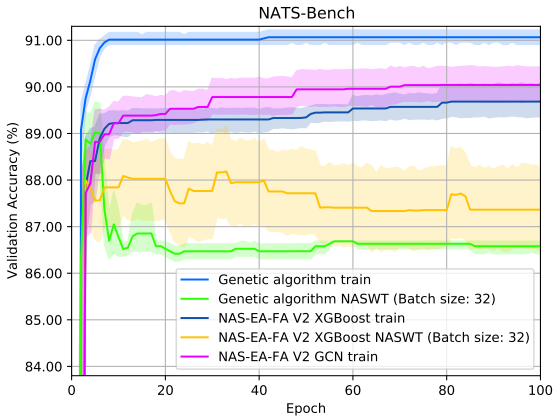
Furthermore, in figures 6.6 and 6.7, the same comparison was performed with the use of the best-performing NASWT score approaches in terms of batch sizes. The conclusion that can be drawn is the same as the one drawn from the previous comparative diagrams. Although altering the batch size of the NASWT score may lead to discovering slightly better architectures, the NASWT score's correlation with the validation accuracy seems to not be strong enough in order for the score to be able to identify the best-performing architectures, meaning that the validation accuracy is still a more accurate



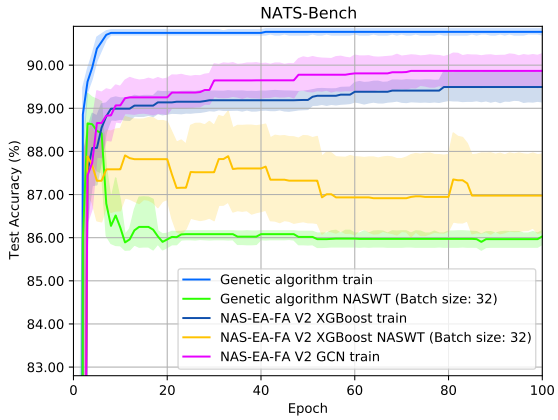
(a) NATS-Bench - Best Validation Accuracy



(b) NATS-Bench - Best Test Accuracy



(c) NATS-Bench - Best Validation Accuracy Based on NASWT Score



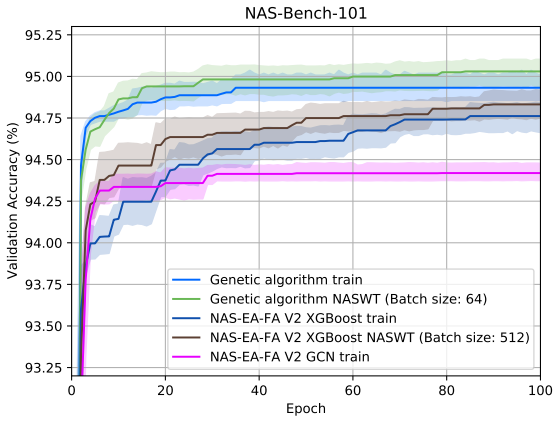
(d) NATS-Bench - Best Test Accuracy Based on NASWT Score

Figure 6.5: Comparison 1 - NATS-Bench - Best Accuracy of 10 Runs (mean and 95% confidence interval displayed)

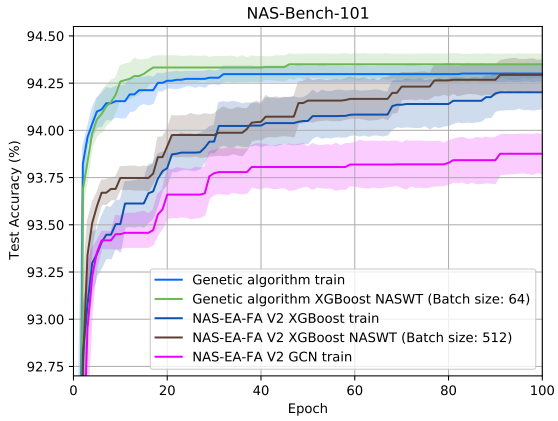
measure of candidate architectures' performance. However, the NASWT score may be a preferred choice in cases where the time and computational resources are severely limited, as the fact that no full training and evaluation is required means that the use of the NASWT score as fitness measure may be capable of discovering adequate architectures quickly and cheaply, although a compromise on the quality of the produced architectures will need to be made.

