UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

# Neural architecture search: a probabilistic approach

*Author:*
Volodymyr LUT

*Supervisor:*
Yuriy KHOMA
Vasilii GANISHEV

*A thesis submitted in fulfillment of the requirements for the degree of Master of Science*

*in the*

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

# Declaration of Authorship

I, Volodymyr LUT, declare that this thesis titled, "Neural architecture search: a probabilistic approach" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

*"It's inspiring to see how AI is starting to bear fruit that people can actually taste. There is still a long way to go before we are truly an AI-first world, but the more we can work to democratize access to the technology—both in terms of the tools people can use and the way we apply it—the sooner everyone will benefit."*

Sundar Pichai, CEO Alphabet Inc., May 17, 2017

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

**Neural architecture search: a probabilistic approach**

by Volodymyr LUT

## *Abstract*

In this project, we introduce the Bayesian Optimization (BO) implementation of the NAS algorithm that is exploiting patterns found in most optimal unique architectures sampled from the most popular NAS dataset and benchmarking tool **NASbench-101** (Dong and Yang, 2020a). The proposed solution leverages a novel approach to path-encoding and is designed to perform reproducible search even on a relatively small initial batch obtained from the random search. This implementation does not require any special hardware, it is publicly available.

# *Acknowledgements*

First of all, I would like to thank my supervisor Mr. Yuriy Khoma and Mr. Vasilii Ganishev for all their support, patience and knowledge shared with me during this project.

I would also like to thank all my colleagues in UCU - those who already graduated, who were working on their masters during this year, and those who left UCU - for being a bright lighthouse, which navigated me towards excellence, for showing a good example and for being a strong and supportive community.

Many thanks to the UCU team - to all those people, who build and design the educational system I am proud being a part of, to all those who inspired me, shared their vision and made my time being a masters program student an unforgettable experience. I thank everyone who works every day in the UCU - no matter whether they are responsible for brilliant coffee or brilliant courses and workshops. Special cheer outs to Mr. Artem Chernodub for interesting homework and deep learning course at all. It helped me a lot during writing about ConvNets in this master's thesis.

Special thanks go to my friend and business partner Serhii Chepkyi for providing big support during my studies and also for providing a big help with visual materials for this project.

Last but definitely not least - I would like to thank my closest people and my family - without your eternal love, this would not be possible. This was the hardest years in my life and I'm glad and thankful that every second I've felt that I'm not alone.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **ML** | **M**achine **L**earning |
| **AutoML** | **Auto**mated **M**achine **L**earning |
| **NAS** | **N**eural **A**rchitecture **S**earch |
| **RL** | **R**einforcement **L**earning |
| **GP** | **G**aussian **P**rocess |
| **BO** | **B**ayesian **O**ptimization |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **ANN** | **A**rtificial **N**eural **N**etwork |
| **MLE** | **M**aximum **L**ikelyhood **E**stimate |

*For all the brave people who make it possible for millions of young Ukrainians to hold books in their hands instead of rifles and grenades.*

# Chapter 1

# Introduction

As machine learning provides a huge variety of automation possibilities for different industries the problem of automation of ML industry itself seems natural. For decades ML engineers were pioneers in the new era of computer science research. As a result, the new industry was shaped and this industry requires automation.

AutoML is a general name of automation in routine work of ML engineers including but not limited to data preparation, feature engineering, feature extraction, neural architecture search, hyperparameters selection, etc.

ML is reshaping businesses and other aspects of everyday life worldwide. We believe that everyone would benefit from the democratization of these new tools. Having the ability to run models on portable devices, IoT chips, and other mass-market hardware we treat AutoML as a big move towards in terms of a variety of different applications created.

Existing AutoML techniques require lots of computational resources and most of the research in the field is covered by tech giants nowadays.

However, since 2016 when one of the most popular projects in the field (Zoph and Le, 2016) was published a lot of new research occurred. The most popular techniques are BO, Reinforcement Learning (RL), different gradient descent-based approaches, evolutionary algorithms - more details could be found in the next chapter.

In 2019 Google Research Team introduced a dataset and benchmarking tool for NAS (Dong and Yang, 2020a). This dataset made a huge impact on the field

Good ANN architectures such as ResNet (He et al., 2015) and DenseNet (Huang et al., 2016) required a lot of domain knowledge and top-level expertise to be developed. ResNet revolutionized architecture design by introducing skip connections; This idea was elaborated even further in DenseNet. Those models obtained higher accuracy with better performance - that's why we are focusing on search of the most optimal connection schemes between layers.

We do this in a probabilistic manner by sampling a random batch from the NAS-bench dataset and trying to find repeating patterns in the models we treat as optimal. This allows us to make a simplification over search space and effectively explore unseen models that have a similar pattern.

While focusing on higher accuracy, we are also accounting training time for each model - in other words, we are trying to explore models as close to the Pareto frontier as possible.

# Chapter 2

# Background overview

## 2.1 History

In 2015, ResNet (He et al., 2015) become a winner of ILSVRC 2015 in image classification, detection, and localization and winner of MS COCO 2015 detection and segmentation. This enormous network contained 152 layers optimized by a lot of professional engineers manually. To provide better generalization developers introduced skip connections - extra connections between nodes in different layers of a neural network that skip one or more layers of nonlinear processing. Skip connections introduced a way to train very deep neural networks - ResNet-152 has 152 layers (to compare: (AlexNet, 2012 - 8 layers, GoogleNet, 2014 - 22 layers). That's why image classification contests are constantly showing a growing amount of layers for best-performing networks. Each competition is turning researchers more and more towards automation of this work - and this is a place where NAS becomes a new trend.

Barret Zoph and Quoc Le from Google Brain team (Zoph and Le, 2016) used a recurrent network to generate the model descriptions of neural networks and trained this RNN using RL agent to maximize the expected accuracy of the generated architectures on a validation set. This paper is one of the most cited in this field.

Even though Neural architectures may be complex, they could be described at some abstraction level using strings (or another encoding). In this project those values were generated by Recurrent Neural Network (RNN) - this gave authors a lot of flexibility since connections between nodes in RNN form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior which makes them quite applicable to the tasks where we need to generate correct sequences of output - in other words, tasks where the output of previous layer matters (see 2.1)

Those sequences are then validated and used to generate a neural architecture that will be evaluated in a distributed system. The reward received after training those architectures is later used to compute gradients and update weights in controller RNN - see 2.2.

