Predicting Image Classifier Performance Using the Synthetic Petri Dish Method

Amber Cassimon$^{1[0000-0002-7471-2508]}$, Liam Hertoghs^{2}, Simon Vanneste$^{1[0000-0002-9664-9925]}$, Phil Reiter^{1}, Kevin Mets$^{3[0000-0002-4812-4841]}$, Tom De Schepper$^{3[0000-0002-2969-3133]}$, Siegfried Mercelis$^{1[0000-0001-9355-6566]}$, and Peter Hellinckx$^{2[0000-0001-8029-4720]}$

University of Antwerp - imec$^{1,\ 3}$, University of Antwerp^{2}
IDLab - Faculty of Applied Engineering^{1}, IDLab - Department of Computer Science^{3}, Faculty of Applied Engineering^{2}
Sint-Pietersvliet 7, 2000 Antwerp, Belgium^{1,3}, Groenenborgerlaan 171, 2020 Antwerp, Belgium^{2}
`amber.cassimon@uantwerpen.be`

Abstract. An important part of any Neural Architecture Search (NAS) system is a good performance estimation strategy. This strategy should ideally have a high accuracy, a low initialization time and a low query time. This paper evaluates the Synthetic Petri Dish method of neural network performance estimation in an image classification context. The method is tested on the CIFAR-10 dataset using the data from the NAS-Bench-101 dataset as ground truth. We examine different ways of constructing motifs and analyze the influence of these motif choices on the final performance. While the motif networks are capable of converging on the synthetic dataset, the outer loop losses show little improvement throughout the training process.

Keywords: Neural Architecture Search · Performance Estimation · AutoML

1 Introduction

In recent years Deep Learning (DL) has proven its ability to effectively solve a wide variety of problems[14][18]. Much of this improvement has been fostered by the availability of increasing amounts of computational power and data, but also the way in which these DL systems are designed. Deep Learning algorithms have the ability to automatically learn and distinguish patterns relevant for the problem at hand. Although these algorithms can learn to recognize patterns automatically, obtaining satisfactory results often still requires a significant engineering effort (i.e., when designing neural network architectures). This presents a significant obstacle to the use of DL in various fields.

1.1 Neural Architecture Search

In order to overcome this obstacle, the design process of DL systems can be partially automated using Neural Architecture Search (NAS) techniques. NAS systems will automatically try to find the best Neural Network (NN) architecture with minimal human involvement using various techniques such as Evolutionary Algorithms (EA) [5] and Reinforcement Learning (RL) [18].

Following the work of Elsken et al. [6], a NAS system can be divided into three main components: the search space, the search strategy and the performance estimation. The search space is used to define which NNs can be evaluated by the NAS system. A poor choice of search space can lead to suboptimal results from the NAS algorithm, for example, by excluding certain operations that could result in significantly better performance. The performance estimation strategy is used to determine the performance of a given NN on a given task, without having to train the NN to convergence. Finally, the search strategy will determine how the search space is explored.

This paper will focus on performance estimation using the synthetic petri dish method originally described by Rawal et al. [13]. We will evaluate this methodology's performance in the image classification domain.

We attempt to answer the following research questions in this paper:

1. Is the use of the Synthetic Petri Dish method in this domain feasible?
2. What is the influence of the choice of motif on the generated synthetic dataset?

This paper is organized as follows: We start by discussing related work in the field of NAS performance estimation (Section 2), followed by a detailed description of our modifications to the synthetic petri dish method (Section 3). Next, we explain the experimental set-up we used (Section 4) and we analyze the results from our experiments (Section 5). Finally, we draw some conclusions based on the results of our experiments (Section 6), and outline possible future research tracks (Section 7).

2 Related Work

NAS systems have shown that they can find architectures that outperform man-made architectures in various problem domains [5], [18]. These NAS systems commonly use EA [5] or RL [18], to explore their architecture search space. In early work [18], training a NN to convergence was used as a performance estimator. This has the advantage of being a near-perfect performance estimator (although variations in performance can still occur from hyperparameter tuning and different random initializations), but the required computational resources present a severe disadvantage. Because of this, a range of more efficient methods to predict task performance of NN architectures have been explored.

2.1 Low Fidelity Training

One performance estimation strategy is to lower the fidelity of training by: lowering the amount of training epochs [3] or the amount of training time [14], [1]. This was demonstrated by Domhan et al. [3] in their paper, where they sped up the hyperparameter tuning process for NNs by terminating the training of NNs early when it became clear that they weren't going to improve on the best configuration seen so far.

2.2 Learning Curve Extrapolation and Scheduling

Furthermore, we can increase the accuracy and efficiency of the performance estimation by extrapolating the performance during training [3], and stopping the training process on architectures whose extrapolation predicts poor final performance [9]. The main challenge with this method is performing accurate extrapolations with limited data from a limited number of training epochs [7]. Domhan et al. demonstrated this by using a linear combination of curves to extrapolate the learning curve of a NN, allowing them to determine which experiments were worthwhile to continue, and which experiments could be stopped early.

