DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

Bayesian Optimization for Neural Architecture Search using Graph Kernels

Bharathwaj Krishnaswami Sreedhar

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Author

Bharathwaj Krishnaswami Sreedhar - bks@kth.se
Electrical Engineering and Computer Science
KTH Royal Institute of Technology

Place for Project

Sony Europe B.V.
Stuttgart, Germany

Examiner

Dr. Amir Payberah
KTH Royal Institute of Technology

Supervisor

Dr. Magnus Boman

KTH Royal Institute of Technology

Abstract

Neural architecture search is a popular method for automating architecture design. Bayesian optimization is a widely used approach for hyper-parameter optimization and can estimate a function with limited samples. However, Bayesian optimization methods are not preferred for architecture search as it expects vector inputs while graphs are high dimensional data. This thesis presents a Bayesian approach with Gaussian priors that use graph kernels specifically targeted to work in the higher-dimensional graph space. We implemented three different graph kernels and show that on the NAS-Bench-101 dataset, an untrained graph convolutional network kernel outperforms previous methods significantly in terms of the best network found and the number of samples required to find it. We follow the AutoML guidelines to make this work reproducible.

Keywords

Neural architecture search, Bayesian optimization, Graph kernels,
Graph convolutional networks

Abstract

Neural arkitektur sökning är en populär metod för att automatisera arkitektur design. Bayesian-optimering är ett vanligt tillvägagångssätt för optimering av hyperparameter och kan uppskatta en funktion med begränsade prover. Bayesianska optimeringsmetoder är dock inte att föredra för arkitektonisk sökning eftersom vektoringångar förväntas medan grafer är högdimensionella data. Denna avhandling presenterar ett Bayesiansk tillvägagångssätt med gaussiska prior som använder grafkärnor som är särskilt fokuserade på att arbeta i det högre dimensionella grafutrymmet. Vi implementerade tre olika grafkärnor och visar att det på NAS-Bench-101-data, till och med en otränad Grafkonvolutionsnätverk-kärna, överträffar tidigare metoder när det gäller det bästa nätverket som hittats och antalet prover som krävs för att hitta det. Vi följer AutoML-riktlinjerna för att göra detta arbete reproducerbart.

Nyckelord

Neural-arkitektursökning, Bayesian-optimering, Graph-kernels, Grafkonvolutionsnätverk

Contents

Chapter 1

Introduction

The field of deep learning has experienced tremendous growth in the past decade. Many deep learning models have achieved human-level performance or better on a wide variety of tasks such as image classification [51, 52, 62], speech synthesis [37, 55] and language processing [17, 54] to name a few. A deep learning model or network consists of various mathematical operations or *layers* and connections between these layers. Although there are many important factors associated with the development of a deep learning network, two main concepts are 1) *architecture engineering* which deals with the design of the network and how the various layers involved are connected, 2) *hyper-parameters selection* which decides the training configuration.

Typically, both processes are done manually and require prior knowledge and experience with deep learning architectures. Since each architecture needs to be trained to determine its performance and there are potentially infinite architectures that can solve a particular problem, the process of obtaining the best model for a task is a time consuming one and can be resource-intensive when multiple GPUs are used to train different models simultaneously. *Neural Architecture Search* (NAS) is a technique that automates the process of architecture design. It is used to search for a neural network amongst a limited set of possible networks that can achieve the desired performance in a relatively short time. Deep learning networks obtained through NAS have already outperformed many hand-crafted models in various areas like image classification [3, 62] and language modelling [62]. The process of NAS can be defined as an optimization problem, as shown in equation 1.1.

$$A^* = \underset{A \in \mathbb{A}}{\text{argmin}} J(\theta, A; D_{val}) \tag{1.1}$$

A^* is the desired architecture which minimizes the cost function or *objective function* J over the set of architectures $\mathbb{A}$, when each architecture is trained with the hyper-parameters θ. The cost function J represents the performance of the architecture A on the validation dataset D_{val} having been trained on the dataset D_{train}. Examples of D_{train} and D_{val} include CIFAR-10 [26], Imagenet [44] and many more. In contrast, hyper-parameter optimization (HPO) [5, 49] that has certain similarities to NAS, differs from it as the main objective is to find the set of optimal hyper-parameters $\theta^*(A)$ for an architecture A over the set of all possible hyper-parameters Θ.

$$\theta^* = \underset{\theta \in \Theta}{\text{argmin}} J(\theta, A; D_{val}) \tag{1.2}$$

1.1 Problem

A typical NAS algorithm draws network samples from the search space, trains the network, and updates its parameters based on the resultant performance [12]. Most NAS algorithms [13, 22, 57, 62] require a large number of samples to reach the neighbourhood of the best network in the search space, while some algorithms are time-consuming and computationally expensive [3, 62]. With limitations in hardware and time constraints, such methods are not viable. Bayesian optimization (BayesOpt or BO) is a highly preferred optimization method for non-convex, non-linear, black-box functions that are expensive to evaluate. The derivatives of such functions are not accessible. BayesOpt is known for its ability to solve such problems with as few samples as possible. This makes it suitable for NAS. However, BayesOpt is mostly used in the lower dimension (Euclidean spaces) and cannot be directly applied to the space of neural nets. Therefore the question that arises is, if it is possible to develop a BayesOpt approach that can handle high dimensional space while still satisfying the hardware and time constraints.

1.2 Objective

For the application of BayesOpt in the search space of neural networks, it is necessary to encode the network data into a lower dimension that the model can interpret. Graph kernels [25] are functions that operate on graphs and produce encoded vectors. Since we want to model the performance of neural networks by sampling and training only a small number of models, an assumption can be made that, architectures with similar performances are in the neighbourhood of each other. This means that there is an n-dimensional space, where the distance between architectures with similar performance is minimal. This assumption is valid as many architectures that have similar performance measures are usually modifications of each other. The objective of this research can be formulated as :

1. Is it possible to implement a graph kernel that correlates networks in the search space based on their performance?
2. If such a kernel is possible, then can it be used with BayesOpt?
3. How should the BayesOpt be designed to satisfy the constraints?

This thesis presents three different graph kernels that are capable of encoding the graph data. While the concept of applying graph kernels is not new, we present a novel choice for the kernel[1] which we show is better than previous methods. We also explore the various parameters such as sampling and acquisition functions associated with BayesOpt and try to find the right combination, that requires fewer samples and produces a good result.

1.3 Methodology

The objective of this thesis is to sample the best network from the NAS-Bench-101 dataset by observing as few samples as possible. The best network is the network with the least test error when evaluated on the CIFAR-10 [26] dataset. The success of a NAS algorithm primarily depends on how well it compares to the random search [6]. Algorithms that cannot find a better architecture than the random search at the end of the training duration are considered failures. While this metric is generally

[1]At the time of starting this project no approaches were present for the use of WL and GCN kernels in BayesOpt. A recent paper [43] follows a similar approach.

preferred, a qualitative approach can also be considered especially for determining the performance of a graph kernel. Here the ability of the kernel to group similar networks is estimated approximately. It is assumed that a good kernel can segregate the good and bad networks based on their test error and therefore have a higher chance of identifying the best network.

1.4 Limitations

The scope of this project is limited to finding the best model in the NAS-Bench-101 [60] search space in as few samples as possible while placing a constraint on the hardware (number of GPUs) used. No attempt is made to use the model as a network performance estimator. The performance on other datasets such as NAS-Bench-201 [11] or DARTS [32] are left for future work.

1.5 Outline

- Chapter 2 provides the necessary information to understand the various concepts and theories discussed. It also provides a literature review of previous works related to NAS.
- Chapter 3 describes the graph kernels implemented in detail.
- Chapter 4 discusses the implementation details of the entire process. Various experiments carried out using acquisition on sampling functions are described here. It also introduces the NAS-Bench-101 dataset.
- Chapter 5 presents the results obtained and provides observations related to them.
- Chapter 6 discusses the possible impacts of this thesis and future works that could be carried out.

Chapter 2

Background

This chapter provides the necessary information to understand and interpret the theories and results presented in this project. This covers an introduction to Bayesian optimization, beginning with the Bayes' theorem followed by an overview of the optimization process. Previous works regarding research on NAS and the current state-of-the-art models are discussed at the end.

2.1 Bayesian Optimization

A typical optimization problem can be defined as

$$\min_{x \in S} f(x) \tag{2.1}$$

where, the input x is a vector ($x \in \mathbb{R}^d$) belonging to the set S. The objective function f is a continuous function that maps the samples in the set S to a real value. It is required to find the value of x that minimizes f. When f is convex, the *local* minimum is also the *global* minimum. In most cases however f is non-convex, non-linear and expensive to evaluate. This makes the task of finding the global minimum a difficult one. In many cases it is also difficult to represent the objective function f mathematically making it hard to compute its derivatives. Also there is a chance that the function f is not noise-free and hence the actual evaluated result is $f(x) + \eta$. Hence it is necessary to consider the function f as a *black-box* function and minimize it without taking the actual function into consideration. The noise is usually approximated as a zero centered normal distribution $\mathcal{N}(0, \sigma^2_{\text{noise}})$.

While there are many algorithms to solve such black-box optimization problems [28, 61], most algorithms require a large number of samples and are sometimes time-consuming. Bayesian optimization [7, 36] is a powerful algorithm that builds a prior function over the objective and combines it with evidence from evaluating the function f in order to get the posterior function that tries to model the objective. This approach is preferred over the other algorithms as the number of samples evaluated is much smaller. Bayesian optimization has been used for optimization of non-convex function since the 1960's [27, 35]. It has gained much significance in the past decade due to its application in hyper-parameter tuning of machine learning algorithms [5, 49]. In this section, the main motivation behind Bayesian optimization and the steps involved in the process are described.

2.1.1 Bayes' Theorem

Bayes' theorem [20] related to the context of optimization can be stated as -"the posterior probability of a model (or theory, or hypothesis - $\mathcal{M}$) given evidence (or data, or observations - E) is proportional to the likelihood of E given $\mathcal{M}$ multiplied by the prior probability of $\mathcal{M}$" [7, p. 2].

$$P(\mathcal{M}|E) \propto \underbrace{P(E|\mathcal{M})}_{\text{Likelihood}} \underbrace{P(\mathcal{M})}_{\text{Prior}} \tag{2.2}$$

The equation 2.2 provides a method to validate the priors or *beliefs* about the objective function, when given samples from the search space and their evaluation via the objective function. Let $\{x_1, x_2, \ldots, x_t\}$ be samples accumulated from the set S over t time steps and let $\{y_1, y_2, \ldots, y_t\}$ be their evaluations from the objective function. Collectively the samples and their evaluations make up the sequential data D, $D_{1:t} = \{x_1, y_1, \ldots, x_t, y_t\}$, which is used to make prior assumptions about the objective function. The term $P(D_{1:t}|f)$ can be interpreted as - given the priors about the objective function, how likely is the data to be observed. Since the process is sequential, as t increases, the posterior get updated as shown below. Note that since the posterior is marginalized over all samples, the proportionality can be ignored. The posterior function which tries to model the objective is often known as *surrogate* function ($\mathcal{M}$) or *response surface*.

$$P(f|D_{1:t}) = P(D_{1:t}|f)P(f) \tag{2.3}$$

While there are many choices for the distribution of the priors such as the Wiener process [27] or the Tree-Parzen-Estimator [5], Gaussian process (GP) [36, 61] is the most preferred due to its versatility and ease of computation. A multivariate Gaussian distribution can represent most continuous functions. The theory behind GP is presented in later sections.

2.1.2 Bayesian Optimization Approach

As shown in the previous section, Bayesian optimization is an iterative approach. In each time step t, a new sample is selected from the search space and evaluated. This selection process is carried out by the *acquisition* function, which selects the candidate sample with the help of the surrogate function $P(f|D_{1:t-1})$. Acquisition functions direct the search process by selecting those samples which it determines to be helpful for the minimization process. Typically there is a trade-off between exploration and exploitation of the search space. There are many different acquisition functions such as expected improvement [19], lower confidence bound [10], and Thompson sampling [53], to name a few. These functions are explained in detail in Section 4.4.