That distributed system contained 800 GPU which were training 800 architectures at the same time concurrently - and whole training process took a lot of time - since good results could be obtained only after controller sampled 12,800 architectures.

So even though this work is the most cited, it is also the least reproduced - and this was a general weakness of a lot of modern NAS research. Then in 2019 two things in Google happened.

In 2019, Google researchers developed a family of models, called EfficientNets, which surpass state-of-the-art accuracy with up to 10x better efficiency (smaller and faster) using AutoML - see Tan and Le, 2019.

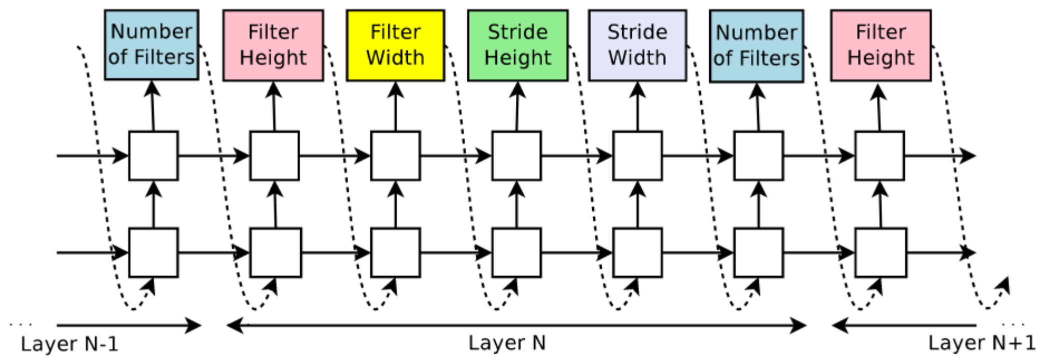

FIGURE 2.1: Controller RNN from Zoph and Le, 2016

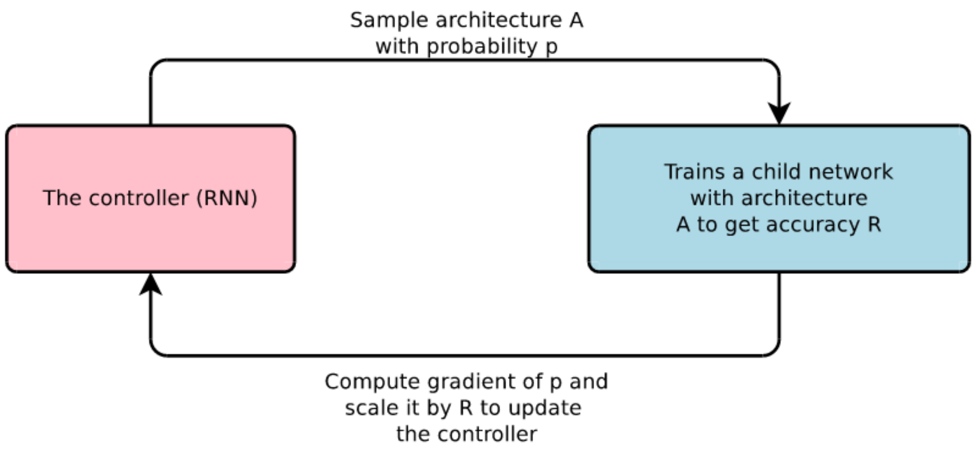
Sample architecture A with probability p
The controller (RNN)
Trains a child network with architecture A to get accuracy R
Compute gradient of p and scale it by R to update the controller

FIGURE 2.2: Training process in Zoph and Le, 2016

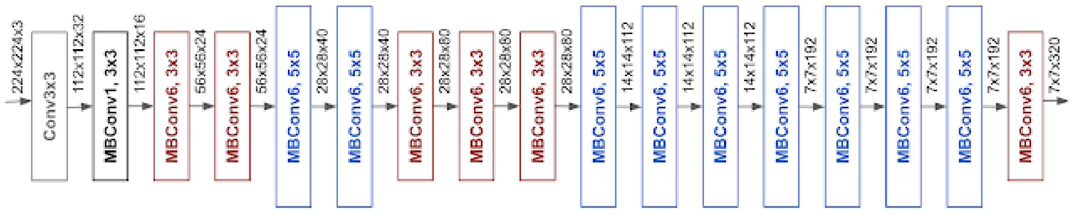

FIGURE 2.3: The architecture for EfficientNet's baseline network EfficientNet-B0 from Tan and Le, 2019

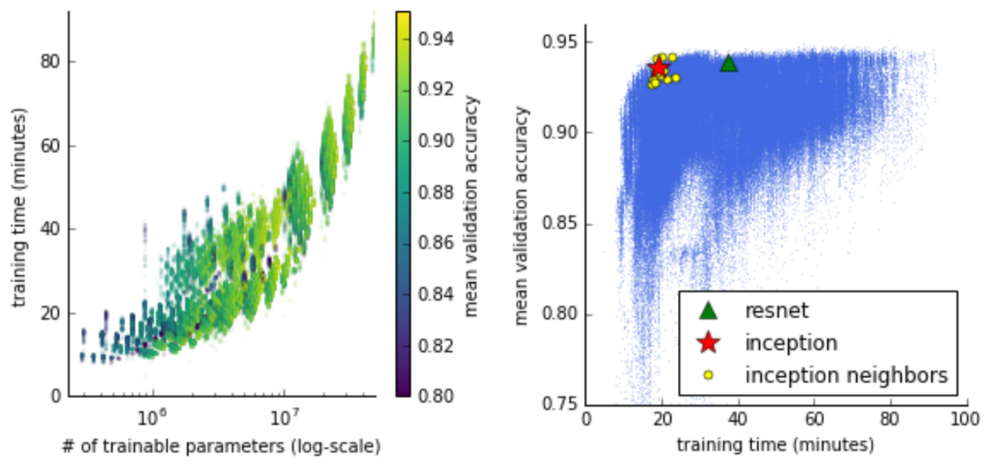

FIGURE 2.4: Training time vs mean accuracy from Dong and Yang, 2020a

## 2.2 NASbench-101

Also in 2019, another team of Google researchers released NASbench-101 Dong and Yang, 2020a. This is a tabular dataset containing of evaluation results of CNNs builded in ResNet-like (He et al., 2015) and Inception-like (Szegedy et al., 2015) manner.