2.3 Weight Sharing

Another interesting method of performance estimation is by using weight sharing [12], sometimes also referred to as supernetwork training [10]. This involves initializing a supernetwork, which is a superposition of all networks in the search space. This supernetwork is then trained, by sampling a subnetwork, and using gradient descent to train this subnetwork. This procedure of sampling a subnetwork and training is repeated multiple times, until the entire supernetwork has been trained sufficiently. During performance estimation, a subnetwork is sampled, and can be evaluated on the target task with minimal or no additional training. While weight sharing is an efficient method, it is not without issues, with continuous relaxation systems suffering from architecture collapse [16], and low ranking correlation being observed between the ground truth and weight sharing settings [15].

2.4 Synthetic Petri Dish

In 2020, Rawal et al. [13] introduced their synthetic petri dish method. This method involves the generation of synthetic data in such a way that smaller versions of a NN have a similar performance ranking compared to their full size counterparts. Since this method is the focus of this paper, we will elaborate on it further.

In the original paper, Rawal et al. considered two problems, MNIST [2] and Penn Treebank (PTB) [11]. In the MNIST setting, they used a fully connected 2-layer NN, with 100 neurons in each layer. They generated a smaller version of this network, called a motif, by reducing the number of neurons in the fully connected layers down to 10. A similar methodology was used to reduce the size of a Recurrent Neural Network (RNN) used for natural language generation on the PTB dataset, where the ground truth setting had a layer size of 850, while the motif setting had a layer size of 3, reducing a NN with $27M$ parameters to a NN with 140 parameters.

After generating a set of N motifs, a synthetic dataset needs to be generated. This dataset is initialized randomly, and has features similar to the original dataset, although it is usually significantly smaller. In their paper, Rawal et al. accomplished this for the MNIST dataset by randomly initializing 40 samples with 10 features each, compared to MNIST's original $60k$ samples with 784 (28x28) features each. For the PTB dataset, they generated 20 samples of length 10, with 10 features each, compared to the PTB datasets $923k$ (training) + $73k$ (validation). This reduction in data fidelity is important to allow the smaller motifs to sufficiently converge on an otherwise much too complicated problem.

Next, all motifs are trained on the synthetic data. After the training procedure, the motifs are compared to their ground truth counterparts based on the loss they achieved during training. The Mean Squared Error (MSE) between the motifs and ground truth NNs is calculated, and used to update the synthetic dataset. This cycle of training motifs and updating the synthetic data is repeated for a fixed number of iterations. After this point, a new, unseen motif can be trained on the synthetic dataset, and its loss after training on the synthetic dataset should reflect the loss in the ground truth setting. Since the motif is much smaller than its full-size equivalent, it allows for much quicker performance estimation than training its full size equivalent.

3 Methodology

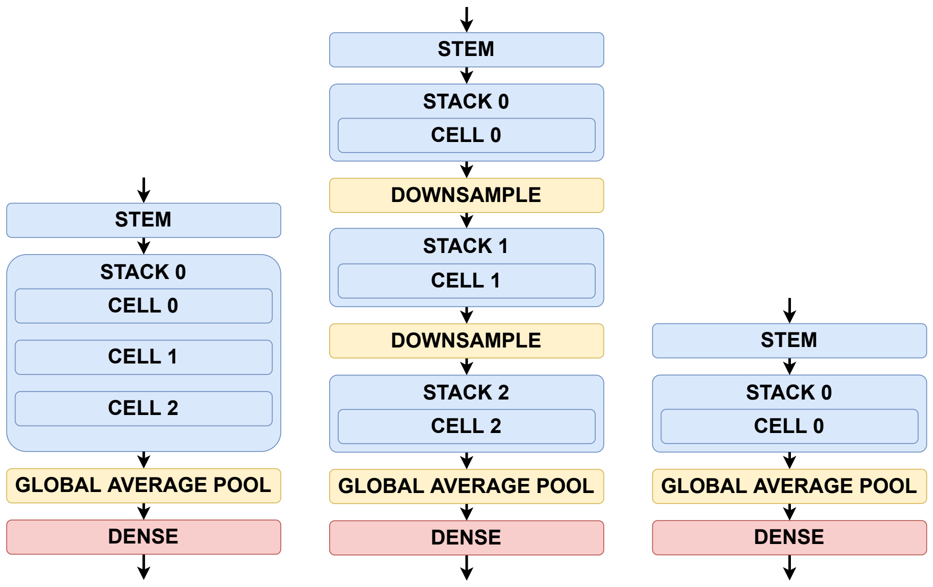
In this section, we will explain the methodology used in this paper, noting where it differs from the methodology described in Rawal et al.'s paper [13].

3.1 Performance Evaluation

In order to evaluate the performance of Convolutional Neural Networks (CNNs) in the ground truth setting, we will make use of the NAS-Bench-101 dataset [17]. This dataset contains data of around $423k$ CNNs, including their image classification accuracy, training times and the total number of parameters present in each. Originally, the normalized MSE loss between the loss in the motif setting and the ground truth setting was used to optimize the synthetic data. Unfortunately, the NAS-Bench-101 dataset does not contain any data on the loss values of the trained architectures. To circumvent this, we will use the accuracy values included in the dataset, more specifically, the accuracy on the validation dataset after training for 108 epochs. The NAS-Bench-101 dataset contains the accuracy on training, test and validation set after 4, 12, 36 and 108 epochs, with no data on the gaps between these epochs. Based on this, we selected the 108-epoch setting, since architectures trained in this setting usually have not fully converged after 36 epochs.