Algorithm 1: Step by step Bayesian optimization

```
1 sample $n$ points at random from $S$;
2 evaluate the points $(y_1, \dots y_n)$;
3 initialize the GP with observed samples and values;
4 for $t = 1, 2, \dots,$ do
5     get $x_t$ from the search space using the acquisition function;
6     evaluate the sample $x_t$, $y_t = f(x_t) + \eta$;
7     Augment the dataset $D_{1:t} = \{D_{1:t-1}, (x_t, y_t)\}$;
8     Update GP surrogate with $D_{1:t}$
9 end
```

Take the case of a simple 1D, non-convex, non-linear function such as the 1-D *Ackley* function which is defined in Equation 2.4.

$$y_{\text{ackley}}(x) = -20e^{-0.2\sqrt{0.5x^2}} - e^{0.5\cos 2\pi x} + e + 20 \tag{2.4}$$

In order to model this function to find the minima, we initially sample two points and fit their values along with their evaluations to the GP model resulting in Figure 2.2. It can be seen that with just two samples, the GP is unable to model the objective function and is uncertain in its prediction in most of the areas except at the sampled points. It

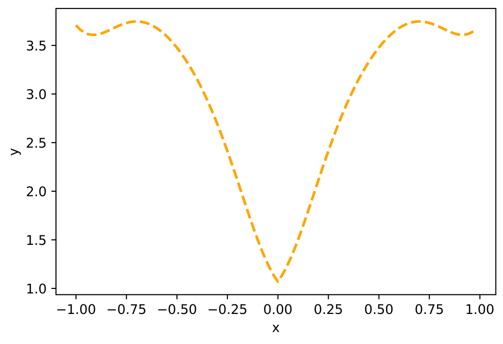


Figure 2.1: Noise-free Ackley function, where $x \in \mathbb{R}$, $-1 \leq x \leq 1$

should be noted that the region of uncertainty is much smaller around a sampled point as compared to other places due to the inherent understanding that points that are closer have similar results. When more samples are acquired, the surrogate becomes

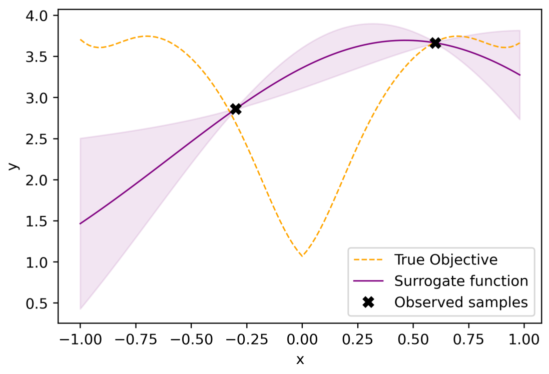


Figure 2.2: Bayesian optimization with GP using RBF kernel is applied for the minimization of Ackley function. Samples are marked by "x". The unbroken violet line represents the surrogate function (current model of the objective function) while the shaded regions represent uncertainty in predictions.

closer and closer to the objective function, albeit with noise. The ability of Bayesian optimization to estimate the objective functions with very few samples (compared to other optimization algorithms) can be seen in Figure 2.3, where the GP surrogate is close to the objective with just a total of 6 samples. In the right side, the acquisition function (LCB) is plotted. The acquisition function tries to direct the search towards the minimum and selects a candidate to be sampled. Initially, it explores the space, but once it samples a point from the neighbourhood of the minimum, it becomes exploitative and repeatedly samples around this point in order to reach the minimum.

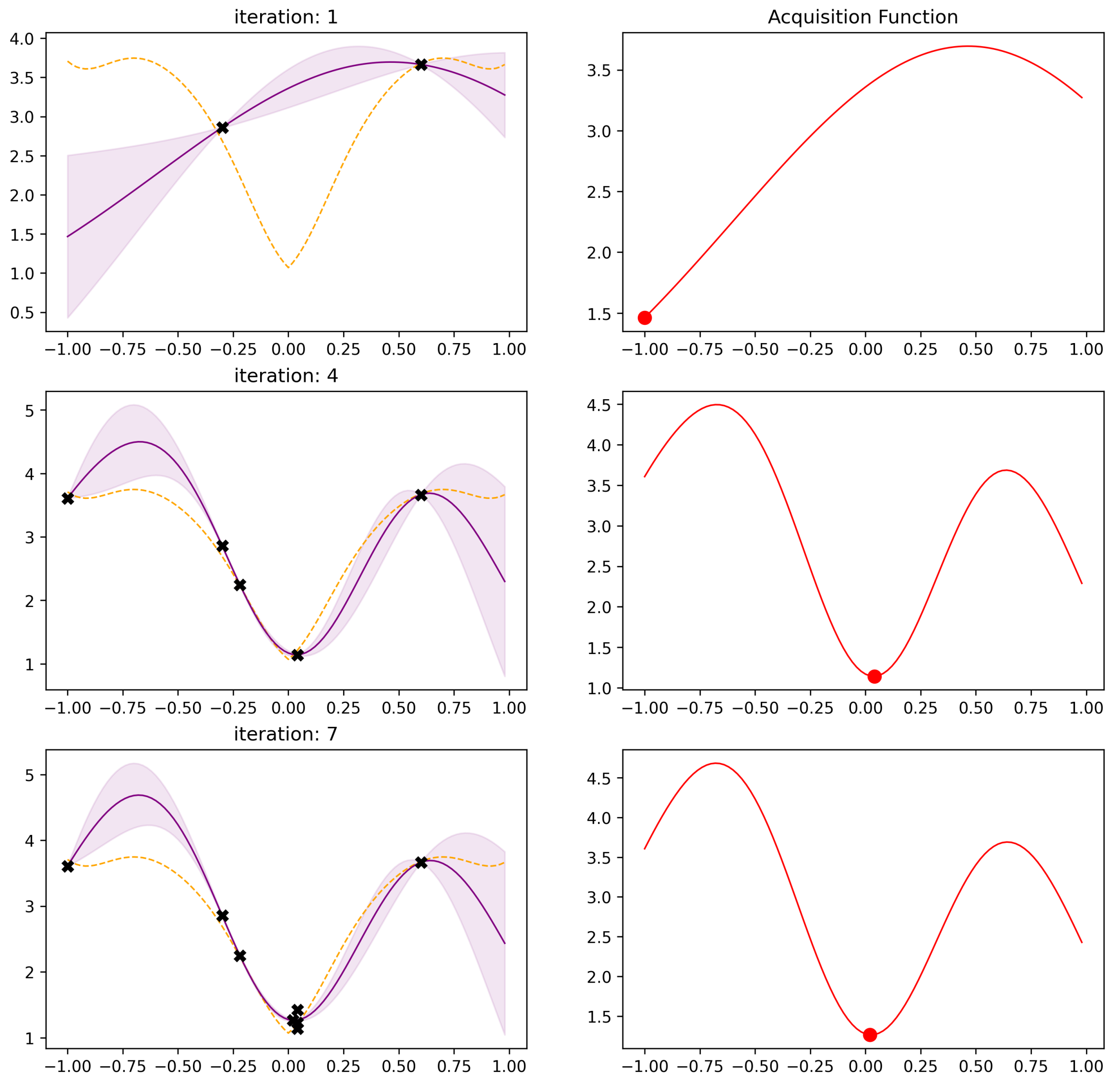


Figure 2.3: Evolution of Bayesian optimization over time. The regions of uncertainty decrease as more points are sampled. In the right the working of the acquisition function is plotted. The red dot indicates the location of the candidate sample.

2.1.3 Gaussian Process

A Gaussian process (GP) is used to describe a distribution over a set of functions. A GP is a collection of stochastic samples where every sample is represented as a multivariate Gaussian distribution ($\mathcal{N}(\vec{\mu}, \Sigma)$) and the joint distribution over these samples gives the distribution of the GP. A GP is completely defined by its mean function $m(x)$ and covariance $k(x, x')$. The objective function when modelled as a GP is written as

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{2.5}$$

Figure 2.4 depicts the GP for the example of Ackley function from the previous section. Given two samples with their evaluations (mean) and the covariance function k (RBF), it can be seen that the GP generates a set of possible functions that satisfy the given criteria. Over time, when more points are sampled, the uncertainty reduces, thereby restricting the space for the priors.

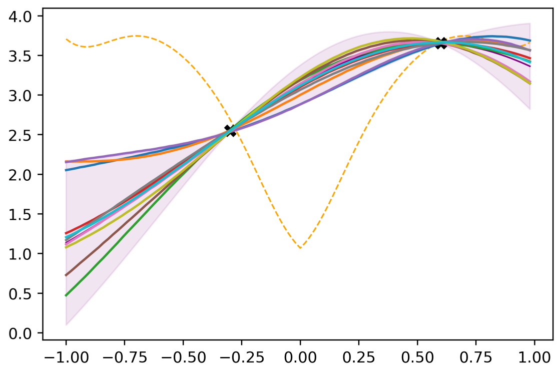


Figure 2.4: Few of the possible priors for the objective function. Each of the Gaussian functions have the same mean and covariance.

Applying a Gaussian process as priors to Bayesian optimization is often called *kriging*. The kernel function is used to generate a covariance matrix (**K**), which comprises of pairwise correlation between n observed samples. When the model is noise-free, then the diagonal values are 1. A requirement for the kernel is that **K** should be positive semi-definite.

$$\boldsymbol{K} = \begin{bmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \vdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{bmatrix} \tag{2.6}$$

Since the $k(x, x')$ is independent of $k(x, x'')$, the kernel matrix can be easily updated for time $t+1$ as follows [7]:

$$\mathbf{K}_{t+1} = \begin{bmatrix} \mathbf{K} & \mathbf{k}_{t+1}^T \\ \mathbf{k}_{t+1} & k(x_{t+1}, x_{t+1}) \end{bmatrix} \tag{2.7}$$

$$\text{where, } \mathbf{k}_{t+1} = \begin{bmatrix} k(x_{t+1}, x1) & \cdots & k(x_{t+1}, x_t) \end{bmatrix}$$

If at time t, data $D_{1:t}$ has been observed, it is possible to predict the evaluation of the objective function at point x_{t+1}. The values for $\mu_t(t+1)$ and $\sigma_t^2(t+1)$ can be obtained with the help of the Sherman-Morrison-Woodbury formula [7, 40] as shown:

$$\begin{aligned} \mu_t(t+1) &= \mathbf{k}^T \mathbf{K}^{-1} \mu_{1:t} \\ \sigma_t^2(t+1) &= k(x_{t+1}, x_{t+1}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k} \end{aligned} \tag{2.8}$$

Covariance functions

The covariance or kernel function $k(x, x')$, controls the shape and smoothness of the GP. A kernel is used to find the correlation between the different samples observed in order to fit the Gaussian model and predict new values. Two samples which are close together or similar can be expected to have a large correlation and *vice-versa* two dissimilar points could be expected to have small correlation. There are many kernel functions [36, 40] of which two important ones are the *squared exponential* or Radial Basis Function (RBF) [36] kernel and the Matern kernel [33].

The RBF kernel is written as

$$k(x, x') = \exp\left(-\frac{||x - x'||^2}{2l^2}\right) \tag{2.9}$$

where $||x - x'||$ is the Euclidean distance between x and x' and l is a hyper-parameter which scales the distance thereby controlling the width of the kernel and its smoothness. This kernel is infinitely differentiable, which means that GPs are very smooth.

The Matern kernel is a generalization of the RBF kernel where additional parameters are included to control the smoothness of the function. It is defined as

$$k(x, x') = \frac{1}{\Gamma(\nu) 2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{l} ||x - x'||^2\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{l} ||x - x'||^2\right) \tag{2.10}$$

K_ν is the modified Bessel function of order ν [2] and Γ is the gamma function [1]. The parameter ν controls the smoothness of the function. Unlike the RBF kernel the Matern kernel is only $\nu - 1$ times differentiable.