Google Brain used their resources to create a NAS benchmark by evaluating a small cell search space. Those cells are easily scaled to bigger ones because of their ResNet-like design. Architectures used in this dataset were trained on CIFAR-10 because it is computationally cheap. Best performing models could then be scaled similarly as in Zoph and Le, 2016 to be evaluated on bigger datasets such as ImageNet or COCO.

This means that good neural architectures, at least for image classification tasks could be obtained in seconds instead of being evaluated on GPU.

The dataset contains approximately 423k different architectures, with 3 repetitions each. Most of the architectures score more than 90% validation accuracy on CIFAR-10 (see 2.4).

Search space is defined by a set of all valid directed acyclic graphs with 9 edges and all combinations of 3 operations (3X3 Max Pooling, 3X3 Convolution, 1X1 Convolution) in 5 available slots.

They've also published a great paper (Dong and Yang, 2020a) with a lot of explanatory statistical analysis over the dataset and details about benchmarking of SOTA algorithms.

## 2.3 SOTA

A variety of methods have been proposed to perform NAS, including reinforcement learning, Bayesian optimization with a Gaussian process model, sequential model-based optimization (SMAC), evolutionary search, and gradient descent over the past few years. This is a short overview of the most popular approaches to solve the NAS problem.

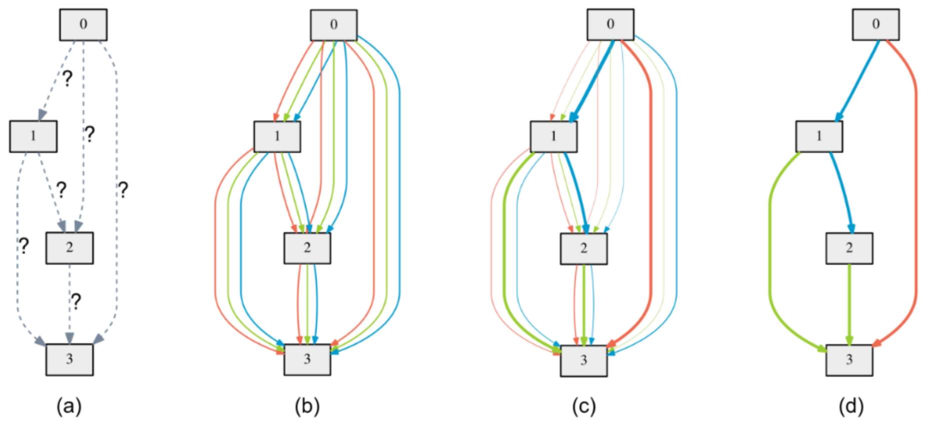

FIGURE 2.5: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights. (d) Inducing the final architecture from the learned mixing probabilities. Original image from Liu, Simonyan, and Yang, 2018

## 2.4 One-shot models

One-shot neural architecture search has played an important role in making NAS methods computationally feasible in practice.

One of the most successful one-shot model implementations is ENAS (Pham et al., 2018). This paper is an evolution of the Reinforcement Learning algorithm introduced by Zoph and Le, 2016. It introduced parameter sharing between models which made the algorithm 1000x less expensive than standard Neural Architecture Search. This unexpected effect of parameter sharing made the technique really popular in the field.

There is also a brilliant evolutional algorithm Real et al., 2017 that is able to discover models for the CIFAR-10 and CIFAR-100 datasets with accuracies of 94.6% (95.6% for ensemble) and 77.0%, respectively. But those solutions are usually hard to reproduce without strong domain knowledge or they are extremely expensive. Since benchmarking of different one-shot models using NASBench-101 could be tricky because weight-sharing algorithms have many factors that control their dynamics, two concurrent solutions were proposed on top of original NASBench to provide a better profiling and more diagnostic information - NASBench-201 (Dong and Yang, 2020b) and NASBench1-shot-1 (Zela, Siems, and Hutter, 2020).

One of the best performing solutions in NAS is not actually searching in some search space. In DARTS Liu, Simonyan, and Yang, 2018 architecture is fixed and algorithm is performing continuous relaxation of the search space to find the most optimal operations at the edges (see 2.5)

## 2.5 Probabilistic Methods

Project PARSEC (Casale, Gordon, and Fusi, 2019) is providing a probabilistic modeling framework for sampling-based optimization methods to learn a probability distribution over high-performing architectures for a specified supervised task.

In BANANAS (White, Neiswanger, and Savani, 2019), an ensemble of neural networks is trained to predict the mean and variance of validation error for candidate neural architectures in order to obtain a good acquisition function.

This method is not new and a lot of other researchers are trying to build cheap surrogate models in order to balance the high computational cost of obtaining real samples.

We are nothing different - by clustering most promising network architectures, we are trying to reduce search space to sample less real data from NASBench during optimization iteration.

# Chapter 3

# Proposed approach

Our primary goal is searching the best accuracy vs training time tradeoff in the search space available in NASbench-101 (**102**.

We are applying a search space limitation by setting available operations to *CONV3X3*. There are three available operations in search space and approximately 423$k$ unique models overall. Using one operation would reduce this search space. This is also done to ensure uniqueness of the architectures sampled - see 3.1 where different adjacency matrices encode same computation.

To query the batch data from NASbench, we randomly construct $7 \times 7$ upper-triangular matrices and use build-in NASbench methods to validate if constructed matrix is a 7-vertex directed acyclic graph.

Even though we not necessarily need exactly maximum amount of vertices to obtain a good accuracy, we want to obtain as big adjacency matrix as possible, because later we would calculate a probability of having edge in the exact index of adjacency matrix - to have a good probability measurement it is better to have all graphs of same vertex size.

Obtained adjacency matrices are later hashed to ensure that we do not query same architecture again - we want to spend budget only on unique ones.