6.2 Validation Accuracy Distributions

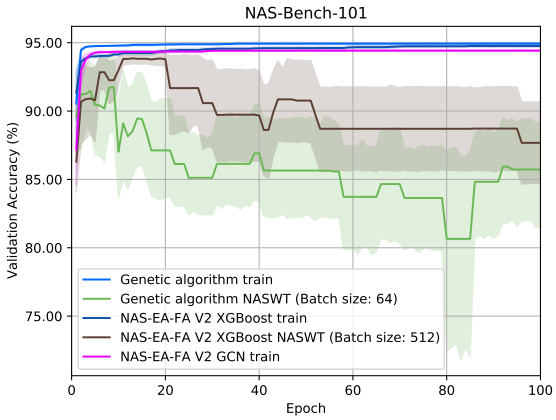
Figures 6.8 and 6.9 display the distributions of the validation accuracies of the architectures produced by the different variations of the genetic algorithm and NAS-EA-FA V2 search procedures. In the case of NAS-Bench-101, the majority of the produced architectures have a validation accuracy between 90% and 95%, whereas there is also a small peak around 85%, which probably depicts architectures that are generated at the start



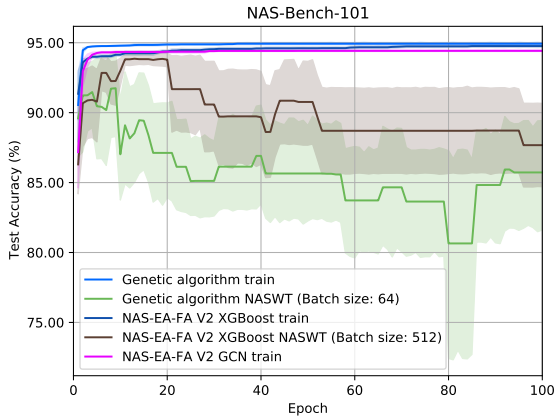
(a) NAS-Bench-101 - Best Validation Accuracy



(b) NAS-Bench-101 - Best Test Accuracy



(c) NAS-Bench-101 - Best Validation Accuracy Based on NASWT Score

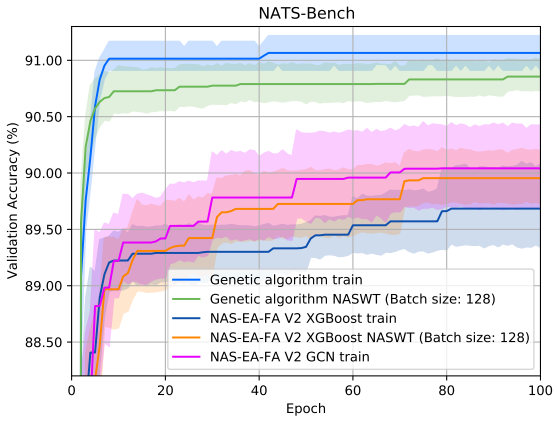


(d) NAS-Bench-101 - Best Test Accuracy Based on NASWT Score

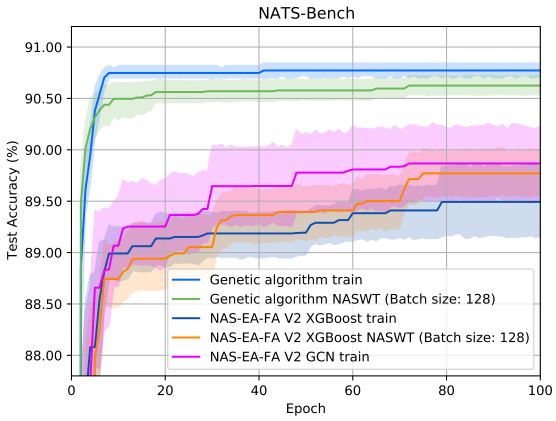
Figure 6.6: Comparison 2 - NAS-Bench-101 - Best Accuracy of 10 Runs (mean and 95% confidence interval displayed)

of the search procedure. In the case of NATS-Bench, we observe two peaks in the validation accuracy distributions. The highest peak is at around 90%, whereas there is also a lower peak at around 10%, probably representing architectures that appear at the start of the search procedure. We observe that the distributions of the validation accuracies are broader in the case of NATS-Bench, meaning that with the use of the NATS-Bench topology search space the search procedure produces architectures that are more diverse in terms of validation accuracy compared to when the NAS-Bench-101 search space is used.