2.2 Neural Architecture Search

Architecture engineering for deep learning is a complex and time consuming process. While simple structures are fairy easy to design, complex models such as InceptionNet [52], ResNet [16] or DenseNet [18] require lots of intuition and are perfected through trial and error. Neural Architecture Search (NAS) was introduced in order to provide a systematic approach for architecture design where rather than training each iteration of the network design, networks are selected based on the statistical data of the network search space. NAS algorithms primarily depend on three main sections : search space, search strategy and performance estimation.

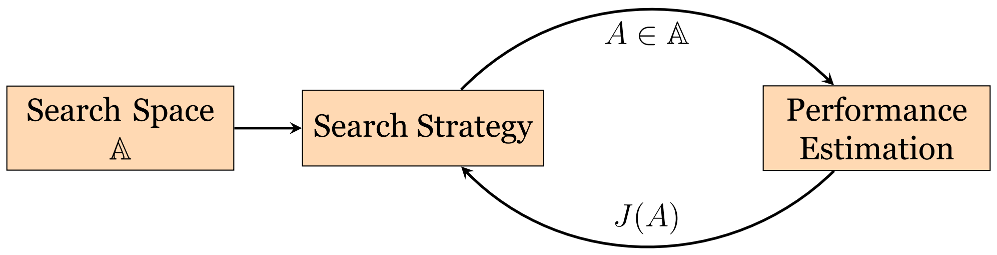


Figure 2.5: General overview of NAS algorithms [12].

2.2.1 Search Space

The search space ($\mathbb{A}$) refers to the set of all possible architectures that can be applied to a dataset. The search space is usually subjected to constraints such as the maximum number of nodes or edges that are allowed in a network [32, 60]. Search spaces can be classified into two main sections - direct and indirect. In direct search spaces (Figure 2.6 (a)), the networks are a sequence of layers that can have complex connections amongst the layers. Each layer L_{i-1} feeds data to the next layer L_i. The search space is constrained by the number of layers and the type of operations present, such as convolution [29] layer, pooling layer [45] and recurrent layers.

Indirect search spaces (Figure 2.6 (b)) have a predefined structure where certain operations are fixed. In between these fixed operations, a *cell* or a stack of layers are added whose design is the target of NAS. Cell-based architectures became popular after outperforming conventional structures [16, 52]. In each cell, the number of operations, number of nodes and number of connections are constrained. Most of the current datasets for NAS are usually cell-based [32, 60]. Since the cells are repeated, even a small number of nodes in a cell can produce excellent networks. This makes the cell-based approach faster and easy to search [3]. Also, since cell networks share many parameters, they can be easily transferred to other datasets and models. Most cell networks are trained on smaller data such as CIFAR-10 [26] and later transferred to ImageNet [44].

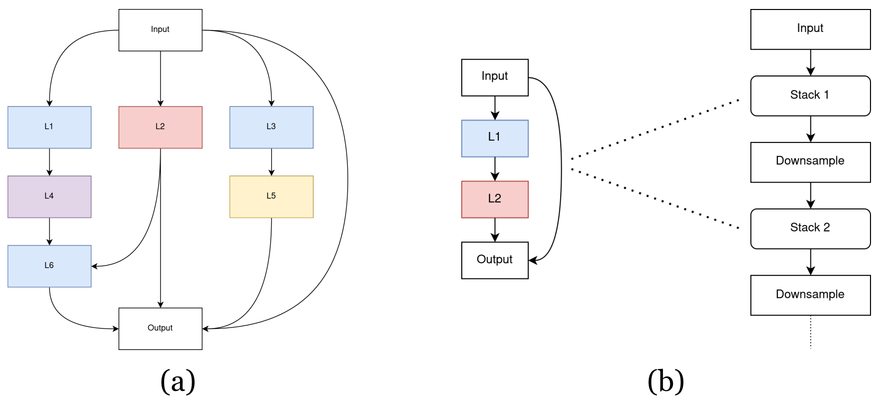


Figure 2.6: Search space representations. (a) A direct search space sample with complex connections between the layers. (b) An indirect search space with repeated stacks of ResNet cells.

2.2.2 Search Strategy

Search strategy [12] is the technique used to explore the search space and select architectures for estimation, where typically the architecture is trained and its validation accuracy is obtained. The parameters of the search strategy are updated each time a network is evaluated. Based on the estimation results, the search algorithm can decide between exploring the dataset or searching for better networks in the neighbourhood of the previously sampled ones.

Some of the search strategies used are random search [6], evolutionary search [21, 42, 50], Bayesian approaches [4, 34] and reinforcement learning based models [62]. Random search is the easiest and most naive approach, where networks are randomly sampled and evaluated. While this is similar to the trial-and-error approach, it does work surprisingly well for datasets like the NAS-Bench-101 [60], where the distribution of networks is highly skewed as most networks have 80% accuracy. While it may be the fastest approach for NAS, its reliability is questionable over different datasets. Evolutionary search methods like regularized evolution [42] sample a limited set of networks to build a population. Then from this population, networks are mutated randomly to generate sibling networks that retain some similarity to the original network. In each iteration, the oldest network in the population is removed.

Bayesian optimization (BayesOpt) has been used for hyper-parameter tuning for a long time [5, 13, 22, 49]. One of the main problems of applying BayesOpt for NAS is that typical BayesOpt methods focus on vectors, where the distance between two vectors can be easily computed. This is not applicable to high dimension data like networks (graphs) where the distance computation needs to be explicitly defined. NASBOT [22] is an approach that uses a BayesOpt with Gaussian priors. This is obtained by designing a distance metric (OTMANN) which computes the similarities between layers. Through this approach, NASBOT performed better than previous models such as random and evolutionary search methods.

RL based approaches consider the search space as the action space and model the agent, whose reward is defined by the performance of the network generated [3, 62]. These approaches usually have a sequential network such as RNN or LSTM that generate an encoded string that contains the structure of the neural network. Another approach [9] is to generate an architecture sequentially by considering a Markovian approach where the actions that been sampled previously are part of the state of the model. The reward is specified only after generating an entire network. The problem with RL based approaches is that while they are better than random search, they are computationally expensive and require lots of samples to build a model.

2.3 Related Work

The Bayesian approach discusses in this project is based on the foundation laid by NASBOT [22]. Similar to the approach used by the authors of NASBOT, specialized graph kernels are implemented. The high dimensional graph is encoded into a meaningful vector of lower dimension, that the GP can process. An issue with NASBOT is the distance metric (OTMANN), which works well in direct search spaces, but needs to be modified for cell-based networks given it suffers from over-fitting as only the small cells need to be compared rather than the entire network. Also, compared to state-of-art approaches [57, 58] it requires lots of samples to perform well.

The simplest approach for architecture search is random search or random sampling [5, 6]. While this approach has a very low chance of sampling the best architecture in a large distribution, in a skewed dataset such as NAS-Bench-101 [60], where most of the architectures have high accuracy, this method serves as the baseline for comparing NAS models. Two recent models that were developed in 2020 are *BANANAS* [57] and *Local Search* [58]. Both these methods outperform previous approaches on the NAS-Bench-101 dataset.

Bayesian optimization with neural architectures for NAS (BANANAS) is a BO approach where rather than using a conventional prior like Gaussian model, the authors use a feed-forward ensemble neural network to predict the performance of an architecture. Similar to a BO-GP approach, in each iteration, a sample is selected using the acquisition function. Every selected sample is encoded via *path encoding*, which is based on the presence of a certain path in the structure of the sample. The neural network predictor is then trained with all previously evaluated samples to predict the accuracy of unseen networks. While this approach is shown to be very effective, one drawback is the need to train the feed-forward network. Also the encoding used is rather simple and has a large dimension compared to other graph encoding schemes like the Weisfeiler-Lehman graph kernel [47]. Local search is a NAS search algorithm which is based on the method of local search optimization [38]. A network is selected at random and all its neighbours are evaluated. The process is repeated with the best network found. The simplicity of this method is due to the fact that it is very similar to the random sampling approach. Local search has been shown to have similar performance or even slightly outperform even BANANAS on NAS-Bench-101. While, this is true of other small datasets like NAS-Bench-201 [11], the authors state that this method fails for large datasets like DARTS [32].

Chapter 3

Graph Kernels

This chapter provides details on how each deep learning architecture is encoded into a lower dimensional vector used to build the Gaussian model. This is done by specialized *graph kernels*. The three graph kernels that were implemented are discussed here.

3.1 Overview

The structure of a neural network can be well represented by a graph G defined by its vertices and edges such that $G = (V, E)$. The nodes (V) represent the mathematical operations (layers) and the edges (E) indicate the connections between the various layers. Typically, the nodes are given a label (σ) to identify them based on the type of operation uniquely. As shown in Figure 3.1, each architecture $A_n \in \mathbb{A}$ can be represented as a graph $G_n = (V_n, E_n)$. Here each operation is given a unique label and colour.

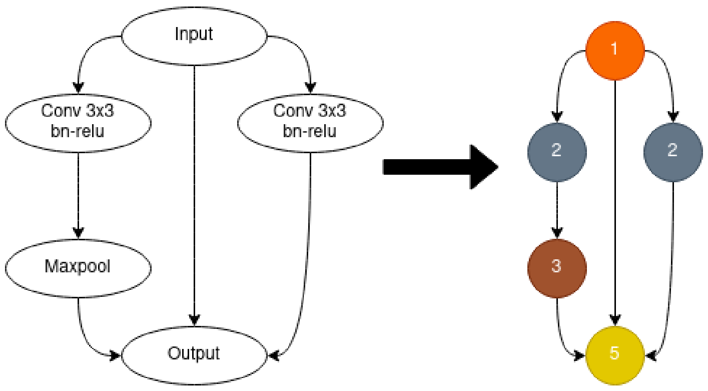


Figure 3.1: A deep learning model is represented as a labelled graph.

Although informative, this graph representation cannot be directly processed by conventional kernel functions like the RBF kernel or the Matern kernel, as they usually expect a vector input. Therefore it is necessary to convert the graph representation into a vector representation. This vector representation is then used to obtain the covariance between two graphs. The process of transforming the graph into a vector is known as *graph encoding*. A major concern in graph encoding is the loss of information when moving from a high dimensional representation to a lower dimension. Hence there is a need for specialized kernels that can encode graphs while retaining the information about the graph structure. Three such kernels - the Weifeiler-Lehman subtree kernel, edge kernel and GCN kernel were implemented. The encoding process is depicted in Figure 3.2.

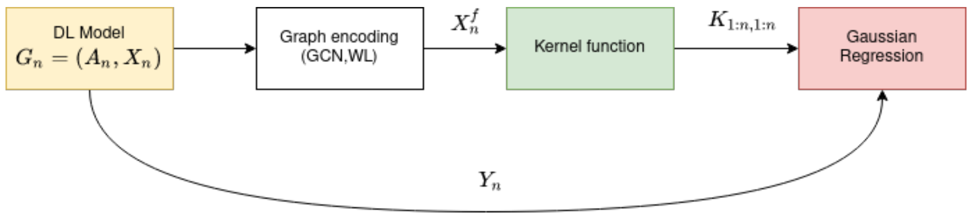


Figure 3.2: Graph encoding and kernel computation.

Figure 3.3 depicts the entire workflow of the BO process used in this thesis. In every iteration, a set of sample graphs are drawn from the search space. Amongst them, one graph is selected using the acquisition function. The graph kernels encode the selected graph into a vector, and this vector is used to compute the covariance values using the kernel functions. These values are then passed to the Gaussian regressor which updates its belief using the observed samples and the process is repeated. Initially, few samples are drawn at random in order to build the prior for the Gaussian model.