3.2 Motifs

Since we operate in a cell-based NAS context, generating motifs can be done by varying the number of cells. For the generation of the motifs, we use the framework layed down in the NAS-Bench-101 dataset, where neural networks consists of 3 stacks, each with 3 cells. Stacks are separated by downsampling layers, the first stack is preceded by a stem and the last stack is followed by a global average pooling and a dense layer. For an overview of this structure, we refer to the top-left part of Figure 1 in [17]. In NAS-Bench-101, each cell is constructed using a directed acyclic graph that represents that cell architecture. In order to determine the channel counts for each operation in the cell, a simple rule set is used. The number of channels in the input and output node of the graph always remains the same. Each stack has double the number of channels of the previous stack, resulting in 128, 256 and 512 channels in the first, second and third stack respectively. Feature maps along edges that go into the output node are concatenated, while feature maps along edges going into any other node are added. This implies that the output channels will be evenly distributed among all edges incident to the output node. 1x1 convolutions are inserted as necessary, to match channel counts.



(a) Motif A (b) Motif B (c) Motif C

Fig. 1: The different motifs.

While there are many possiblities, we will evaluate three different types of motifs, shown in figure 1. Figure 1a shows Motif A, where the number of cells in each stack has been kept at 3, but the total number of stacks has been reduced to 1. Since there is only 1 stack, the downsampling layers have also been removed. Motif B is shown in Figure 1b, where the number of cells in each stack has been reduced to 1, but 3 stacks were kept. This results in the same number of cells, but a higher amount of trainable parameters. Each downsampling layer in this motif halves the spatial dimensions of the feature maps, but doubles the number of channels, which causes the higher total number of trainable parameters. Finally, Motif C is shown in Figure 1c. Both the number of stacks, and the number of cells in each stack have been reduced to 1. Motif A uses its stem to convolve the initial 3 channels to 32 channels, while motifs B and C's stem convolves to 16 channels.

3.3 Synthetic Data

The synthetic data is initialized by randomly generating 200 images with a resolution of 16x16 pixels and 3 channels. Each image is assigned one of 10 classes, making sure to retain the same class balance as found in the CIFAR-10 dataset [8]. This is in contrast to the CIFAR-10 dataset that was used to generate the NAS-Bench-101 dataset, which contained $50k$ training images, each with a resolution of 32x32 pixels and 3 color channels, also assigned to 1 of 10 classes each.

3.4 Motif Training Procedure

The synthetic petri dish algorithm consists of two nested loops: an inner loop and an outer loop. The inner loop is used to train each motif network on the synthetic data, while the outer loop uses the trained motif networks to update the synthetic dataset.

3.5 Synthetic Data Update Procedure

Updating the synthetic dataset is done through gradient descent. After training the motifs in the inner loop, they are passed to the outer loop for evaluation, during which gradients of the motif outputs with respect to the synthetic data are tracked and later used to update the synthetic data. In order to generate an effective synthetic dataset, we need to find a metric that can be calculated from the outputs of the motifs in a differentiable way, and which can later be used to compute a loss function when comparing this motif metric to the data present in the NAS-Bench-101 dataset. The NAS-Bench-101 dataset contains a number of metrics that might be used for this, but unfortunately, it does not contain any loss values. The dataset does contain accuracy values however, which we will use to compute an outer-loop loss. First, we must derive a differentiable metric that we will compare with the accuracy in the NAS-Bench-101 dataset.

We start by computing the output of the motifs, $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} = f_\theta\left(\mathbf{x}\right) \tag{1}$$

In equation 1 $\hat{\mathbf{y}}$ are the raw logits from the motif network, f_θ. Next, we compute a probability distribution $\hat{\mathbf{p}}$ over the outputs, by applying the softmax function to the logits:

$$\hat{\mathbf{p}} = softmax\left(\hat{\mathbf{y}}\right) \tag{2}$$

Next, we determine which element in the ground-truth vector $\mathbf{y}$ has the highest value, and store its index in i. We note that, while the argmax operation itself is not differentiable, we can still use its output to index into $\hat{\mathbf{y}}$ or $\hat{\mathbf{p}}$ without jeopardizing the differentiability of the result:

$$i = argmax\left(\mathbf{y}\right) \tag{3}$$

Finally, we will use the element from $\hat{p}$ at index i as the performance metric:

$$\hat{z} = \hat{\mathbf{p}}\left[i\right] \tag{4}$$

Intuitively, this metric should increase as classifier performance improves, since it represents the predicted probability of the true class by the motif. We can also prove this, by computing the derivative of the inner loop loss function of the motifs with respect to this metric. We expect this derivative to be negative, indicating that, as the metric improves, the inner loop loss function should decrease. The motifs were trained using cross-entropy loss between the ground-truth labels as a one-hot vector with K elements, q, and the probabilities returned from the motifs, $\hat{p}$:

$$\frac{d\mathcal{L}}{d\hat{z}} = \frac{d}{d\hat{z}} - \sum_{j=1}^{K} \mathbf{q}\left[j\right] \cdot log\left(\hat{\mathbf{p}}\left[j\right]\right) \tag{5}$$