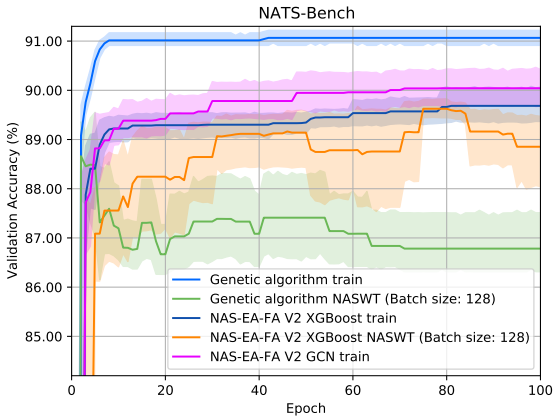
The first observation that can be made is that the architectures produced when the graph convolutional network is used for the fitness approximation produce a validation accuracy distribution that is broader than that of the case where XGBoost is used as a fitness approximation model as well as that of the vanilla genetic algorithm that uses the validation accuracy as fitness. Concerning the case where the NASWT score is used as the fitness score, we observe that in the case of the NATS-Bench topology search space,



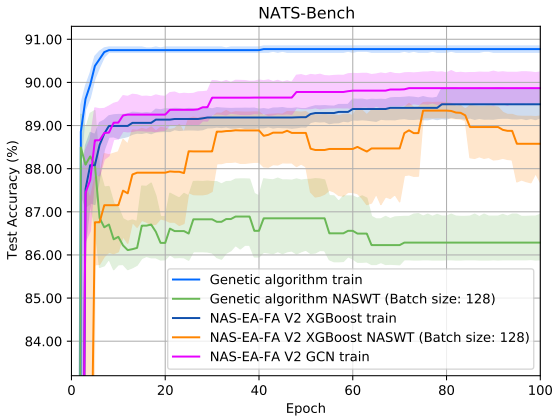
(a) NATS-Bench - Best Validation Accuracy



(b) NATS-Bench - Best Test Accuracy



(c) NATS-Bench - Best Validation Accuracy Based on NASWT Score

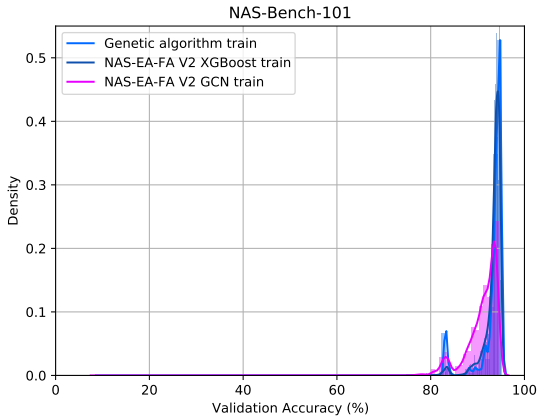


(d) NATS-Bench - Best Test Accuracy Based on NASWT Score

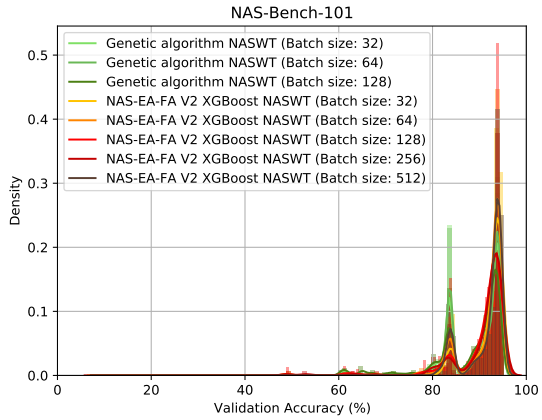
Figure 6.7: Comparison 2 - NATS-Bench - Best Accuracy of 10 Runs (mean and 95% confidence interval displayed)

the validation accuracy distributions have a somewhat sharper peak in the case where the genetic algorithm is used as optimisation method.