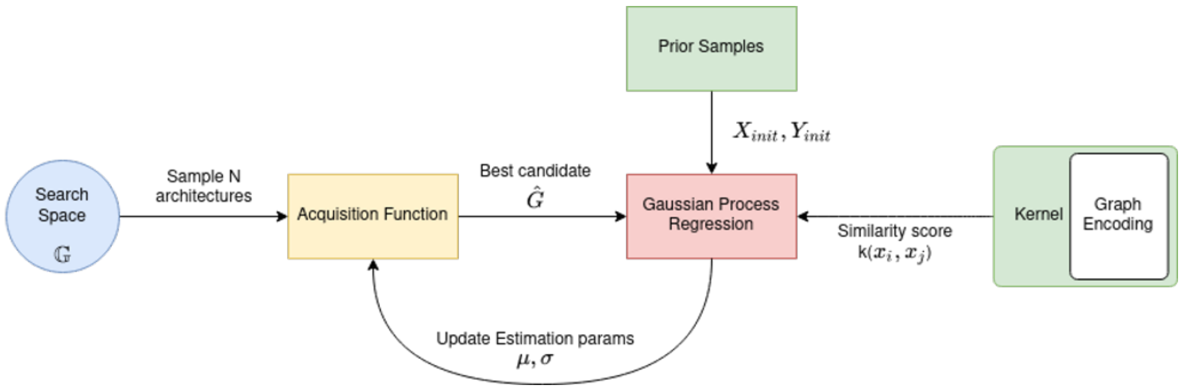


Figure 3.3: Workflow of the Bayesian optimization process.

3.2 Weisfeiler-Lehman Subtree Kernel

The Weisfeiler-Lehman (WL) subtree kernel [25, 46, 47] is a graph kernel based on the Weisfeiler-Lehman test of isomorphism [56]. The WL test for isomorphism provides a method to obtain a canonical representation of a graph G such that the node labels and their connections are preserved. The WL process is an iterative process that captures the structure of a graph by combining the information of a node of a graph with all its neighbourhood nodes. Consider two directed graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ as shown in Figure 3.4. If Σ (label map) denotes the set of all possible labels (σ) for a set of N graphs with a total of l nodes, let $\Sigma_0 \subset \Sigma$ denote the set of node labels occurring at least once at the start of the process. Similarly, let $\Sigma_i \subset \Sigma$ indicate the set of labels occurring at least once at the end of iteration i including the previous iteration. In the given example, $\sigma \in \mathbb{Z}^+, \Sigma_0 = 1, 2, 3, 4, 5$.

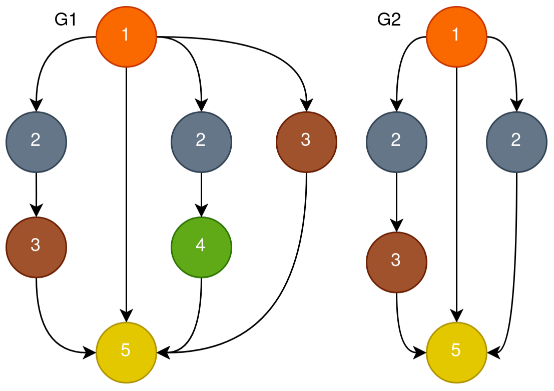
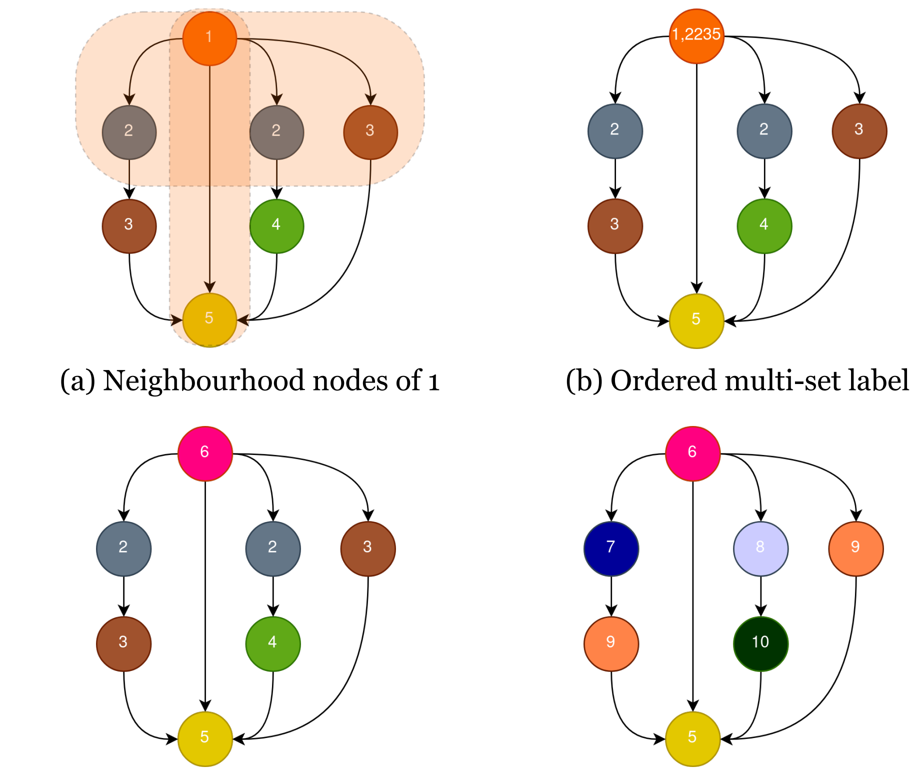


Figure 3.4: Graphs $G_1$and G_2, where the colours and labels uniquely identify a node.

In each iteration i, for every node v_n^j whose current label is σ_{ij}, the labels of its neighbourhood nodes are collected and grouped as shown in Figure 3.5 (a). Once the labels are collected, they are sorted and appended to the original label to create a new label, called the *multi-set label* ($\hat{\sigma_{ij}}$) as seen in Figure 3.5 (b). The multi-set label is then compared with the current label set Σ_{i-1} and in the event that it does not exist in Σ_{i-1}, the multi-set label is compressed to create a unique label ($\bar{\sigma_{ij}}$) that follows the order of the label set and assigned to the node in a step called “relabelling” (Figure 3.5 (c)). This step is repeated for all nodes in all graphs over the common set Σ_{i-1} leading to the creation of a new set $\Sigma_i = \Sigma_{i-1}, \bar{\sigma_{i1}}, \bar{\sigma_{i2}}, ..\bar{\sigma_{il}}$ where all necessary nodes are relabeled as shown in Figure 3.5 (d). The WL process for node 1 of G_1 for the first iteration can also be shown numerically as follows:

$$
\begin{aligned}
\Sigma_{i-1} = \Sigma_0 &= \{0:0, 1:1, 2:2, 3:3, 4:4, 5:5\} \\
\text{Grouping} &= (1-2, 5, 2, 3) \\
\text{Sorting} &= (1-2, 2, 3, 5) \\
\text{multi-set label - } \hat{\sigma_{11}} &= (1, 2235) \\
\text{Relabelling} &= \hat{\sigma_{11}} \notin \Sigma_0, \implies \bar{\sigma_{11}} = 6 \\
\Sigma_1 = \{\Sigma_0, \bar{\sigma_{11}}\} &= \{0:0, 1:1, 2:2, 3:3, 4:4, 5:5, 1, 2235:6\}
\end{aligned}
$$

Once all graph nodes are processed, the next graph undergoes the same steps with the updated label set. Figure 3.6 depicts the updated node labels at the end of the first iteration. The process is repeated for h iterations, where h denotes the height of the kernel.



(a) Neighbourhood nodes of 1 (b) Ordered multi-set label

(c) Label compression and relabelling (d) Updated labels

Figure 3.5: Step by step WL process applied to Graph 1. Here the labels are represented by natural numbers. Note that since the graph is directed, 5 has no neighbours and hence remains unchanged.

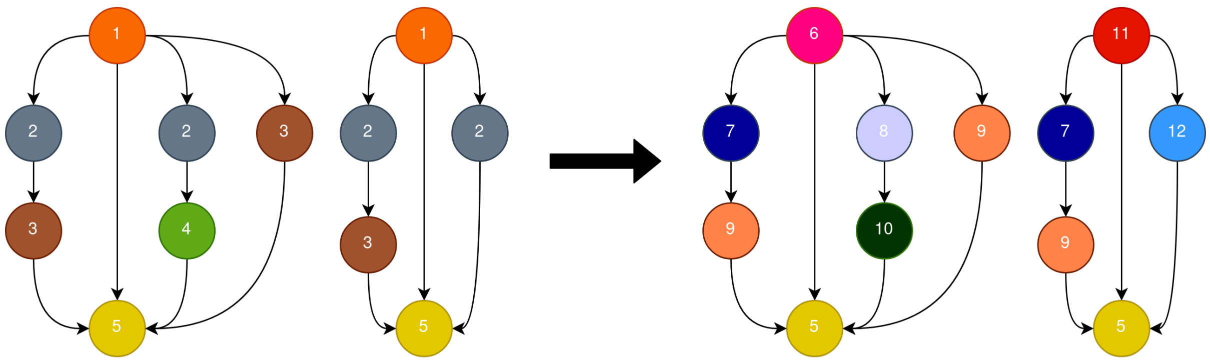


Figure 3.6: Updated node labels after relabelling process is applied to both the graphs.

It can be seen from Figure 3.5 (c) and (d) that the compressed labels inherently contain the structure of the graph. For example, consider the label 6, which explicitly denotes a pattern where the node with label 1 is connected to two nodes with label 2, one node with label 3 and one node with label 5. Here the information from the second level (node-set: 2,2,3) is passed to the first level. In the second iteration, as shown in Figure 3.7, a new multi-set label $6,5789 \to 13$, is created. This label is made of encapsulated labels 6, 7, 8 and 9 (each of the labels contain information about their neighbours - 6:1,2235, 7: 2,3, ...). It can be observed that the new label contains almost the entire structure of the graph, as the label 13 contains information about all the four levels of the graph. In every iteration, the information about subsequent levels is passed to the nodes in the previous levels.

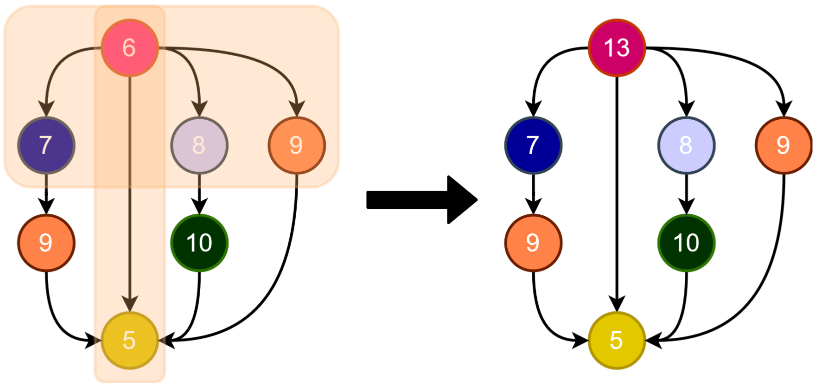


Figure 3.7: Relabelling step for Node 6 in iteration 2.

Once the relabelling is completed for h iterations for all N graphs, the kernel needs to be defined to compute the covariance between the graphs. In order to do so, the *feature vector* (Φ_{WL}) for a graph G is defined as the count of node labels present in each graph. The WL subtree kernel can be summarized as a graph kernel that encodes a graph into a multi-set label count vector.

$$\Phi_{WL}(G) = \{c_0(G, \sigma_{01}), ..c_0(G, \sigma_{0l}), ...c_h(G, \sigma_{hl})\} \tag{3.1}$$

where c_i denotes the function that counts the occurrence of a label σ (original and compressed) at the end of iteration i for graph G. In Table 3.1, the values in each column represents the count of each label in Graphs G_1 and G_2 at the end of the first iteration. The kernel, a function of these feature vectors is simply defined as the inner product of the corresponding feature vectors.

$$k_{WL}(G, G') = \langle \Phi_{WL}(G), \Phi_{WL}(G') \rangle \tag{3.2}$$

Labels→	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$\Phi_{WL}(G_1)$	1	2	2	1	1	1	1	1	2	1	0	0
$\Phi_{WL}(G_2)$	1	2	1	0	1	0	1	0	1	0	1	1

Table 3.1: Feature vectors at the end of iteration 1 for graphs G_1 and G_2.