Since q is a one-hot vector, only 1 element (the element at index i) will actually contribute to this sum, we thus reduce the sum to this element:

$$\frac{d\mathcal{L}}{d\hat{z}} = \frac{d}{d\hat{z}} - log\left(\hat{\mathbf{p}}\left[i\right]\right) = \frac{d}{d\hat{z}} - log\left(\hat{z}\right) \tag{6}$$

From this, we can compute the final derivative for $\mathcal{L}$ with respect to $\hat{z}$:

$$\frac{d\mathcal{L}}{d\hat{z}} = -\frac{1}{\hat{z}} \tag{7}$$

We note that $\hat{z}$ is a probability, thus, in equation 7 the derivative is always negative, showing that an increase in $\hat{z}$ correlates with a reduction of the inner loop loss.

Using the metric from equation 7, we now need to formulate a loss function that compares the motif metric to the accuracy on the validation dataset after 108 epochs of training that is included in the NAS-Bench-101 dataset. Since we are comparing two different metrics, we will allow for a linear relationship to exist between both, by optimizing the Pearson correlation coefficient between them, rather than optimizing the difference between them directly. We formulate this as follows:

$$\mathcal{L}(z, \hat{z}) = 1 - |r_{z\hat{z}}| \tag{8}$$

In equation 8, $r_{z\hat{z}}$ represents the Pearson correlation between two datasets: z and $\hat{z}$. We take the absolute value of the Pearson correlation, since, for our purposes, a perfect negative correlation is equally valuable as a perfect positive correlation. Since we are solving a minimization problem, the loss function will be formulated as $1 - |r_{z\hat{z}}|$, rather than $|r_{z\hat{z}}|$, which represents a maximization problem.

4 Experiments

Every outer loop iteration considers a batch of 6 motifs. The synthetic data was updated using the ADAM optimizer with a learning rate of 1×10^{-2} and no weight decay. When training the motifs in the inner loop, we also used the ADAM optimizer with a learning rate of 5×10^{-5}, and a batch size of 20, applying weight decay regularization with a factor of 1×10^{-6}. The inner loop uses a cross-entropy loss function. Every motif was trained for 100 epochs during the inner loop, and we carried out a total of 170 outer loop epochs. In order to speed up the inner loop, all motifs are trained in parallel.

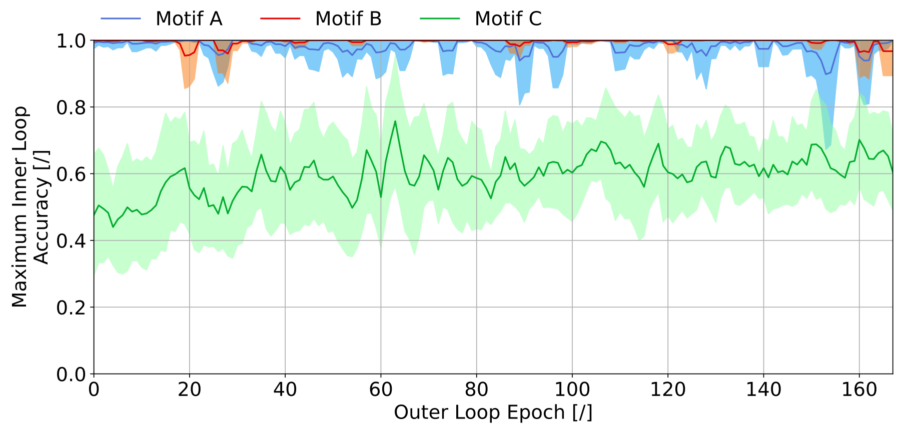


Fig. 2: The mean of the maximum accuracy achieved by the six motifs in each outer loop epoch. The filled area corresponds to the mean +/- 1 standard deviation. Data was run through a moving average filter with filter size 3 before plotting.

5 Results

We initially generated one synthetic data set for each motif shown in figure 1. Based on these results, we decided to carry out a second set of experiments, which involved training motifs A and B with a shorter inner loop training cycle.

In a NAS setting, the goal is to find the best performing NN architecture. In other words, it is important that we are able to determine an accurate ranking of NN architectures, rather than being able to accurately predict the exact performance of each NN. Based on this observation, we will use the Kendall-Tau ranking correlation coefficient to evaluate the synthetic dataset.