This limitations have impact on the batch however - we can clearly state that some architectures are completely out of our search space (see 3.2)

Before we proceed, some things about Figure 3.2 should be explained:

- Points on the left chart is individual run values whereas points at the right chart (full data) - means of individual values.
- Chart on the left was collected for same vertex number whereas chart on the right (full data) contains vertex number from 3 to 7 (those are the boundaries because we have maximum 7 vertices and also we cannot have less than 3 vertices because this number includes also required input and output layers).
- Omitting pattern would be same non-depending on seed value - those are architectures we are unable to query.

Also it's important to notice that almost all samples scores more than 0.9 (see 3.3) of test accuracy. Since the maximum mean test accuracy value in NASBench is 94.32% points would be relatively close to each other we will need to carefully differentiate efficient and unefficient architectures.

## 3.1 Unsupervised clustering

General idea is that good architectures should share some edges in graph in common.

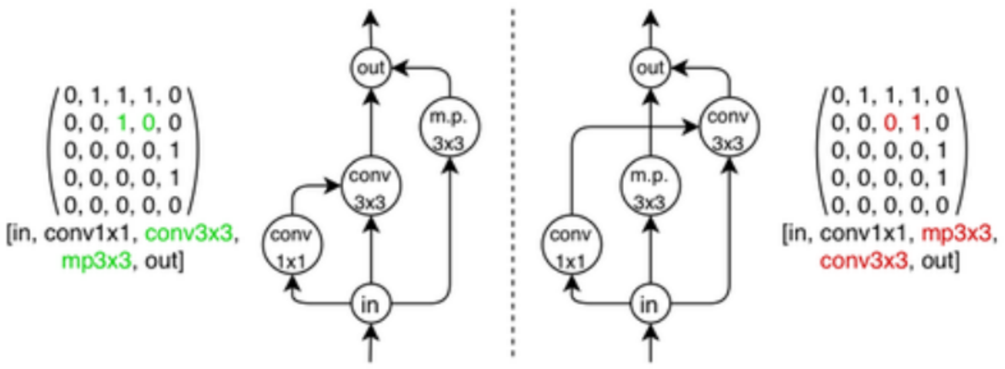

FIGURE 3.1: Different adjacency matrices encode same computation - example from Dong and Yang, 2020a

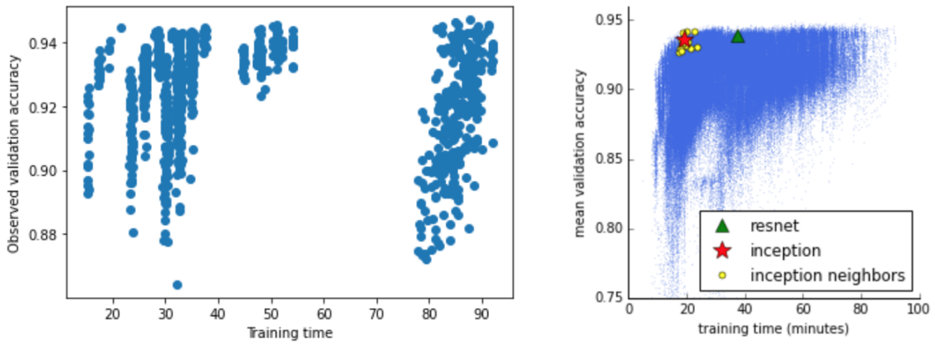

FIGURE 3.2: Sampled batch (left) compared to full dataset (right) Dong and Yang, 2020a

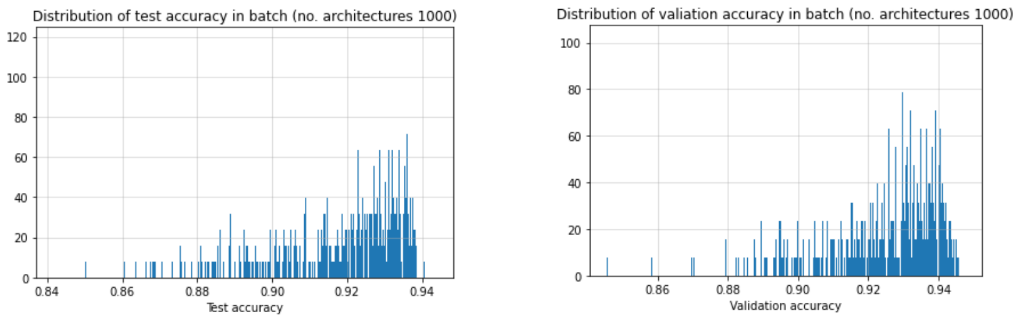

FIGURE 3.3: Test accuracy (left) and validation accuracy (right) distributions

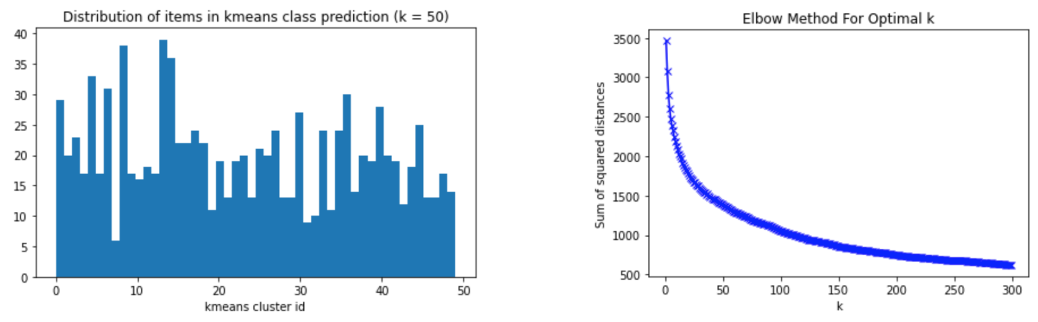

FIGURE 3.4: Number of elements in cluster(left); Similarity between elements in cluster vs k number (right)

Since we do not have any labels, we would need to group architectures from sampled batch using some unsupervised clustering algorithm.

This is done via KMeans; By default number of K is set to 50 to have bigger variance in each group - however, bigger value of K is also allowed (as long as it makes sense - see Figure 3.4)

After KMeans evaluation we would have 50 clusters in the dataset.