The rows of Table 3.1 denotes the feature vectors obtained for graphs G_1 and G_2. The kernel function is used to build the $N \times N$ kernel matrix which normalized such that $k_{WL}(G, G') = 1 \rightarrow$ indicates maximum correlation (the same graph) and $k_{WL}(G, G') = 0 \rightarrow$ indicates an absence of any relationship between the graphs. One of the major advantages of this kernel is its label compression which intuitively provides information about the graph structure which results in a higher accuracy in identifying similar graphs and predicting covariance at relatively cheaper computation as compared to other approaches such as random walk [14, 23] or deep learing approaches that require lots of samples [15, 39]. In general, the WL process is applied to every pair of graphs present in the dataset resulting in the formation of a $N \times N$ kernel matrix. While a global label set could be maintained across all operations, it would be a time-consuming process as the feature vector for each graph needs to be determined sequentially. Also, the size of the label set Σ would increase exponentially. This can be avoided by utilizing the fact that the covariance between two graphs is not affected by any external factor, i.e. $k_{WL}(G, G')$ is independent of the computation $k_{WL}(G', G'')$. Hence for large N, the values of the kernel matrix can be computed in parallel.

If all labels are unique at the start of the process, then the set Σ_0 at the most has l labels equaling the total number of nodes. For every iteration, a maximum of l labels can be added to the current set, while the number of multi-set grouping operation depends on the number of edges present ($|E|$) in each graph. Hence the runtime complexity of the process in naive application for N graphs at height h is $\mathcal{O}(Nh|E| + N^2hl)$.

3.3 Naive Edge Kernel

The naive Edge kernel, based on [25, 47], is a straight forward kernel where the feature vector $\Phi_E(G)$ of a graph G is simply the count of different edge labels present in the graph. An edge e, connecting nodes a and b, $a, b \in V$, forms an ordered node pair (a, b). Each unique node pair is assigned a label σ_E^i, where $\sigma_E^i \in \Sigma_E$ which denotes the set of all edge present across N graphs. Figure 3.8 shows the edge labels for the two graphs G_1 and G_2 from section 3.2.

Similar to the WL subtree kernel, the edge kernel is defined as the inner product of the graph feature vectors and the kernel matrix is normalized between 0 and 1. Table 3.2 depicts the feature vectors of graphs G_1 and G_2.

$$k_E(G, G') = \langle \Phi_E(G), \Phi_E(G') \rangle \tag{3.3}$$

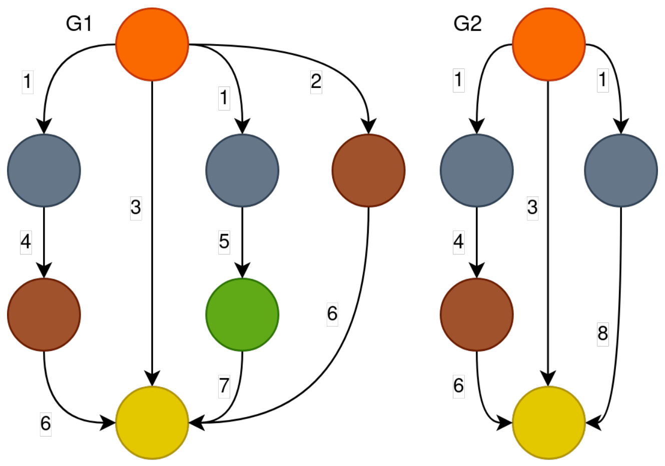


Figure 3.8: Graphs G_1 and G_2, where each unique node pair is assigned a label $\sigma \in \mathcal{Z}$. Here the node colour is used to differentiate the various nodes and thereby node pairs.

Like the subtree kernel, it can also be computed in parallel due to the mutually exclusive nature of the kernel matrix. While typical edge kernels follow an iterative approach like the subtree kernel undergoing various steps such as multi-set label creation and relabelling, in this project, the focus is more on the naive version (i.e h = 0) in order to develop a model with minimal runtime as compared to the subtree kernel. Due to this restriction, the runtime complexity for obtaining the kernel matrix becomes $\mathcal{O}(N^2|E|^2)$.

Labels$\rightarrow$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\Phi_E(G_1)$	2	1	1	1	1	1	1	0
$\Phi_E(G_2)$	2	0	1	1	0	1	0	1

Table 3.2: Feature vectors for Edge kernel

3.4 Graph Convolution Network

While kernels like WL subtree are effective in encoding graphs with complex connections, they sometimes fail at simple graphs like a chain where the height h needs to be large in order to capture the entire graph. Also the computation of multi-set labels and relabelling bottlenecks the entire BO process.

Graph Convolution Networks [24] (GCN) are basically neural networks which can operate on graphs. They are more expressive than ordinary graph kernels and can obtain better results in a variety of different tasks such as node classification, node prediction and network compression [8, 24, 59]. The main idea of GCN is collecting both self features and neighbour features. For GCN, the graph is fed as an input in the form of its adjacency matrix (A) of size $|V|$ x $|V|$ (number of nodes) and its feature description matrix (X_f) of size $|V|$ x $|L|$, where L denotes the set of node labels. X_f is usually a one-hot encoded matrix where the presence of a label at a node is indicated by 1 and absence by 0. With respect to the NAS-Bench search space, $|V|$ = 7 and $|L|$ = 6 (including null operation).

The layer wise propagation of a GCN layer in a network with j layers is defined as

$$f_{\text{gcn}}(H^{(j)}, \mathrm{A}) = \sigma\left(\mathrm{A}H^{(j)}W^{(j)}\right) , \tag{3.4}$$

where, $H^{(j)}$ is the input to the node (node feature matrix at first layer), $W^{(j)}$ is the weight matrix for layer j and σ (not to be confused with the standard deviation) is the non-linear activation function. The above equation states that in a feed-forward system, the network accumulates the weighted feature vectors of its neighbours. To make it more effective the adjacency matrix needs to include self-loops to collect its own features and then be normalized to prevent the exploding gradients. A diagonal matrix $\boldsymbol{D}$ can be used to normalize the matrix A, where $\boldsymbol{D}$ is the degree matrix (a diagonal matrix, where the values on the diagonal represent the number of nodes connected to a particular node).

$$\mathrm{A}_{norm} = \boldsymbol{D}^{-1}\mathrm{A}\boldsymbol{D} \tag{3.5}$$

Then, the updated output of a layer j is given as

$$\begin{gathered} f_{\mathrm{gcn}}(H^{(j)}, \mathrm{A}) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{\mathrm{A}}\hat{D}^{-\frac{1}{2}}H^{(j)}W^{(j)}\right), \\ \hat{\mathrm{A}} = \mathrm{A} + \boldsymbol{I} \end{gathered} \tag{3.6}$$

$\boldsymbol{I}$ being the Identity matrix. The number of layers in a GCN model indicates the distance the node features can travel. A GCN model typically has 2-3 layers of the GCN operation in order to obtain the best performance [24]. The GCN model used in the kernel operation is shown in Figure 3.9. Every layer has 256 node output features. After pooling, each graph is represented an encoded vector of length 256. The encoded vector is then passed on to a simple RBF kernel function (section 2.1.3). In this approach, the network is not trained and is only used as a feed forward network with random initialization. The reason that this works is because GCN's are basically parameterized WL subtree kernels as both focus on node-neighbour aggregation.

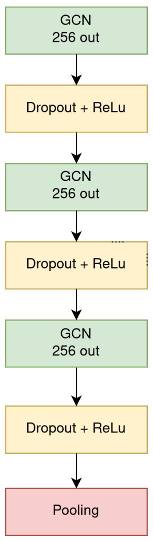


Figure 3.9: Structure of GCN layer used. Pooling aggregates node level outputs and provides outputs at graph level.

Chapter 4

Experiments

This chapter discusses the dataset used and the details on how the idea was implemented and various experiments carried out. It covers information about the various sampling strategies used to obtain a subset of networks, the various acquisition functions tested out, and some general information regarding the practical implementation that can help in reproducibility.

4.1 NAS-Bench-101

The dataset used throughout for evaluating the model and comparing results is the NAS-Bench-101 [60]. It consists of around 423,000 architectures based on the cell network, as seen in Figure 2.6 (b). Each cell can consist of a maximum of seven nodes and nine edges. Of the seven nodes, the first node is always the input and the last is the output. The remaining five nodes can take the following operations: 1 x 1 convolution followed by batch normalization, 3 x 3 convolution followed by batch normalization and 3 x 3 max pool.

Every network is trained on the CIFAR-10 dataset for image classification. Each network is trained for four different epochs 4/12/36/128 for three repeats each. This means that it is not necessary to train the model since the results are already provided. This helps in quick prototyping and prevents the need for expensive hardware for training. Each architecture is represented by its adjacency matrix (A) and a label vector (L) which specifies the operations present in it and its performance in terms of training, validation and testing accuracy including training time. NAS-Bench-101 is a highly

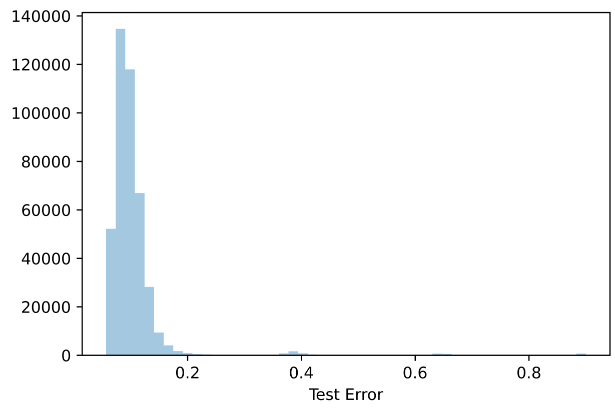


Figure 4.1: Distribution of NAS-Bench-101 test error over all samples.

skewed dataset (Figure 4.1) where most of the networks have a low error. It is an open-sourced benchmark dataset that provides a common ground for reproducibility and comparison.

4.2 Kernels

Each graph is considered to be undirected in order to increase the information collected by the kernels. During the architecture search, the GP was fit with validation error scaled by a factor of 10. Since the kernel matrix values are mutually exclusive, it is possible to run the kernel function for each pair in parallel. This is only necessary when $N >> 5000$. For all values of $N < 5000$, vector computation provides the fastest result.

4.3 Sampling

The Bayesian optimization process depends on choosing relevant samples from the sample space to update the Gaussian model. While this is not an issue with datasets where the search space is small, it becomes a computationally expensive task for datasets such as NAS-Bench-101 that has roughly 420,000 samples given that the kernels have runtime complexity in terms of $\mathcal{O}(N^2)$. This makes it necessary to sample a small subset from the search space ($\bar{\mathbb{A}} \subset \mathbb{A}$), and run the optimization process on it. This approach is justified since the Bayesian optimization process is an iterative one where the sampling process is repeated for each iteration; the probability that most of the graphs are covered is high.

A network belonging to the NAS-Bench-101 search space is defined by at most seven vertices and nine edges consisting of five operations. Therefore, it is possible to generate a random adjacency matrix of suitable dimensions and allocate operations to these nodes randomly. For uniformity, every network is permitted only one input and one output node. The first node in a network is always the input and the last node is the output. Such a specification is then validated and if it exists in the dataset, it is accepted into the sampling process.

It is essential that the sampling strategy used has the right balance between *exploration* and *exploitation*. Exploring the original search space can lead to obtaining various results and prevent getting stuck in a local optimum. It provides a better generalization of the dataset by decreasing the distribution's overall uncertainty as the samples are spread apart. While on the other hand, exploitation refers to making intelligent choices based on available data like observed samples or information about the distribution of the dataset. While it is possible to prevent repetitions with the sample set $\bar{\mathbb{A}}$, this is not recommended as it would bottleneck the entire process. Three sampling strategies covering the exploration vs exploitation spectrum are discussed here.