5.1 Motif Performance

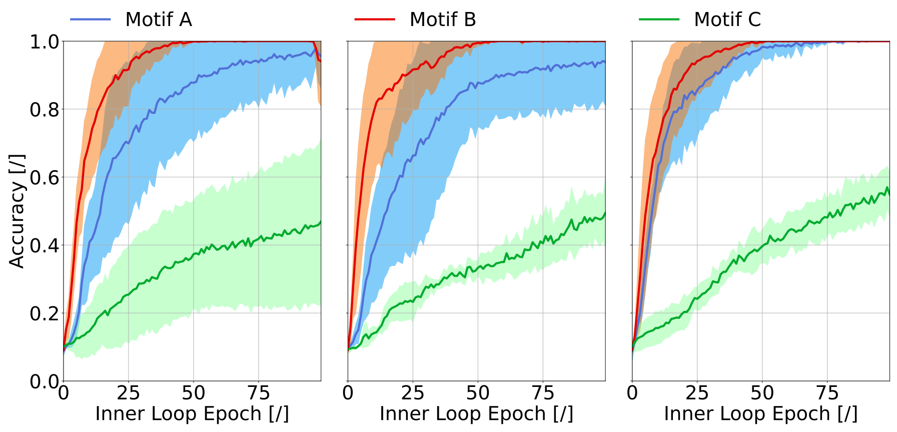


Fig. 3: The inner loop training curve at the beginning (Outer loop epoch 0, left), middle (Outer loop epoch 85, middle) and end (Outer loop epoch 169, right) of the synthetic petri dish algorithm. The line shows the mean of the 6 motifs considered in that outer loop epoch, with the shaded area representing the mean +/- 1 standard deviation.

From figure 2 we can see that motif A and B have no trouble consistently becoming perfect classifiers for the synthethic data, with respective mean accuracies over the last 5 epochs of 97.81% and 99.43%. Since these two motifs have the most trainable parameters, it is likely that they overfit the synthetic dataset. Motif C seems to exhibit a small increase in mean maximum accuracy as the synthetic dataset generation progresses, starting at 48.5% accuracy (Mean of the first 5 epochs), and ending at 64.5% (Mean of the last 5 epochs). This is also reflected in figure 3, showing the training curves during different inner loops.

An important observation here is the fact that motifs A and B are almost always perfect classifiers, making it very difficult for them to approximate the accuracy ranking in the NAS-Bench-101 dataset. In fact, when using motif A, we noticed that in 14.71% of all epochs, all 6 motif used in the outer loop became perfect classifiers. For motif B this number is even higher at 43.53%, while this never occurred with motif C.

We can also see that motif B outperforms motif A relatively consistently, by having higher mean accuracy and a lower standard deviation. We suspect this happens because for the same architecture, motif B has a higher number of trainable parameters in around 80% of cases. The fact that motif A still has a higher parameter count in 20% of cases can be attributed to the rules used in NAS-Bench-101 to allocate channels, and the use of projection convolutions to ensure the channel counts of the different convolutions in a cell match.

5.2 Synthetic Dataset Generation

Next, we will analyze the process of generating a synthetic dataset.

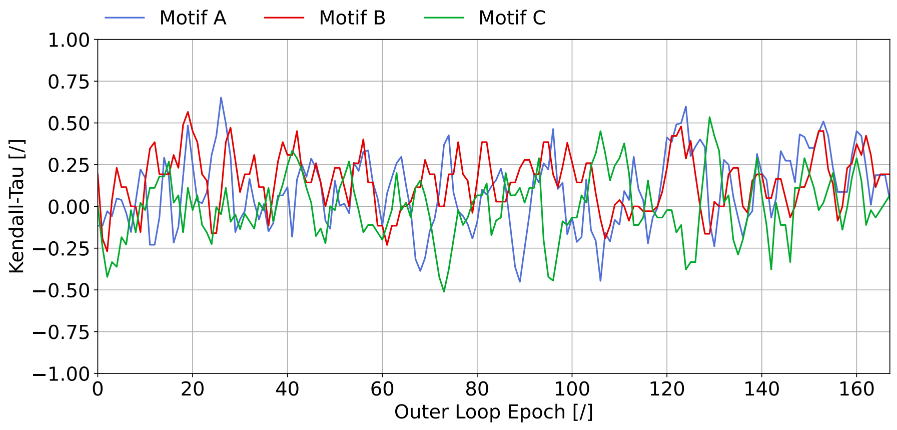


Fig. 4: The Kendall-Tau Ranking Correlation computed on the 6 motifs used in the outer loop update. Data was run through a moving average filter with filter size 3 before plotting.

From figure 4 we can clearly see that the outer loop training process was unable to produce any meaningful improvement on the synthetic data. We note that in figure 4, it was not always possible to compute a ranking correlation. When all motifs attained 100% accuracy, it is impossible to determine the ranking, and no meaningful Kendall-Tau value can be computed. We assigned these points the worst possible correlation score of 0. As mentioned before, this happened 14.71% of the time with motif A, 43.53% with motif B and never for motif

C. The overall mean of the Kendall-Tau correlation for motif A, B and C was 0.092, 0.152 and −0.002 respectively when including the 0 values, or 0.108, 0.270 and −0.002 when excluding them. Interestingly, while there were clear differences between motifs in the inner loop, these differences are much less pronounced in the outer loop, with motif B performing best and motif C performing worst.