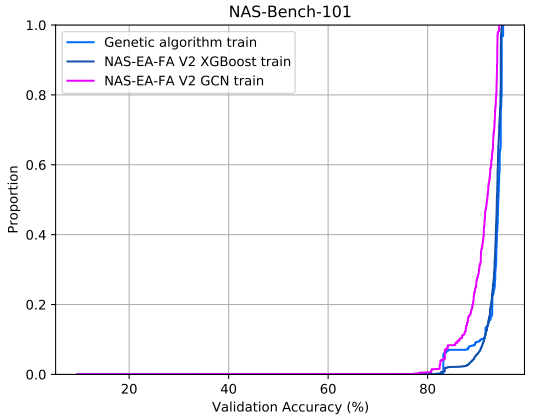
In order to determine whether the validation accuracies produced by the proposed approaches belong to the same distributions, the two-sample Kolmogorov-Smirnov test [33] was used. The results can be seen in table 6.1. The conclusion that can be drawn is that the validation accuracies that are produced with the use of the vanilla genetic algorithm belong to different distributions compared to the validation accuracies of the architectures that NAS-EA-FA V2 produces.



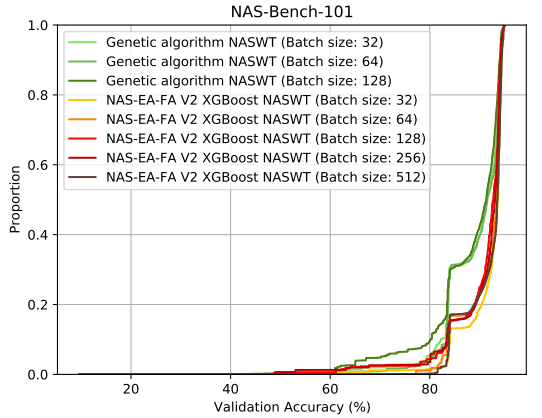
(a) NAS-Bench-101 - Validation Accuracy Distribution



(b) NAS-Bench-101 - Validation Accuracy Distribution



(c) NAS-Bench-101 - Validation Accuracy Empirical Cumulative Distribution

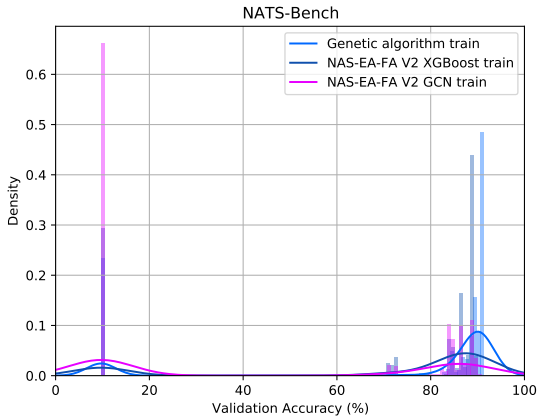


(d) NAS-Bench-101 - Validation Accuracy Empirical Cumulative Distribution

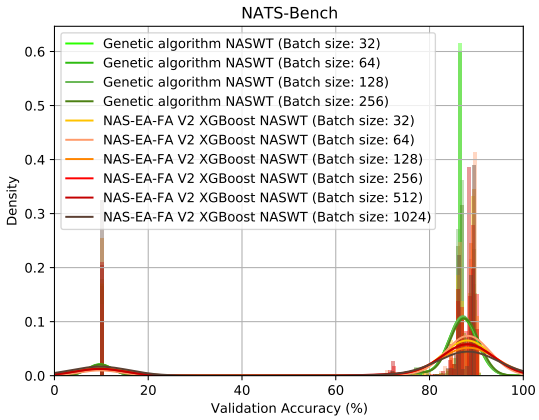
Figure 6.8: NAS-Bench-101 - Distributions of the Validation Accuracies of the Produced Architectures