4.3.1 Random Sampling

It is a purely exploration-based sampling approach where, in every iteration N graphs are sampled from the search space such that none of the already observed samples are among these N graphs. Let $\mathbb{X}_{obs}$ and $\mathbb{Y}_{obs}$ be the observed graphs and their respective performance metrics. The sampled set $\bar{\mathbb{A}}$ can be defined as

$$\bar{\mathbb{A}} = \{x_i \,|\, x_i \in \mathbb{A}, x_i \notin \mathbb{X}_{obs}\} \tag{4.1}$$

Every sample in the search space has an equal probability of being selected. One of the main advantages of this approach is that unlike exploitation based approaches, the probability of getting stuck in a local optima is very low. When performed for multiple iterations, there is a high chance of sampling the global optima. However, on the downside, in datasets like NAS-Bench-101 where the distribution is highly skewed, having a sampling strategy that has a uniform probability distribution would, in the worst case, take much longer to sample the global optima compared to approaches that make use of the dataset distribution.

4.3.2 Mutation Sampling

This strategy is based on the regularized evolution search strategy [41], an exploitation-based approach that tries to sample networks similar to the best network found. The main idea behind this approach is that most of the good networks differ from each other only by a small origin. Hence if one of these networks is sampled, it would be possible to reach the global optimum. Let x_{best} be the best sample observed up to the current iteration. Each of the N samples from the search space is selected in such a way that they differ from x_{best} by a maximum of one node and / or one edge and repetitions are avoided. Figure 4.2 depicts some of the mutated networks that can be obtained from the base network represent in Figure 3.1. Note that in most cases the network is pruned to remove dangling nodes and open connections. Since a single network can only provide a limited set of mutations, another approach is to derive the mutations of the top k networks. If $\Psi(x_{best})$ represent the mutation of the best observed sample, the mutation sampling approach can be modelled as follows:

$$\bar{\mathbb{A}} = \{\Psi(x_{best}) \mid \Psi(x_{best}) \in \mathbb{A}, \Psi(x_{best}) \notin \mathbb{X}_{obs}\} \tag{4.2}$$

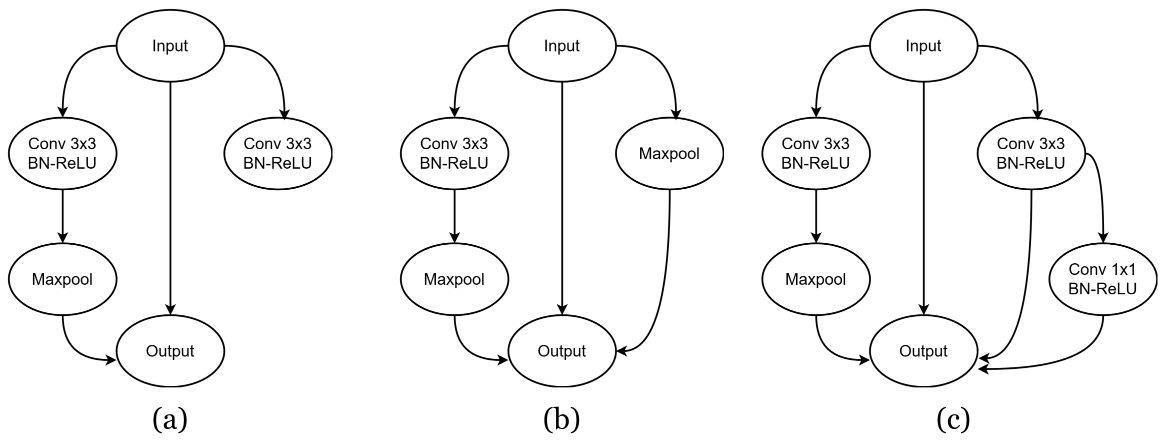


Figure 4.2: Examples of mutated architectures of network from Figure 3.1. (a) An edge is removed from the rightmost node which leaves the node open. (b) The operation of the rightmost node is modified to maxpool. (c) A new node is added to network.

4.3.3 Hybrid Sampling

This approach provides a balance between the two previous strategies. It combines both the mutation sampling and random sampling by balancing both exploration and exploitation with the help of the trade-off parameter or threshold ξ. During the sample selection process, a random number r, is generated for every sample. Depending on the value of r, a choice is made between random or mutation sampling.

$$\bar{\mathbb{A}} = \begin{cases} x_i & r \leq \xi \\ \Psi(x_{best}) & r > \xi \end{cases} \tag{4.3}$$

For highly skewed datasets such as NAS-Bench-101, the value of ξ is usually low (0.25) and favours mutation sampling.

4.4 Acquisition Functions

Once the subset $\bar{\mathbb{A}}$ is sampled, the acquisition function is required to select the best candidate(A_t) from $\bar{\mathbb{A}}$ for evaluation. This means that the network selected by the acquisition function will be trained, and the network along with its validation error is used to update the Gaussian regression model. The acquisition function makes its decision using the predicted estimation of mean and standard deviation of the networks in $\bar{\mathbb{A}}$ provided by the learned Gaussian process. Let μ_i, σ_i be the mean and standard deviation of network x_i, $x_i \in \bar{\mathbb{A}}$ predicted by the learned model ($\mathcal{M}(x_i)$). Let $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ be the vector representation of the predictions of the entire subset $\bar{\mathbb{A}}$.

$$\begin{aligned} \mathcal{M}(\bar{\mathbb{A}}) &= (\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ \boldsymbol{\mu} &= \{\mu_1, \mu_2, \dots, \mu_N\} \\ \boldsymbol{\Sigma} &= \{\sigma_1, \sigma_2, \dots, \sigma_N\} \end{aligned}$$

4.4.1 Greedy

This is a purely exploitative function which selects the sample with the lowest predicted error. There is a possibility of the function getting stuck in a local minima. The candidate is given by

$$A_t = \underset{A \in \bar{\mathbb{A}}}{\text{argmin}} \, (\boldsymbol{\mu}) \tag{4.4}$$

4.4.2 Expected Improvement

Expected improvement (EI) can indicate a measure of the magnitude of the improvement. In order to get an accurate distribution of the objective function, EI has an effective trade-off between exploration and exploitation using the parameter ξ. Typically $\xi = 0.01$. For an architecture A_i, EI(A_i) is defined as

$$\begin{aligned} D &= \mu_i - \min(\boldsymbol{\mu}) - \xi \\ Z &= \begin{cases} \frac{D}{\sigma_i} & \text{if } \sigma_i > 0 \\ 0 & \text{if } \sigma_i = 0 \end{cases} \\ \mathrm{EI}(A_i) &= \begin{cases} D\Psi(Z) + \sigma_i \psi(Z) & \text{if } \sigma_i > 0 \\ 0 & \text{if } \sigma_i = 0 \end{cases} \end{aligned} \tag{4.5}$$

where $\Psi(Z), \psi(Z)$ are the cumulative distribution function (CDF) and probability distribution function (PDF) of Z respectively. A_t is given by

$$A_t = \underset{A \in \bar{\mathbb{A}}}{\operatorname{argmax}} \left(\mathrm{EI}(A)\right) \tag{4.6}$$

4.4.3 Lower Confidence Bound

Lower confidence bound or LCB (UCB for maximization) is a function which is defined as

$$\begin{aligned} \mathrm{LCB}(A_i) &= \mu_i - \xi \sigma_i \\ A_t &= \underset{A \in \bar{\mathbb{A}}}{\operatorname{argmin}}(\mathrm{LCB}(A)) \end{aligned} \tag{4.7}$$

where the factor $\xi \geq 0$, controls exploration-exploitation trade-off. When $\xi = 0$, this LCB is same as greedy function. For higher values, areas with larger uncertainty are preferred.

4.4.4 Thompson

Thompson sampling [53] is a unique method where rather than directly using the predicted mean and variance, a new normal sample is created with the predicted values as parameters. This ensures that each sample has some generalization.

$$\begin{aligned} \mathrm{TS}(A_i) &= \mathcal{N}(\mu_i, \sigma_i) \\ A_t &= \underset{A \in \bar{\mathbb{A}}}{\operatorname{argmin}}(\mathrm{TS}(A)) \end{aligned} \tag{4.8}$$

4.5 Reproducibility

The AutoML NAS checklist [31] introduced by the AutoML organization helps in developing NAS methods that are reproducible and whose results can be validated. This thesis follows most of the steps mentioned in their checklist.

- Each algorithm was run for 50 times with different seeds and their mean results were compared. All seeds were saved for verification.
- All algorithms were tested on NAS-Bench-101 with the same parameters on the same hardware.
- Performance was compared in terms of error vs number of samples which can be roughly approximated into GPU hours.
- Only validation error was used during architecture search.
- The algorithm was compared with random search. This is because random search is currently considered as the baseline strategy for NAS-Bench-101, given that most of the architectures in the dataset are very good and there is almost an 80% chance of getting a good network when sampled randomly.

Chapter 5

Results and Observations

This chapter presents the results of the various experiments conducted with their interpretations. It also compares the Bayesian approach developed during this project with the existing state-of-the-art methods on the NAS-Bench-101 dataset. Since the workflow is divided into various segments as seen in Section 3.1, the results of each segment are presented as separate sections. In all cases, the results of the Bayesian optimization are plotted over the number of samples observed. Each process has been repeated 50 times and for every 10 samples seen, the mean and the standard deviation of the best samples found at that time are plotted. Note that 150 samples trained sequentially is roughly equivalent to 47 GPU hours (2 days) [57].

5.1 Graph Kernels

We evaluate the kernels qualitatively by their ability to group networks and qualitatively based on best error sampled and the number of samples taken to reach it.

5.1.1 Qualitative analysis

The main goal of the kernel is to identify similar networks that have similar accuracy. In order to test the effectiveness of the kernel qualitatively, a small set of known networks (1000) is sampled from the search space and is evaluated with the kernel. Based on the covariance values in the kernel matrix, the networks are separated into arbitrary groups. Since the accuracy of each network is known, it is plotted alongside the groups to indicate how well architectures with similar accuracy are grouped as shown in Figure 5.1. The grouping is similar to the k-nearest neighbour classification, where arbitrary centres/networks are selected, and the location of each architecture is dependent on its covariance value with respect to the chosen centres. Its prediction accuracy decides the colour of the network. The higher the accuracy, the lighter the colour.