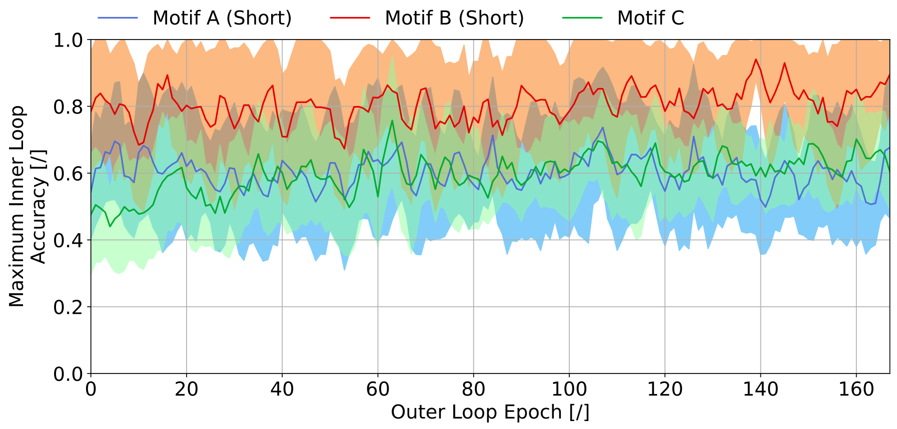


Fig. 5: The mean of the maximum accuracy achieved by the six motifs in each outer loop epoch with only 16 training epochs in the inner loop. Motif C was trained with 100 epochs in the inner loop. Data was run through a moving average filter with filter size 3 before plotting.

5.3 Shortened Training

In section 5.1 we saw that motifs A and B were able to overfit the synthetic data quite easily, which would make any attempt at further synthetic data generation difficult, since there is no performance difference between perfect classifiers. In this section we will analyze whether stopping the inner loop training early could mitigate some of the these issues, and how this influences the performance of the outer loop.

From figure 5 we can see that lowering the number of training epochs on motif A and B seems to have brought them more in line with motif C. We see that, even in a reduced training setting, motif B still outperforms motif A by around 20%, with motif B attaining a mean accuracy over the final 5 epochs of 85.13%, compared to motif A's 59.27%. Similar to our earlier observations, we notice that in the reduced training setting, there is still little to no evolution in the motif's accuracy as the synthetic data is updated. The training curves in figure 6 tell a similar story. Interestingly, from figure 6 we can see that motif B is quite quick to converge, and seems much closer to convergence than motif

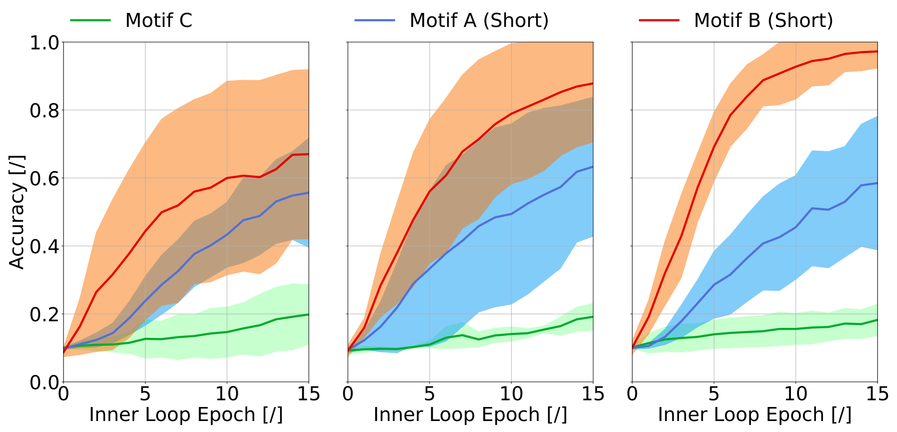


Fig. 6: The inner loop training curve at the beginning (Outer loop epoch 0, left), middle (Outer loop epoch 85, middle) and end (Outer loop epoch 169, right) of the synthetic petri dish algorithm. Motifs A and B were trained for 16 epochs, the training curve for motif C was simply truncated at 16 epochs to provide a comparison. The line shows the mean of the 6 motifs considered in that outer loop epoch, with the shaded area representing the mean +/- 1 standard deviation.

A and C, despite its higher parameter count. This is likely the result of the presence of pooling layers, which may improve the networks robustness to noisy characteristics of the synthetic dataset.

While the inner loop accuracy shows different results from before, the outer loop was still unable to produce any meaningful ranking correlation, as can be seen in figure 7. The shorter training procedure has eliminated instances where all motifs are perfect classifiers, for each of the three motifs. As expected, training motif B for a shorter amount of time reduces its Kendall-Tau correlation, down to 0.209, with motif A also doing worse at 0.043.

5.4 Computational Efficiency

The main goal of a performance estimation technique in NAS is to estimate the performance of a NN faster than training the NN in the ground truth setting. In their comparison of different performance estimation strategies, White et al. [15] define the computational cost of using a performance estimator as consisting of two separate components: initialization time and query time. In this section, we will use these criteria to evaluate the synthetic petri dish performance estimation method.