Table 6.1: Kolmogorov-Smirnov Two-Sample Test

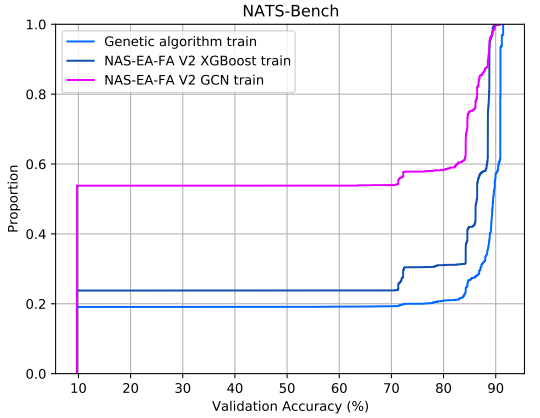
Benchmark	Method 1	Method 2	Statistic	P-Value
NAS-Bench-101	Genetic algorithm train	NAS-EA-FA V2 XGBoost train	0.211	$8.02 \cdot 10^{-98}$
	Genetic algorithm train	NAS-EA-FA V2 GCN train	0.5538	0.0
	Genetic algorithm NASWT (Batch size: 32)	NAS-EA-FA V2 XGBoost NASWT (Batch size: 32)	0.2698	$1.89 \cdot 10^{-160}$
	Genetic algorithm NASWT (Batch size: 64)	NAS-EA-FA V2 XGBoost NASWT (Batch size: 64)	0.2216	$6.14 \cdot 10^{-108}$
	Genetic algorithm NASWT (Batch size: 128)	NAS-EA-FA V2 XGBoost NASWT (Batch size: 128)	0.202	$1.24 \cdot 10^{-89}$
NATS-Bench	Genetic algorithm train	NAS-EA-FA V2 XGBoost train	0.5566	0.0
	Genetic algorithm train	NAS-EA-FA V2 GCN train	0.5694	0.0
	Genetic algorithm NASWT (Batch size: 32)	NAS-EA-FA V2 XGBoost NASWT (Batch size: 32)	0.327	$6.40 \cdot 10^{-237}$
	Genetic algorithm NASWT (Batch size: 64)	NAS-EA-FA V2 XGBoost NASWT (Batch size: 64)	0.3652	$8.18 \cdot 10^{-297}$
	Genetic algorithm NASWT (Batch size: 128)	NAS-EA-FA V2 XGBoost NASWT (Batch size: 128)	0.342	$1.40 \cdot 10^{-259}$



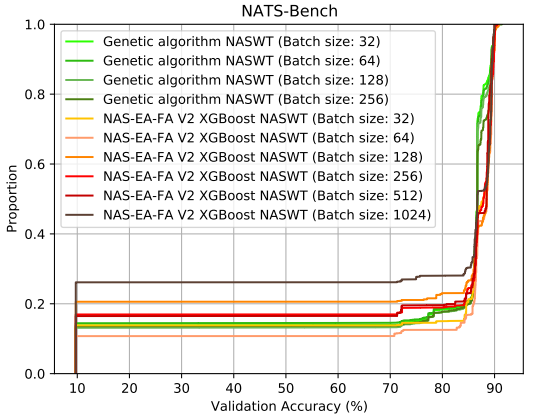
(a) NATS-Bench - Validation Accuracy Distribution



(b) NATS-Bench - Validation Accuracy Distribution



(c) NATS-Bench - Validation Accuracy Empirical Cumulative Distribution



(d) NATS-Bench - Validation Accuracy Empirical Cumulative Distribution

Figure 6.9: NATS-Bench - Distributions of the Validation Accuracies of the Produced Architectures

Chapter 7

Conclusion

Deep learning techniques are widely used in today's society, as their representation and predictive power can be leveraged to solve various different problems. As the success of deep learning relies on the selection of an appropriate neural network architecture for the given task, the task of automatically selecting neural network architectures has recently become a topic of research. This is how the field of neural architecture search has emerged, which is a subfield of automated machine learning that aims to automatically decide on optimal neural network architectures by leveraging the power of optimisation algorithms. Neural architecture search has so far succeeded in discovering state-of-the-art architectures for a wide range of problems.

The neural architecture search procedure requires performing candidate evaluation in order to determine the quality of architectures that are generated during the search process and thus guide the search towards selecting high-performing neural network architectures. The candidate evaluation process is typically a time-consuming and computationally-intensive process, as it is typically performed with the use of a standard training and validation procedure of all the generated architectures on a given dataset. This thesis focuses on the acceleration of candidate evaluation in neural architecture search by employing various acceleration techniques and assessing their effect on the search procedure and the quality of the produced architectures. We explored the possibility of using fitness approximation models to estimate the candidate architectures' fitness instead of calculating it directly. The use of two different fitness approximation models, an XGBoost model and a graph convolutional network model, was investigated. Furthermore, the use of the NASWT score as the fitness measure of candidates was investigated. The NASWT score requires no training and can be calculated within a few seconds on a single GPU, thus providing a cheap alternative to the validation accuracy, which is usually the chosen fitness measure of candidate architectures in neural architecture search processes. Extensive experiments were conducted on two neural architecture search benchmark datasets with an aim of determining the effect of the proposed acceleration techniques.