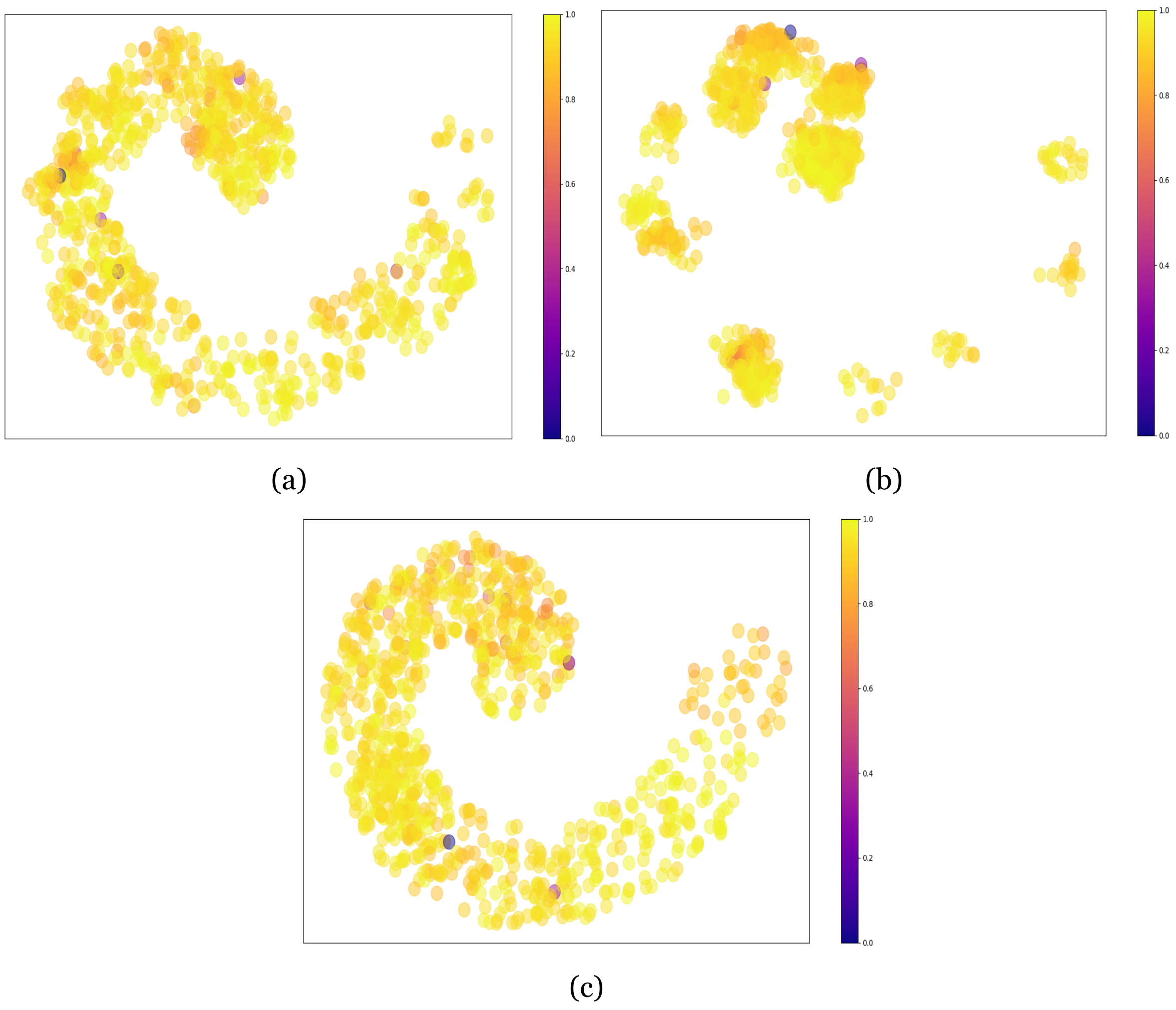


Figure 5.1: Spiral graph of the different kernels evaluated using a constant set of networks. The location of each network is dependent on its covariance value, w.r.t the chosen centres. The colours represent the validation accuracy of the networks. An ideal kernel would be able to segregate the poor performing networks from the good ones. (a) WL subtree kernel (b) Edge kernel (c) GCN kernel

From the qualitative analysis in Figure 5.1, we can observe that compared to WL subtree and GCN kernels, the edge kernel has denser groups. While some of these have accuracy in the same neighbourhood, it also incorrectly groups many poor networks with the good ones. The WL kernel performs better as the networks are spread apart and the number of incorrect grouping is less. The GCN kernel is very similar to the

WL, the key difference being that the worst networks are clearly visible, which means that they are not grouped along with the good ones.

5.1.2 Quantitative analysis

Figure 5.2 shows how each kernel affects the Bayesian optimization process. It is clear that GCN kernel is the best amongst the three as it reaches a much lower error (closer to the global minimum) and faster. The performance of the WL kernel though not as good as the GCN but, it is still an improvement over the random sampling approach 5.2 (d). The edge kernel is only slightly better than random sampling. It is not surprising given that it is a naive implementation that only counts edges and does not take the entire network structure into account.

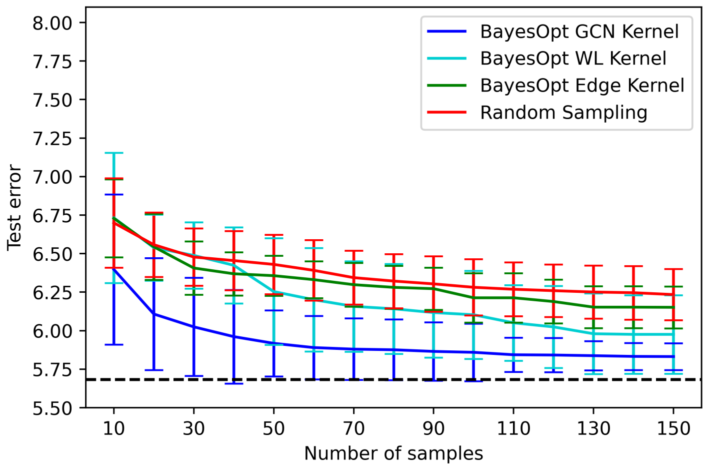


Figure 5.2: Evolution of the best architecture found. The dotted black line indicates the best possible error in the NAS-Bench-101 dataset.

5.2 Sampling function

To compare the best sampling function, we evaluate them under the same initialization. We use the GCN kernel with the LCB/UCB acquisition and sample 3000 points from the search space for every iteration. From the Figure 5.3, it is evident that the hybrid sampling function is the best among the sampling functions. This is because unlike the rest, it has a good trade-off between exploring and exploiting the search space and does not get stuck in a local minima.

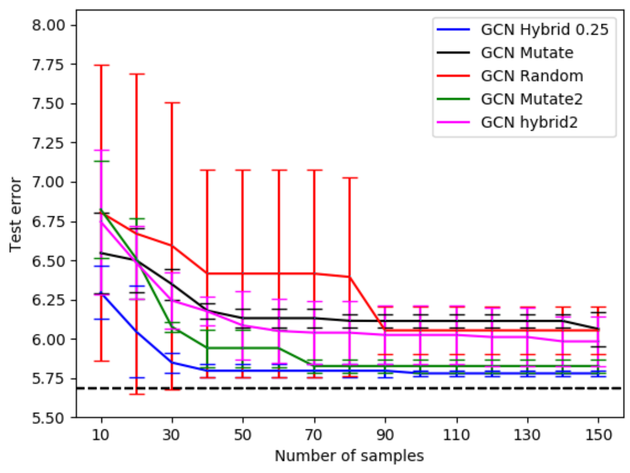


Figure 5.3: Comparison of the various sampling techniques under same conditions. Mutate2 and Hybrid2 refer to the sampling functions where the top n networks are mutated rather than just the top one.

5.3 Comparison with SOTA

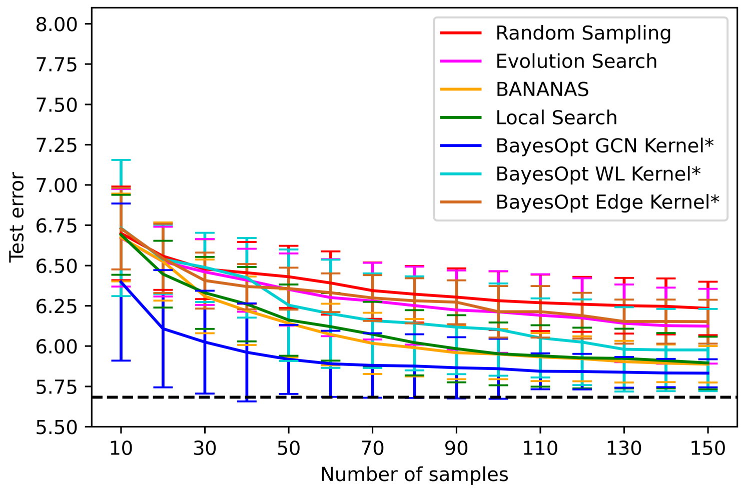


Figure 5.4: Comparison with different algorithms on the NAS-Bench-101 dataset.

Figure 5.4 shows the result of the comparison between the GCN kernel and WL kernel against the state-of-the-art methods such as BANANAS [57], local search [58] and also against regularized evolution [42] and random search. Both the GCN and WL kernels use the LCB acquisition function with hybrid sampling. Results of BANANAS and local search were obtained from their open-sourced code. From the graph, it can be observed that the GCN kernel is able to outperform the current SOTA methods by a large margin. It is able to reach the sample with a better error at around 70 samples while both local search and BANANAS are not able to reach this even after observing 150 samples. While the improvement is only around 4% in accuracy when compared to local search, the major advantage of the GCN kernel is its effectiveness in reaching a good network. This translates to saving a time of $\sim$ 20 GPU hours which is a huge reduction in resources. Table 5.1 augments the previous statements showing that the GCN kernel is consistently able to sample a very good network much faster than any other methods. This is a very surprising conclusion given that the GCN model was randomly initialized and was not trained.

	Avg accuracy(%)	**Avg number of samples**
Random sampling	93.75	86.4
Regularized evolution	93.87	91.36
Local search	94.112	84.6
BANANAS	94.104	95.26
BayesOpt Edge	93.84	91.9
BayesOpt WL	94.07	135.33
BayesOpt GCN	**94.168**	**54.4**

Table 5.1: Average accuracy of best samples at the end obtained by each model and average number of samples needed to reach it.

Chapter 6

Conclusion

In this thesis, we had discussed the potential of Bayesian optimization as an effective search strategy for Neural architecture search. We estimated the error of a set of networks as a multivariate Gaussian distribution parameterized by the network structure and dataset distribution through Gaussian process regression. For this purpose, in order to fit the GP, three graph kernels were implemented that encode the high dimensional graph data into a low dimensional vector that can be processed by conventional covariance functions. During the thesis, different acquisition functions and sampling functions were explored. The scope of the project was limited to finding the best model and hence the estimated model would not be a good predictor for neural network performance.

While there are plenty of search strategies, including a few based on BO, most approaches either require lots of samples to build a decent model or are computationally expensive. We developed a simple method which does not require expensive hardware and could still achieve better results on the NAS-Bench-101 benchmark dataset compared to other approaches. Especially, the model with the untrained GCN kernel was able to identify the best architectures at almost half the time taken by other methods.

6.1 Discussion

All three kernels implemented are able to perform better than random search, a method considered to be the benchmark to beat by many experts [31]. This is especially difficult in search spaces like NAS-Bench-101 that is highly unbalanced and does not have a uniform distribution with respect to error. Though the results might be impressive, it raises the question - "how does it scale to other datasets?". NAS-Bench-101 is one of many benchmark datasets. Recently other such benchmarks like NAS-Bench-201 [11] and NAS-Bench-301 [48] provide diverse networks where strategies that work in one search space may fail elsewhere [58]. So it is necessary to test on multiple search spaces and believe that most search spaces are covered. Also BayesOpt heavily depends on acquisition function to explore the search space. In the case of poor choice of acquisition function, the effectiveness of BayesOpt decreases drastically.

6.2 Future Work

The next step would be to test the model on different datasets and see if it is still able to perform as well. It would also be interesting to check whether search strategies are transferable across search spaces. Follow up suggestions to this thesis include incorporating hyberband [30] to improve performance estimation and reduce the time taken for training. Another idea would be to use a GCN based auto-encoder to encode the graph data. The feature vector from the latent space might be more informative than a simple feed-forward network.

References

[1] Abramowitz, M. and Stegun, I. A. (Eds.). “Gamma (Factorial) Function”. In: *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing*. 1972, pp. 253–294.

[2] Arfken, G. “Spherical Bessel Functions”. In: *Mathematical Methods for Physicists*. 3rd edtion. FL: Academic Press, 1985, pp. 622–636.

[3] Barret Zoph and Vijay Vasudevan and Jonathon Shlens and Quoc V. Le. “Learning Transferable Architectures for Scalable Image Recognition”. In: *CoRR* abs/1707.07012 (2017). arXiv: `1707.07012`. URL: `http://arxiv.org/abs/1707.07012`.

[4] Bergstra, J, Yamins, D, and Cox, D D. “Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. JMLR.org, 2013, pp. I–115–I–123.

[5] Bergstra, James S., Bardenet, Rémi, Bengio, Yoshua, and Kégl, Balázs. “Algorithms for Hyper-Parameter Optimization”. In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger. Curran Associates, Inc., 2011, pp. 2546–2554. URL: `http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`.

[6] Bergstra, James and Bengio, Yoshua. “Random Search for Hyper-Parameter Optimization”. In: *Journal of Machine Learning Research* 13.10 (2012), pp. 281–305. URL: `http://jmlr.org/papers/v13/bergstra12a.html`.

[7] Brochu, Eric, Cora, Vlad M, and Freitas, Nando de. “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning”. In: *arXiv e-prints* (Dec. 2010), arXiv:1012.2599. arXiv: `1012.2599 [cs.LG]`.

[8] Bruna, Joan, Zaremba, W., Szlam, Arthur, and LeCun, Y. “Spectral Networks and Locally Connected Networks on Graphs”. In: *CoRR* abs/1312.6203 (2014).