We start by examining the initialization time in table 1. Interestingly, we notice that motif A is slower than motif B, even though in around 80% of cases, it has fewer trainable parameters. Since motif B also outperforms motif A in

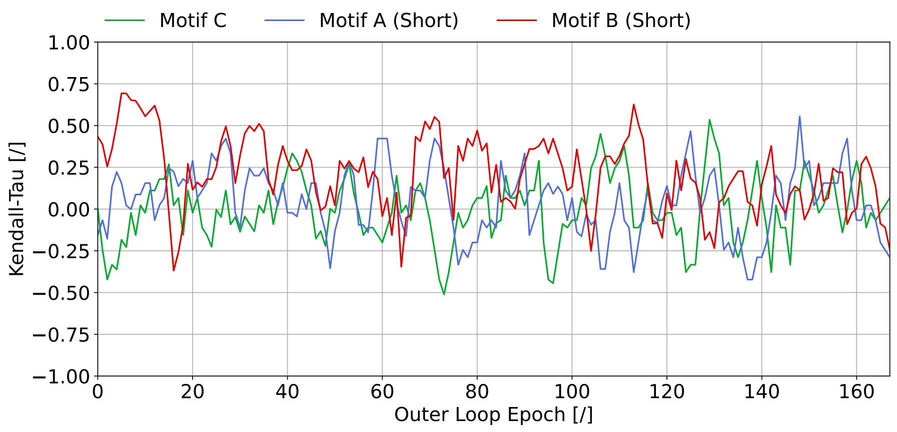


Fig. 7: The Kendall-Tau Ranking Correlation computed on the 6 motifs used in the outer loop update, with only 16 training epochs in the inner loop. Motif C was trained with 100 epochs in the inner loop. Data was run through a moving average filter with filter size 3 before plotting to show trends more clearly.

Setting	**Minimum**	**Mean** ($\pm$ **Std.**)	**Maximum**
A	$0.193s$	$4.282s \pm 2.078s$	$10.361s$
B	$0.174s$	$3.471s \pm 1.701s$	$10.397s$
C	$0.075s$	$1.427s \pm 0.628s$	$3.259s$
NAS-Bench-101	$2.632s$	$17.893s \pm 8.508s$	$51.511s$

Table 1: The time necessary to complete 1 inner loop epoch of training.

terms of ranking correlation, we can see that motif A is a suboptimal choice of motif, while motifs B and C provide trade-offs in terms of ranking correlation and initialization time. We note that the training time for NAS-Bench-101 was taken from the dataset, which used TPU v2 accelerators. The data for motifs was gathered on AMD Ryzen 9 5900X CPU. We trained the motifs using CPUs, rather than GPUs since the motifs and synthetic dataset are so small, the benefits from executing them on GPUs tend to be outweighed by the time it takes to transfer data between CPU and GPU.

Table 2 shows the initialization time calculation for each motif, and compares our approach to that of Rawal et al.[13] As expected, due to the use of a new set of motifs in each outer loop epoch, our method has a larger computational requirement for acquiring the necessary ground truth data than the original method. Unfortunately, we were unable to measure the impact this had in preventing the overfitting of the synthetic dataset to a small subset of motifs, due to the non-convergence of our outer loop.

Next we turn our attention to query time. Since querying the synthetic petri dish is equivalent to training one motif on the synthetic dataset to convergence,

Setting	**Subitem**	**Rawal et al.[13]**	**This work**
NAS-Bench-101	Training Ground Truth Networks	$30 \times 108 \times 17.893s = 57.973s \times 10^3 \approx$ 0.67 days	$6 \times 170 \times 108 \times 17.893s = 1.971 \times 10^6 s \approx 23$ days
Motif A	Generating Synthetic Petri Dish	$30 \times 170 \times 100 \times 4.282s = 2.184 \times 10^6 s \approx 25$ days	$6 \times 170 \times 100 \times 4.282s = 436.764 \times 10^3 s \approx$ 5 days
	Total	**26 days**	**28 days**
Motif B	Generating Synthetic Petri Dish	$30 \times 170 \times 100 \times 3.471s = 1.770 \times 10^6 s \approx 20$ days	$6 \times 170 \times 100 \times 3.471s = 354.042 \times 10^3 s \approx$ 4 days
	Total	**21 days**	**27 days**
Motif C	Generating Synthetic Petri Dish	$30 \times 170 \times 100 \times 1.427s = 727.770 \times 10^3 \approx 8$ days	$6 \times 170 \times 100 \times 1.427s = 145.554 \times 10^3 s \approx$ 2 days
	Total	**9 days**	**24 days**

Table 2: The total initialization time of the synthetic petri dish method

assuming 100 epochs of training on the synthetic dataset, we find a query time of $428.2s$ for motif A, $347.1s$ for motif B, and $142.7s$ for motif C.

When comparing our results to those in [15], we see that, for the NAS-Bench-101 benchmark, in the high initialization ($10^6 s$), medium query time region ($10^2 s$), BANANAS is the optimal predictor. From figure 4 (right), we see that in the NAS-Bench-101 setting, given $10^6 s$ of initialization time, no method achieves less than 0.2 Kendall-Tau ranking correlation, while the best methods achieve 0.8 Kendall-Tau ranking correlation.

6 Discussion and Conclusions

In this publication, we analyzed the use of the synthetic petri dish method in the image classification domain. We note that, even though the motif networks have no problem converging (and overfitting) on the synthetic dataset, this does not guarantee that the outer loop optimization process will work. Interestingly, this can lead to a chicken-and-egg problem between the synthetic data (which does not result in an accurate distribution of motif networks) and motif networks (which have been trained on a non-representative dataset). This likely hampers the stability of the synthetic petri dish method.