The results of our computational study show that the use of a fitness approx-

imation model introduces noise to the neural architecture search process, as there is no direct correspondence between the estimated fitness and the real fitness of candidate architectures. Moreover, the use of fitness approximation models seems to narrow down the search space and force the search algorithm to only explore specific regions of the search space, which may lead to excluding the best-performing architectures. Concerning the NASWT score, is seems that it is able to guide the search to produce high-performing architectures and that is does not impose any restrictions on the search space, however it is unable to accurately distinguish the best-performing architectures, meaning that it is not an accurate enough measure of a network's performance after training. As the NASWT score succeeds in guiding the search towards optimal architectures, it might be possible to use it along with another fitness measure. In such a setting, the NASWT score could be used to accelerate the search procedure and another fitness measure could then be used to identify the best-performing architectures.

Although this study has been quite extensive, there is still room for further experimentation. Future work may include the investigation of the behaviour of other machine learning models as fitness approximators, as well as the use of other metaheuristics as optimisation methods for the search procedure. Potential metaheuristics that could be explored are particle swarm optimisation and ant colony optimisation. Furthermore, as this thesis focused on evolutionary neural architecture search, it might be useful to investigate the use of acceleration techniques in cases where other optimisation methods, such as reinforcement learning and Bayesian optimisation, are used in order to explore the search space.

References

[1] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 7588–7598. [Online]. Available: https://proceedings.mlr.press/v139/mellor21a.html

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, jan 2014.

[4] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[5] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent advances in recurrent neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1801.01078

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 103–111. [Online]. Available: https://aclanthology.org/W14-4012

[8] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[9] 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

[10] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," 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. 4095–4104. [Online]. Available: https://proceedings.mlr.press/v80/pham18a.html

[11] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 2019.

[12] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg

[13] B. M. Oloulade, J. Gao, J. Chen, T. Lyu, and R. Al-Sabri, "Graph neural architecture search: A survey," *Tsinghua Science and Technology*, vol. 27, no. 4, pp. 692–708, 2022.

[14] G. Kyriakides and K. Margaritis, "An introduction to neural architecture search for convolutional networks," *arXiv preprint arXiv:2005.11074*, 2020.

[15] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-598.html

[16] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," *arXiv preprint arXiv:1905.01392*, 2019.

[17] G. Kyriakides and K. Margaritis, "The effect of reduced training in neural architecture search," *Neural Computing and Applications*, vol. 32, no. 23, pp. 17 321–17 332, 2020.

[18] P. Ren, Y. Xiao, X. Chang, P.-y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surv.*, vol. 54, no. 4, may 2021. [Online]. Available: https://doi.org/10.1145/3447582

[19] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *Proceedings*

of the 36th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 7105–7114. [Online]. Available: http://proceedings.mlr.press/v97/ying19a.html

[20] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HJxyZkBKDr

[21] X. Dong, L. Liu, K. Musial, and B. Gabrys, "NATS-Bench: Benchmarking NAS algorithms for architecture topology and size," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3634–3646, 2022.

[22] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210650211000198

[23] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE transactions on Neural Networks and Learning Systems*, 2021.

[24] C. Pan and X. Yao, "Neural architecture search based on evolutionary algorithms with fitness approximation," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–8.

[25] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers of Computer Science*, vol. 14, no. 2, pp. 241–258, 2020.

[26] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794.

[27] T. Danel, P. Spurek, J. Tabor, M. Śmieja, Ł. Struski, A. Słowik, and Ł. Maziarka, "Spatial graph convolutional networks," in *Neural Information Processing*, H. Yang, K. Pasupa, A. C.-S. Leung, J. T. Kwok, J. H. Chan, and I. King, Eds. Cham: Springer International Publishing, 2020, pp. 668–675.

[28] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013. [Online]. Available: https://arxiv.org/abs/1312.6203

[29] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1–23, 2019.

[30] G. Kyriakides and K. Margaritis, "Evolving graph convolutional networks for neural architecture search," *Neural Computing and Applications*, vol. 34, no. 2, pp. 899–909, 2021.

[31] H. Shi, R. Pi, H. Xu, Z. Li, J. T. Kwok, and T. Zhang, "Multi-objective neural architecture search via predictive network performance optimization," 2020. [Online]. Available: https://openreview.net/forum?id=rJgffkSFPS

[32] G. Kyriakides and K. Margaritis, "NORD: A python framework for neural architecture search," *Software Impacts*, vol. 6, p. 100042, 2020.

[33] J. L. Hodges, "The significance probability of the Smirnov two-sample test," *Arkiv för Matematik*, vol. 3, no. 5, pp. 469–486, Jan. 1958.