[9] Cai, Han, Chen, Tianyao, Zhang, Weinan, Yu, Yong, and Wang, Jun. “Efficient architecture search by network transformation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[10] Cox, Dennis D. and John, Susan. “SDO: A Statistical Method for Global Optimization”. In: *in Multidisciplinary Design Optimization: State-of-the-Art*. 1997, pp. 315–329.

[11] Dong, Xuanyi and Yang, Yi. “NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search”. In: *International Conference on Learning Representations*. 2020. URL: `https://openreview.net/forum?id=HJxyZkBKDr`.

[12] Elsken, Thomas, Hendrik Metzen, Jan, and Hutter, Frank. “Neural Architecture Search: A Survey”. In: *arXiv e-prints* (Aug. 2018), arXiv:1808.05377. arXiv: `1808.05377`.

[13] Falkner, Stefan, Klein, Aaron, and Hutter, Frank. “{BOHB:} Robust and Efficient Hyperparameter Optimization at Scale”. In: *CoRR* abs/1807.0 (2018). arXiv: `1807.01774`. URL: `http://arxiv.org/abs/1807.01774`.

[14] Gärtner, Thomas, Flach, Peter, and Wrobel, Stefan. “On Graph Kernels: Hardness Results and Efficient Alternatives”. In: *Learning Theory and Kernel Machines*. Ed. by Bernhard Schölkopf and Manfred K. Warmuth. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 129–143. ISBN: 978-3-540-45167-9.

[15] Grover, Aditya and Leskovec, Jure. “Node2vec: Scalable Feature Learning for Networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 855–864. ISBN: 9781450342322. DOI: `10.1145/2939672.2939754`. URL: `https://doi.org/10.1145/2939672.2939754`.

[16] He, Kaiming, Zhang, X., Ren, Shaoqing, and Sun, Jian. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[17] Hochreiter, Sepp and Schmidhuber, Jürgen. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[18] Huang, Gao, Liu, Zhuang, Maaten, Laurens van der, and Weinberger, Kilian Q. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV].

[19] Jones, Donald R. “A Taxonomy of Global Optimization Methods Based on Response Surfaces”. In: *Journal of Global Optimization* 21.4 (Dec. 2001), pp. 345–383. ISSN: 1573-2916. DOI: 10.1023/A:1012771025575. URL: https://doi.org/10.1023/A:1012771025575.

[20] Joyce, James. “Bayes’ Theorem”. In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Spring 2019. Metaphysics Research Lab, Stanford University, 2019.

[21] Jozefowicz, Rafal, Zaremba, Wojciech, and Sutskever, Ilya. “An Empirical Exploration of Recurrent Network Architectures”. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, 2015, pp. 2342–2350.

[22] Kandasamy, Kirthevasan and Neiswanger, Willie and Schneider, Jeff and Poczos, Barnabas and Xing, Eric. “Neural Architecture Search with Bayesian Optimisation and Optimal Transport”. In: *arXiv e-prints* (Feb. 2018), arXiv:1802.07191. eprint: 1802.07191.

[23] Kashima, H., Tsuda, K., and Inokuchi, A. “Marginalized Kernels between Labeled Graphs”. In: *20th International Conference on Machine Learning* (Aug. 2003), pp. 321–328.

[24] Kipf, Thomas N and Welling, Max. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *5th International Conference on Learning Representations, {ICLR} 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[25] Kriege, Nils M, Johansson, Fredrik D, and Morris, Christopher. “A survey on graph kernels”. In: *Applied Network Science* 5.1 (2020), pp. 1–42.

[26] Krizhevsky, A. *Learning Multiple Layers of Features from Tiny Images*. 2009.

[27] Kushner, H. J. “A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise”. In: *Journal of Basic Engineering* 86 (1964), pp. 97–106.

[28] L. Liberti and N. Maculan. “Global Optimization: From Theory to Implementation.” In: Springer Non-convex Optimization and Its Applications, 2006.

[29] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551.

[30] Li, Lisha and Jamieson, Kevin and DeSalvo, Giulia and Rostamizadeh, Afshin and Talwalkar, Ameet. “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization”. In: *arXiv e-prints* (Mar. 2016). eprint: `1603.06560`.

[31] Lindauer, Marius and Hutter, Frank. *Best Practices for Scientific Research on Neural Architecture Search*. 2020. arXiv: `1909.02453 [cs.LG]`.

[32] Liu, Hanxiao, Simonyan, Karen, and Yang, Yiming. “DARTS: Differentiable Architecture Search”. In: *ICLR 2019*. Apr. 2019. URL: `https://www.microsoft.com/en-us/research/publication/darts-differentiable-architecture-search/`.

[33] Matern, B. “Spatial Variation”. In: *Lecture Notes in Statistics*. Vol. 5. 1960.

[34] Mendoza, Hector, Klein, Aaron, Feurer, Matthias, Springenberg, Jost Tobias, and Hutter, Frank. “Towards Automatically-Tuned Neural Networks”. In: *Proceedings of the Workshop on Automatic Machine Learning*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Vol. 64. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, Feb. 2016, pp. 58–65. URL: `http://proceedings.mlr.press/v64/mendoza%7B%5C_%7Dtowards%7B%5C_%7D2016.html`.

[35] Mockus, J. “Application of Bayesian approach to numerical methods of global and stochastic optimization”. In: *Journal of Global Optimization* 4 (1994), pp. 347–365.

[36] Mockus, J. “The Bayesian approach to global optimization”. In: *System Modeling and Optimization*. Ed. by R. F. Drenick and F. Kozin. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, pp. 473–481. ISBN: 978-3-540-39459-4.

[37] Oord, Aäron van den, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew W., and Kavukcuoglu, Koray. “WaveNet: A Generative Model for Raw Audio”. In: *ArXiv* abs/1609.03499 (2016).

[38] Orlin, James, Punnen, Abraham, and Schulz, Andreas. “Local Search in Combinatorial Optimization”. In: vol. 33. Jan. 2004, pp. 587–596.

[39] Perozzi, Bryan, Al-Rfou, Rami, and Skiena, Steven. “DeepWalk”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’14* (2014). DOI: `10.1145/2623330.2623732`. URL: `http://dx.doi.org/10.1145/2623330.2623732`.

[40] Rasmussen, C. and Nickisch, H. “Gaussian Processes for Machine Learning (GPML) Toolbox”. In: *J. Mach. Learn. Res.* 11 (2010), pp. 3011–3015.

[41] Real, Esteban, Aggarwal, Alok, Huang, Yanping, and Le, Quoc V. *Regularized Evolution for Image Classifier Architecture Search*. 2019. arXiv: `1802.01548 [cs.NE]`.

[42] Real, Esteban, Aggarwal, Alok, Huang, Yanping, and Le, Quoc V. “Regularized Evolution for Image Classifier Architecture Search”. In: *CoRR* abs/1802.01548 (2018). arXiv: `1802.01548`. URL: `http://arxiv.org/abs/1802.01548`.

[43] Ru, Binxin, Wan, Xingchen, Dong, Xiaowen, and Osborne, Michael. *Neural Architecture Search using Bayesian Optimisation with Weisfeiler-Lehman Kernel*. 2020. arXiv: `2006.07556 [cs.LG]`.

[44] Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, Berg, Alexander C., and Fei-Fei, Li. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: `10.1007/s11263-015-0816-y`.

[45] Scherer, Dominik, Müller, Andreas, and Behnke, Sven. “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition”. In: *Artificial Neural Networks – ICANN 2010*. Ed. by Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101. ISBN: 978-3-642-15825-4.

[46] Shervashidze, Nino and Borgwardt, Karsten. “Fast subtree kernels on graphs”. In: *Advances in Neural Information Processing Systems 22*. Ed. by Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta. Curran Associates, Inc., 2009, pp. 1660–1668. URL: http://papers.nips.cc/paper/3813-fast-subtree-kernels-on-graphs.pdf.

[47] Shervashidze, Nino, Schweitzer, Pascal, Leeuwen, Erik Jan van, Mehlhorn, Kurt, and Borgwardt, Karsten M. “Weisfeiler-Lehman Graph Kernels”. In: *Journal of Machine Learning Research* 12.77 (2011), pp. 2539–2561. URL: http://jmlr.org/papers/v12/shervashidze11a.html.

[48] Siems, Julien, Zimmer, Lucas, Zela, Arber, Lukasik, Jovita, Keuper, Margret, and Hutter, Frank. *NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search*. 2020. arXiv: 2008.09777 [cs.LG].

[49] Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. *Practical Bayesian Optimization of Machine Learning Algorithms*. 2012. arXiv: 1206.2944 [stat.ML].

[50] Stanley, K. O., D’Ambrosio, D. B., and Gauci, J. “A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks”. In: *Artificial Life* 15.2 (2009), pp. 185–212. DOI: 10.1162/artl.2009.15.2.15202.

[51] Swersky, Kevin and Duvenaud, David and Snoek, Jasper and Hutter, Frank and Osborne, Michael A. “Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces”. In: *arXiv e-prints* (Sept. 2014), arXiv:1409.4011. arXiv: 1409.4011.

[52] Szegedy, Christian, Vanhoucke, Vincent, Ioffe, Sergey, Shlens, Jon, and Wojna, Zbigniew. “Rethinking the Inception Architecture for Computer Vision”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2818–2826.

[53] Thompson, William R. “On the Likelihood that one unknown probability exceeds another in view of the evidence of two samples”. In: *Biometrika* 25.3-4 (Dec. 1933), pp. 285–294. ISSN: 0006-3444. DOI: `10.1093/biomet/25.3-4.285`. eprint: `https://academic.oup.com/biomet/article-pdf/25/3-4/285/513725/25-3-4-285.pdf`. URL: `https://doi.org/10.1093/biomet/25.3-4.285`.

[54] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz, and Polosukhin, Illia. “Attention is All you Need”. In: *ArXiv* abs/1706.03762 (2017).

[55] Wang, Yuxuan, Skerry-Ryan, RJ, Stanton, Daisy, Wu, Yonghui, Weiss, Ron J., Jaitly, Navdeep, Yang, Zongheng, Xiao, Ying, Chen, Zhifeng, Bengio, Samy, Le, Quoc, Agiomyrgiannakis, Yannis, Clark, Rob, and Saurous, Rif A. “Tacotron: Towards End-to-End Speech Synthesis”. In: *arXiv e-prints*, arXiv:1703.10135 (Mar. 2017), arXiv:1703.10135. arXiv: `1703.10135 [cs.CL]`.

[56] Weisfeiler, B. and Lehman, A. A. “A reduction of a graph to a canonical form and an algebra arising during this reduction”. In: *Nauchno-Technicheskaya Informatsia* Ser. 2 (1968), p. 9. URL: `https://www.iti.zcu.cz/wl2018/pdf/wl_paper_translation.pdf`.

[57] White, Colin, Neiswanger, Willie, and Savani, Yash. “BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search”. In: *arXiv e-prints* (Oct. 2019), arXiv:1910.11858. arXiv: `1910.11858`.

[58] White, Colin, Nolen, Sam, and Savani, Yash. *Local Search is State of the Art for Neural Architecture Search Benchmarks*. 2020. arXiv: `2005.02960 [cs.LG]`.

[59] Xu, Keyulu, Hu, Weihua, Leskovec, Jure, and Jegelka, Stefanie. “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations*. 2019. URL: `https://openreview.net/forum?id=ryGs6iA5Km`.

[60] Ying, Chris, Klein, Aaron, Real, Esteban, Christiansen, Eric, Murphy, Kevin, and Hutter, Frank. “NAS-Bench-101: Towards Reproducible Neural Architecture Search”. In: *CoRR* abs/1902.09635 (2019). arXiv: `1902.09635`. URL: `http://arxiv.org/abs/1902.09635`.

[61] Zhigljavsky, A. and Zilinskas, A. “Stochastic Global Optimization.” In: Springer Optimization and Its Applications, 2008. URL: `https://link.springer.com/book/10.1007/978-3-642-27479-4`.

[62] Zoph, Barret and Le, Quoc V. “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578* (2016).