We selected a new sample of 6 architectures to be used in each outer loop. This leads to a total of 1020 architectures which all require a ground truth evaluation in order to generate the synthetic data. Our analysis of the computational requirements in section 5.4 shows that this is a major downside of our methodology, and underlines the significance of an improvement in ranking correlation before such trade-offs are worthwhile. It also highlights the importance of assessing the generalization power of the synthetic petri dish estimator, something we were unfortunately unable to do, due to bad outer loop performance.

Finally, we evaluated three different choices of motifs. Despite the non-convergence of our outer loop, we were able to determine that motif A is a suboptimal choice of motif, when evaluating its training time and its ranking performance.

We also saw that when performing full-length motif training, motif A and B were frequently perfect classifiers, making it very difficult to relate their performance to that in the ground truth setting. This shows that matching the size of the synthetic dataset to the motifs is important for the method to work.

7 Future work

The main issue with the current methodology is the lack of convergence in the outer loop. The most likely cause for this is a bad choice of outer loop loss function. This can be likely be resolved if data is available on the loss values of the motifs in the ground truth setting, by using a different dataset like NATS-Bench [4] or by recording additional data about the NAS-Bench-101 dataset, so loss values can be used directly.

An alternative could be to optimize the MSE loss between motif and ground truth accuracy directly. From the accuracies obtained by motifs A and B, we can see that with a correctly sized dataset, motif accuracies are in the same range as ground-truth accuracies, and could be used as a good performance predictor. Additionally, moving away from using a correlation coefficient in the loss function would lessen the severity of motifs becoming perfect classifiers, since the performance difference between a perfect motif and an imperfect network in the ground truth setting is still meaningful.

Unfortunately, outer loop convergence is a necessity before a more in-depth of analysis of the merits and demerits of different types of motifs can be conducted.

Acknowledgements This research received funding from the Flemish Government (AI Research Program).
This work was supported by the Research Foundation Flanders (FWO) under Grant Number 1SC8821N.

References

1. Arber, Z., Stefan, A., Frank, H.: Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. In: International Conference in Machine Learning (2018)
2. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)
3. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (Jun 2015), https://www.aaai.org/ocs/index.php/IJCAI/IJCAI15/paper/view/11468

4. Dong, X., Liu, L., Musial, K., Gabrys, B.: Nats-bench: Benchmarking nas algorithms for architecture topology and size. IEEE Transactions on Pattern Analysis and Machine Intelligence **44**(7), 3634–3646 (2022). https://doi.org/10.1109/TPAMI.2021.3054824
5. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via lamarckian evolution. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=ByME42AqK7
6. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: A survey. Journal of Machine Learning Research **20**(55), 1–21 (2019), http://jmlr.org/papers/v20/18-598.html
7. Klein, A., Falkner, S., Springenberg, J.T., Hutter, F.: Learning curve prediction with bayesian neural networks. In: International Conference on Learning Representations (2017), https://openreview.net/forum?id=S11KBYclx
8. Krizhevsky, A., Nair, V., Hinton, G.: Cifar-10 (canadian institute for advanced research) (2009), http://www.cs.toronto.edu/~kriz/cifar.html
9. Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-tzur, J., Hardt, M., Recht, B., Talwalkar, A.: A System for Massively Parallel Hyperparameter Tuning. Proceedings of Machine Learning and Systems **2**, 230–246 (Mar 2020), https://proceedings.mlsys.org/paper/2020/hash/f4b9ec30ad9f68f89b29639786cb62ef-Abstract.html
10. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable architecture search. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=S1eYHoC5FX
11. Marcus, M.P., Santorini, B., Marcinkiewicz, M.A.: Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics **19**(2), 313–330 (1993), https://aclanthology.org/J93-2004
12. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient Neural Architecture Search via Parameters Sharing. In: Proceedings of the 35th International Conference on Machine Learning. pp. 4095–4104. PMLR (Jul 2018), https://proceedings.mlr.press/v80/pham18a.html, iSSN: 2640-3498
13. Rawal, A., Lehman, J., Such, F.P., Clune, J., Stanley, K.O.: Synthetic Petri Dish: A Novel Surrogate Model for Rapid Architecture Search (arXiv:2005.13092) (May 2020). https://doi.org/10.48550/arXiv.2005.13092, http://arxiv.org/abs/2005.13092
14. Runge, F., Stoll, D., Falkner, S., Hutter, F.: Learning to design RNA. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=ByfyHh05tQ
15. White, C., Zela, A., Ru, R., Liu, Y., Hutter, F.: How powerful are performance predictors in neural architecture search? **34**, 28454–28469 (2021), https://proceedings.neurips.cc/paper/2021/file/ef575e8837d065a1683c022d2077d342-Paper.pdf
16. Xue, F., Qi, Y., Xin, J.: RARTS: An efficient first-order relaxed architecture search method. IEEE Access **10**, 65901–65912 (2022). https://doi.org/10.1109/access.2022.3185095, https://doi.org/10.1109%2Faccess.2022.3185095
17. Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: NAS-bench-101: Towards reproducible neural architecture search. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 7105–7114. PMLR (09–15 Jun 2019), https://proceedings.mlr.press/v97/ying19a.html
18. Zoph, B., Le, Q.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (2017), https://openreview.net/forum?id=r1Ue8Hcxg