To filter out the most promising ones we use following algorithm:

1. Calculate percentage of clusters test accuracy means that would be lower than test accuracy mean of given cluster
2. Calculate percentage of clusters training time means that would be lower than training time mean of given cluster
3. Floor percentages to precision of 1 digit after comma to obtain rank $RANK \in (1, 2 \cdots 9, 10)$
4. Filter out models with training time $RANK < 3$
5. Sort data descending by test accuracy rank and ascending by training time rank so we wouldn't have groups with big training time means Subset desired amount of perspective groups from the top of the dataframe

This algorithm leaves out the most promising groups which will be used for further optimization (see 3.5)

## 3.2 Bayesian Optimization

Having adjacency matrices ready for groups, we can calculate the probability of having 1 at each element of the matrix

$$P = \frac{1}{n} \sum_{i=1}^{n} Adj_i$$

Those probabilities form most common patterns for each group (see 3.6)

Unlike BANANAS (White, Neiswanger, and Savani, 2019) where mutations are taken randomly from the perspective cell and ensemble of meta neural networks is used to provide a prior distribution, we once again reduce the search space.

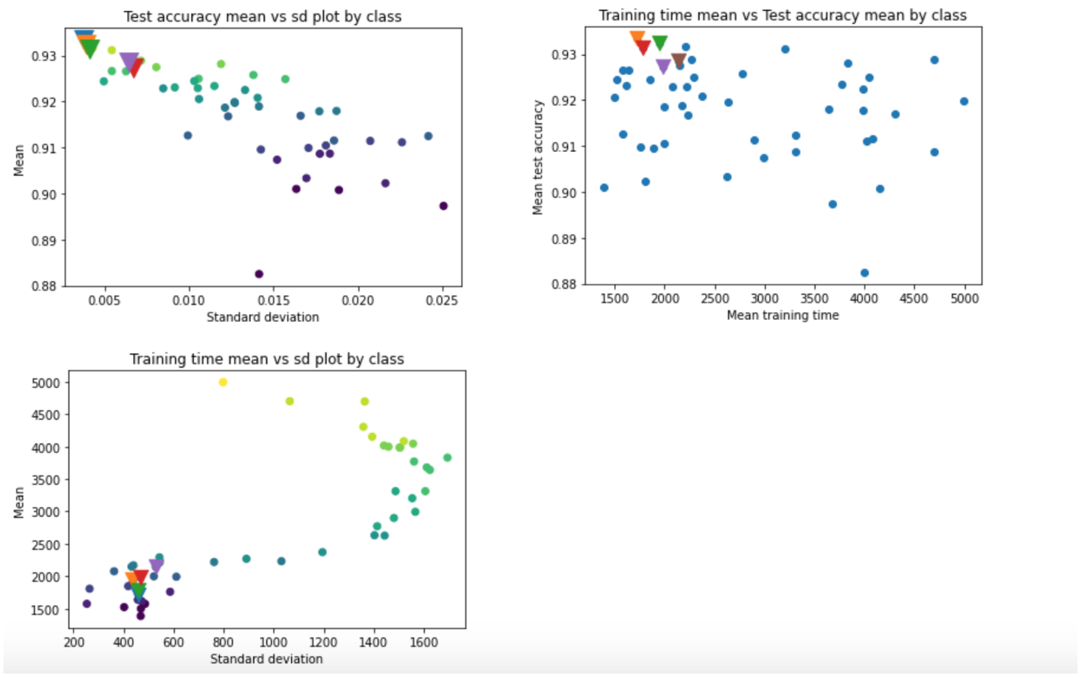

FIGURE 3.5: Most promising clusters

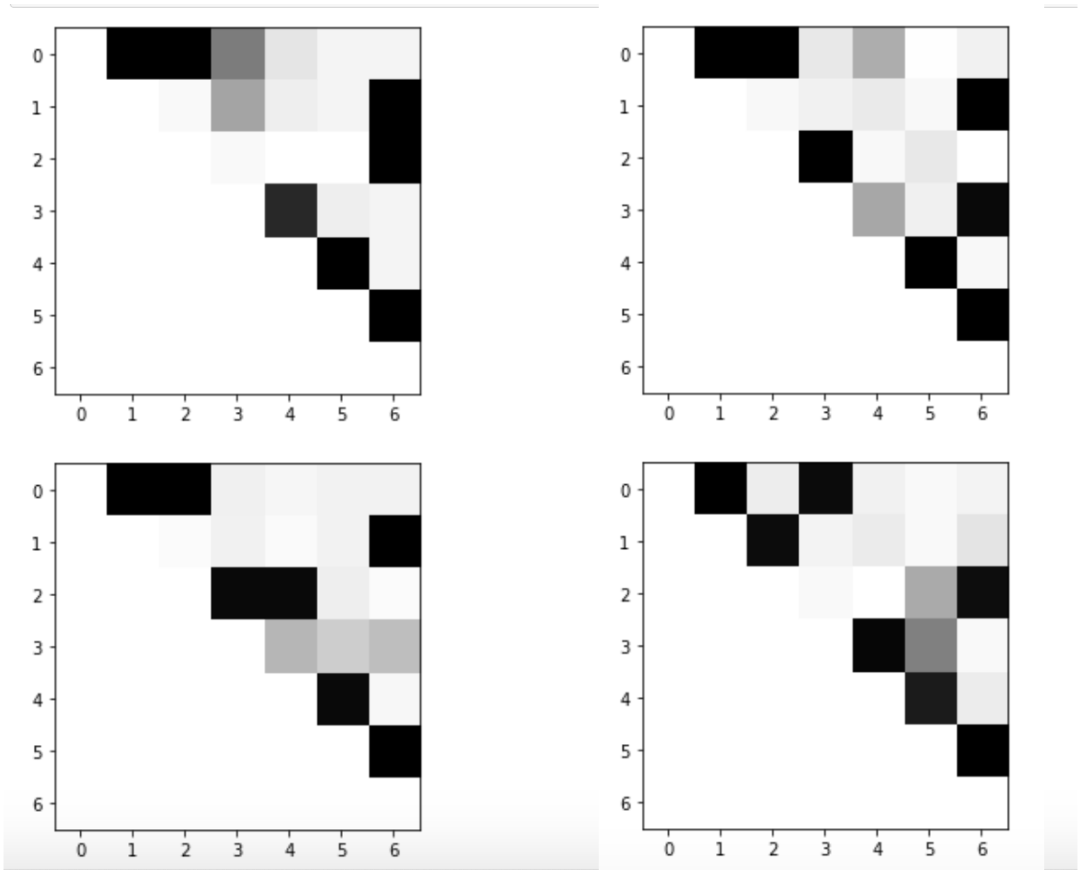

FIGURE 3.6: Most promising architecture probability heatmap

We are exploiting the knowledge from probabilities matrix given for current class. We do not want to loose strongest patterns - therefore we are converting all probabilities bigger than 0.9 (this is a hyperparameter) to 1 and others to zeros.

Parameter $n$ is subtraction of amount of non-zero elements in transformed probability matrix from number of edges left (In NASbench this is equal to 9). Depending on random cell and quality of clustering we are left with a number of 2 or 3 edges we will need to search for.

Parameters to optimize are row and column indices of matrix. We are optimizing a function $f(x_1, y_1, \cdots, x_n, y_n)$ which returns NASbench evaluation one sample of test accuracy for given architecture.

Since

- $f$ is a black box for which no closed form is known
- In real world it is expensive to evaluate $f$ and we are minimizing NASbench budget as much as possible
- Evaluations of $f$ are noizy because querying NASbench API returns one of 3 evaluations of each architecture

BO seems to be natural algorithm choice for this task.

Proposed BO implementation is based on Gaussian Process (GP) Regression. We use Lower Confidence Bound (LCB) as acquisition function and as exploration vs exploitation control.

$$LCB(x) = \mu_{GP}(x) + \kappa$$

After all clusters were optimized, we pick best fit value by discounting test accuracy over training time.

# Chapter 4

# Experiments

## 4.1 Batch generation

During experiments, we were querying 500 - 1000 unique elements for the initial batch.

We used only NASBench data for 108 epochs.

This results in an initial budget of at most 370 GPU hours.

Only 7-vertex directed acyclic graphs were queried. Initial budgeting could be slightly decreased (see 4.1) if we would replace CONV3X3 operation with CONV1X1, however, this was not performed because architectures with only CONV1X1 are showing ±10% lower validation and test accuracy. Because we are not optimizing operations currently, this was left for future experiments.

All CNNs from NASbench are trained on the CIFAR-10 dataset. CIFAR datasets are described in very deep detail in Chapter 3 of Krizhevsky, 2012, especially details about its collection. CIFAR is a set of $32 \times 32$ color images depicting real-world objects.

Classes in CIFAR are exclusive and do not assume instances overlapping - see 4.2.

## 4.2 Environment and results

All experiments were running on the CPU. The average run took ±10 minutes.

To reproduce results, make sure to install Tensorflow [Abadi et al., 2015] and load NASBench data for 108 epochs.

To ensure that the results are not depending on the batch, several experiments were conducted with different seed values.

An algorithm was evaluated with two sets of hyperparameters.

| OPTK | BREED LEN | SIMILARITY TOLERANCE | BATCH SIZE |
|---|---|---|---|
| k in KMeans | Num of perspective clusters | Min probability to preserve | |
| 50 | 5 | 0.9 | 1000 |
| 50 | 5 | 0.9 | 500 |

TABLE 4.1: Hyperparameters

In fact with $seed = 50$ on smaller batch size algorithm calculated optimized architecture with accuracy **0.9358** and number of trainable parameter **7857930** which is comparable to values received on bigger batch size.

It lost accuracy because variance inside clusters increased, but reducing initial random search may be a good tradeoff on other NAS problems.

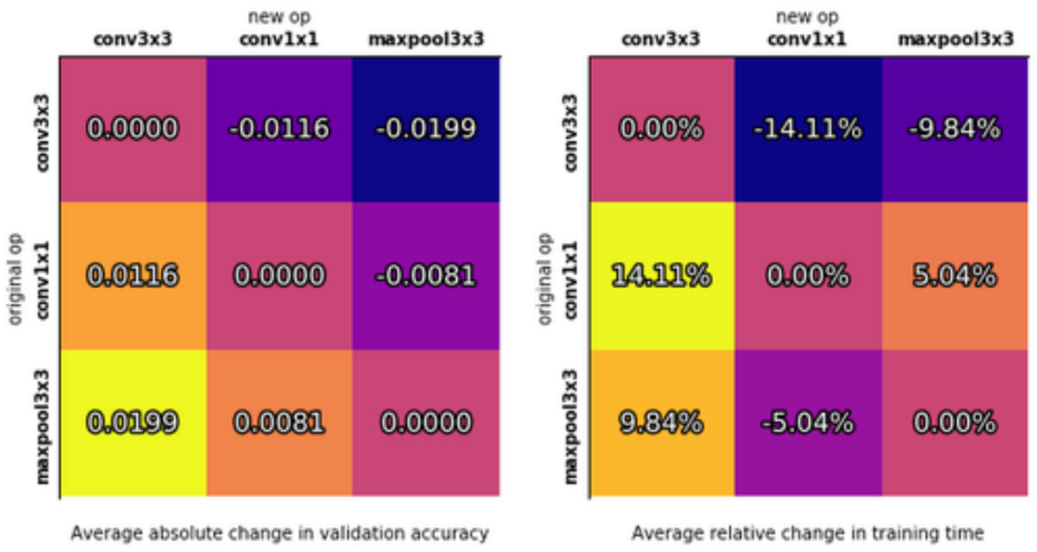

FIGURE 4.1: Average abs. change in accuracy and average relative change in training time after change of one operation Dong and Yang, 2020a

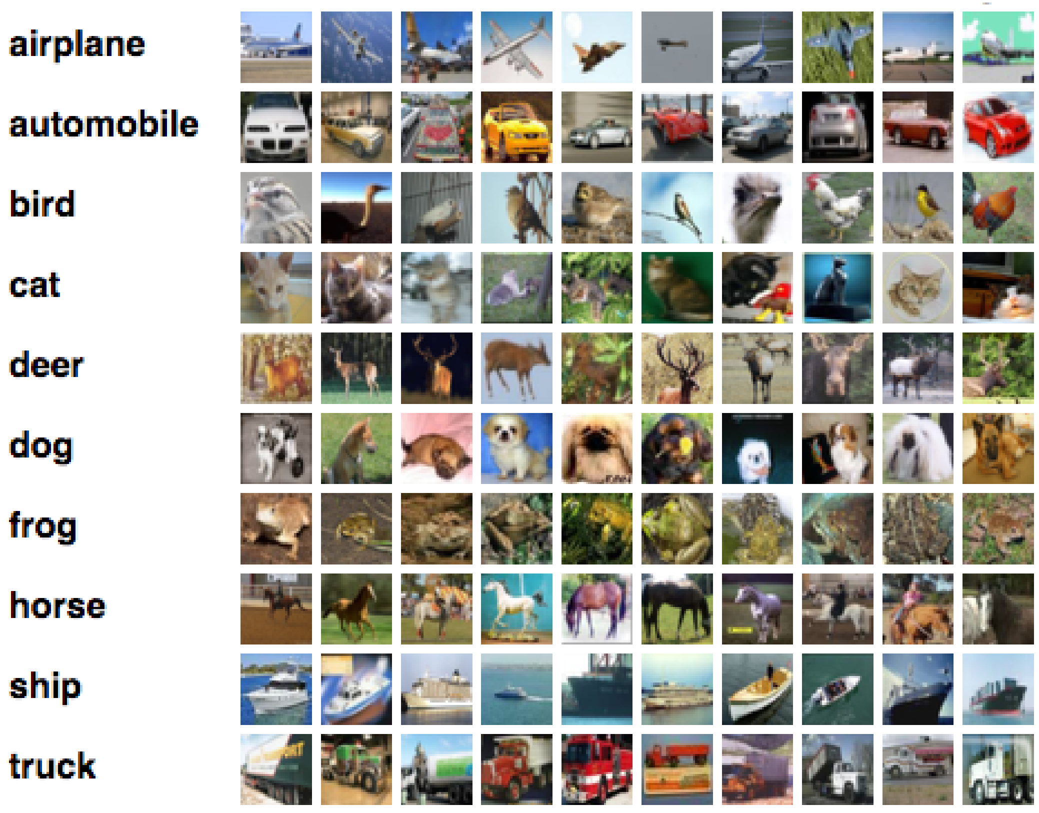

FIGURE 4.2: Classes of CIFAR10 including 10 random images from each Krizhevsky, 2012

## 4.3 Results

In all experiments BO was successfully converging fast (see 4.5). Optimized architectures are more cost-efficient than their neighbors within class (see 4.3)

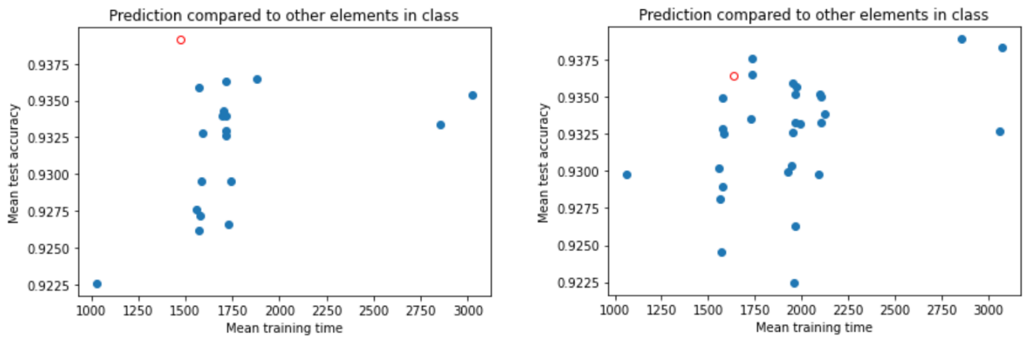

FIGURE 4.3: Optimized architecture vs elements of it's parent class

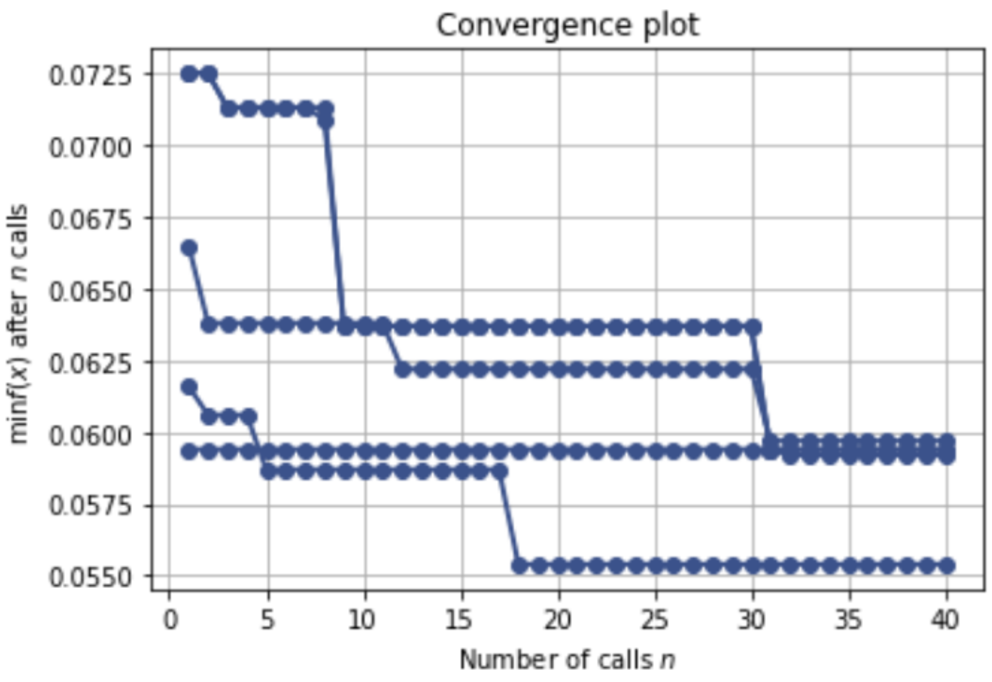

FIGURE 4.4: Convergence of BO for each class

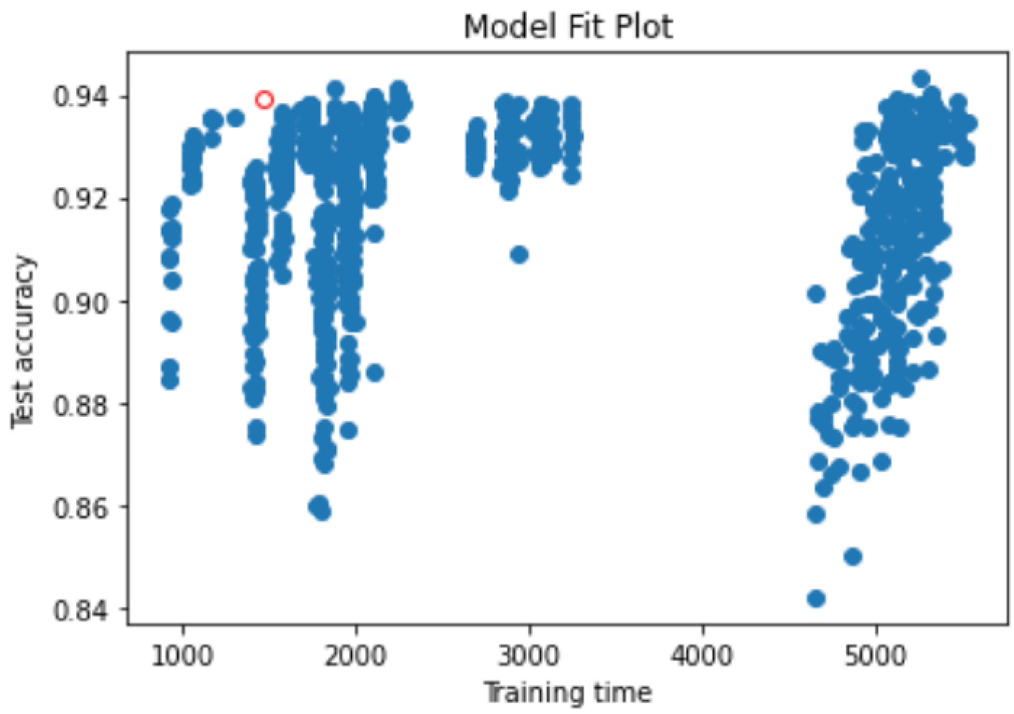

FIGURE 4.5: Predicted value plotted together with initial batch

## 4.4 Comparison with other NAS algorithms

| Algorithm | Search Time | Test Accuracy |
|---|---|---|
| RANDOM SEARCH (Nasbench-202) | 0.01 | 93.70 ± 0.36 |
| REINFORCE (Nasbench-202) | 0.012 | 93.85 ± 0.37 |
| BOHB (Nasbench-202) | 3.59 | 93.61 ± 0.53 |
| Proposed solution | - | 93.91 |

TABLE 4.2: Comparison with other algorithms

Unfortunately, most projects use NAS more like a dataset to train their surrogate models than for actual benchmarking, therefore, most of the results reported are out of NASBench search space.

## 4.5 Reproducibility of results

The proposed algorithm often finds optimal results non-depending of seed value. We can state that it is pretty ignorant to the random sample structure. This is because we are using a probability map of all architectures in the class, so the signal of the pattern is strong.

| Seed | Mean Test Accuracy | Trainable Parameters |
|---|---|---|
| 20 | 0.9318 | 8936714 |
| 30 | 0.9391 | 8294794 |
| 42 | 0.9391 | 8294794 |
| 50 | 0.9391 | 8294794 |

TABLE 4.3: Reproducibility information

# Chapter 5

# Conclusions

In this work, we were using probabilistic approaches to build a pipeline for searching neural architectures in artificially limited state space.

As a result of the research, we've come with a solution that exploits similarities found in clusters of neural architectures to effectively search best architectures based on prior knowledge using Bayesian Optimization (BO) on the Gaussian Process (GP) regression. The proposed solution was validated on a set of different seed values to showcase the reproducibility of the algorithm.

The solution is available on the GitHub [Lut, 2020].

We hope that this example will show that AutoML research is already a thing and could be held by other students and researches even on an occasion of low resources available. We hope to see more and more research in the filed in the upcoming years. As the community is building up this ecosystem we have no doubt NAS would become more reproducible.

# Bibliography

Abadi, Martín et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: `http://tensorflow.org/`.

Casale, Francesco Paolo, Jonathan Gordon, and Nicolo Fusi (2019). *Probabilistic Neural Architecture Search*. arXiv: `1902.05116 [stat.ML]`.

Dong, Xuanyi and Yi Yang (2020a). "NAS-Bench-102: Extending the Scope of Reproducible Neural Architecture Search". In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=HJxyZkBKDr`.

— (2020b). *NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search*. arXiv: `2001.00326 [cs.CV]`.

He, Kaiming et al. (2015). *Deep Residual Learning for Image Recognition*. arXiv: `1512.03385 [cs.CV]`.

Huang, Gao et al. (2016). *Densely Connected Convolutional Networks*. arXiv: `1608.06993 [cs.CV]`.

Krizhevsky, Alex (May 2012). "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto*.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2018). *DARTS: Differentiable Architecture Search*. arXiv: `1806.09055 [cs.LG]`.

Lut, Volodymyr (2020). *Neural Architecture Search: A Probabilistic Approach*. `https://github.com/volodymyrlut/masters-project`.

Pham, Hieu et al. (2018). *Efficient Neural Architecture Search via Parameter Sharing*. arXiv: `1802.03268 [cs.LG]`.

Real, Esteban et al. (2017). *Large-Scale Evolution of Image Classifiers*. arXiv: `1703.01041 [cs.NE]`.

Szegedy, Christian et al. (2015). *Rethinking the Inception Architecture for Computer Vision*. arXiv: `1512.00567 [cs.CV]`.

Tan, Mingxing and Quoc V. Le (2019). "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *arXiv e-prints*, arXiv:1905.11946, arXiv:1905.11946. arXiv: `1905.11946 [cs.LG]`.

White, Colin, Willie Neiswanger, and Yash Savani (2019). *BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search*. arXiv: `1910.11858 [cs.LG]`.

Zela, Arber, Julien Siems, and Frank Hutter (2020). *NAS-Bench-1Shot1: Benchmarking and Dissecting One-shot Neural Architecture Search*. arXiv: `2001.10422 [cs.LG]`.

Zoph, Barret and Quoc V. Le (2016). "Neural Architecture Search with Reinforcement Learning". In: *CoRR* abs/1611.01578. arXiv: `1611.01578`. URL: `http://arxiv.org/abs/1611.01578`.