Neural Networks 144 (2021) 384–393

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Neural network surgery: Combining training with topology optimization

Elisabeth J. Schiessler [a,*], Roland C. Aydin [a,*], Kevin Linka [b], Christian J. Cyron [a]

[a] Helmholtz-Zentrum Hereon, Institute of Material Systems Modeling, Dept. of Machine Learning and Data, Max–Planck–Straße 1, 21502 Geesthacht, Germany
[b] Hamburg University of Technology, Institute of Continuum and Materials Mechanics, Eißendorfer Straße 42, 21073 Hamburg, Germany

ARTICLE INFO

Article history:
Received 31 March 2021
Received in revised form 27 August 2021
Accepted 30 August 2021
Available online 7 September 2021

Keywords:
Neural architecture search
Topology optimization
Singular value decomposition
Genetic algorithm

ABSTRACT

With ever increasing computational capacities, neural networks become more and more proficient at solving complex tasks. However, picking a sufficiently good network topology usually relies on expert human knowledge. Neural architecture search aims to reduce the extent of expertise that is needed. Modern architecture search techniques often rely on immense computational power, or apply trained meta-controllers for decision making. We develop a framework for a genetic algorithm that is both computationally cheap and makes decisions based on mathematical criteria rather than trained parameters. It is a hybrid approach that fuses training and topology optimization together into one process. Structural modifications that are performed include adding or removing layers of neurons, with some re-training applied to make up for any incurred change in input–output behaviour. Our ansatz is tested on several benchmark datasets with limited computational overhead compared to training only the baseline. This algorithm can achieve a significant increase in accuracy (as compared to a fully trained baseline), rescue insufficient topologies that in their current state are only able to learn to a limited extent, and dynamically reduce network size without loss in achieved accuracy. On standard ML datasets, accuracy improvements compared to baseline performance can range from 20% for well performing starting topologies to more than 40% in case of insufficient baselines, or reduce network size by almost 15%.

1. Introduction

A common problem for any given machine learning task making use of artificial neural networks (ANNs) is how to choose a sufficiently good network topology. Picking one that is too small may not yield acceptable prediction accuracy. To improve results, one can keep adding structural elements to the network until the desired accuracy value has been reached. Too large networks on the other hand may cause an explosion in computational cost for both training and evaluation. Finding the optimal balance is heavily dependent on the given task, dataset and further hyperparameters, and often requires expert domain knowledge. A priori optimization is not easily possible, since reliable estimates on network behaviour already require training results, and no generalization exists which topology will fit which problem. Researchers have applied a number of search strategies such as random search (Li & Talwalkar, 2019), Bayesian optimization (Kandasamy, Neiswanger, Schneider, Poczos, & Xing, 2018), reinforcement learning (Zoph & Le, 2017), and gradient-based methods (Dong & Yang, 2019; Li, Khodak, Balcan, & Talwalkar, 2021; Liu, Simonyan, & Yang, 2019; Wang, Cheng, Chen, Tang, & Hsieh, 2021; Xu et al., 2020). Another technique applied since at least (Miller, Todd, & Hegde, 1989) are so called (neuro-) evolutionary algorithms. These algorithms serve to evolve the network architecture, often also training network weights at the same time (Elsken, Metzen, & Hutter, 2019).

In this paper we propose a novel training regime incorporating a genetic algorithm that reduces computational cost compared to state of the art approaches of this kind (Dong & Yang, 2019; Li & Talwalkar, 2019). We achieve this by re-using network weights for competing modification candidates instead of retraining each net from scratch, branching off modification candidates during training, and letting them compete against each other until a new main branch is selected. This fuses the evolutionary optimization paradigm with the ANN training into an integrated framework that folds both processes into a single training/topology optimization hybrid. As such, evolutionary steps are not carried out by a meta-controller or other black-box-like implementations, but instead make use of mathematical tools such as singular value decomposition (SVD) and the Bayesian information criterion (BIC) (Schwarz, 1978) for network weight analysis, decision

* Equal contribution.
E-mail addresses: elisabeth.schiessler@hereon.de (E.J. Schiessler), roland.aydin@hereon.de (R.C. Aydin).

https://doi.org/10.1016/j.neunet.2021.08.034

making, and structural modifications. Network modifications are performed by adapting existing weights such as to incur minimal changes to input–output behaviour.

Our framework for a combined ANN training and neural architecture search consists of three main components: a module that can perform a number of minimally invasive network operations ("surgeries"), a module that analyses network weights and can give recommendations which modifications are most likely to increase (validation) accuracy, and finally a module that serves as a genetic algorithm (the "Surgeon"), containing the former two while gradually evolving any given starting network. With the Surgeon, we are able to evolve and improve models for several benchmark datasets and varying starting topologies. We achieve particularly good results on starting topologies that would a posteriori have proven to be suboptimal. A great benefit of our approach is that it adds topology optimization to ML training while incurring very limited additional computational costs. Convergence is reached for all test cases within a few hours.

The supporting code can be accessed via https://github.com/ElisabethJS/neural-network-surgery.

This paper contributes a computationally cheap ansatz for a genetic neural architecture search algorithm that makes evolutionary decisions based on mathematical analysis.

2. Related work

Neural architecture search (NAS) has been an increasingly popular research topic for many years (Elsken et al., 2019), starting as early as Miller et al. (1989), who presented one of the earliest neuro-evolutionary algorithms to search for suitable network topologies. Recent approaches by Dong and Yang (2019), Li and Talwalkar (2019), and Zoph and Le (2017) reach competitive performance on benchmark datasets such as CIFAR-10. However, this often comes at the cost of vast computational resources, with Zoph and Le (2017) making use of up to 800 GPUs for several weeks.

Cai, Chen, Zhang, Yu, and Wang (2018) attempt to reduce computational costs by re-using network weights, as well as training and applying a reinforcement meta-controller for structural decisions. They make use of a number of function-preserving transformations (net2net) introduced by Chen, Goodfellow, and Shlens (2016), and extend them to allow also non-sequential network structures, such as DenseNet (Huang, Liu, van der Maaten, & Weinberger, 2017). DiMattina and Zhang (2010) introduce and rigorously prove conditions, under which gradual changes of the parametrization of a neural network are possible, while keeping the input–output behaviour constant.

İrsoy and Alpaydın (2020) learn the network structure via so-called "budding perceptrons", in which an extra parameter is learned for each layer, that indicates whether or not any given node needs to branch out again or be removed altogether. Their method focuses on growing the network to the required size from a minimal starting topology. Frankle and Carbin (2019) present a method to identify particularly good network initializations that can train sparse networks to competitive accuracy. Another approach in NAS is to prune down from a larger starting topology (Blalock, Gonzalez Ortiz, Frankle, & Guttag, 2020). Popular pruning techniques include applying SVD to existing network weights (Denton, Zaremba, Bruna, LeCun, & Fergus, 2014; Girshick, 2015; Xue, Li, & Gong, 2013).

There are also a number of neural architecture search strategies that do not depend on manual network modifications. Liu et al. (2019) introduced DARTS, a method for differential architecture search that re-formulates the task of searching for network architectures as a graph optimization problem, where all possible network configurations are represented as nodes on a directed acyclic graph. This technique has rapidly become very popular and has seen a great number of extensions in various directions, such as Dong and Yang (2019), Li et al. (2021), Wang et al. (2021) and Xu et al. (2020). The downside of the DARTS based algorithms is that all possible network variations are predefined in advance and cannot be adapted during training based on the network state. Additionally, since all nodes of a certain hierarchy level need to be interchangeable or even skippable, the connections between network blocks are highly restricted with regard to network structure.

The novelty of our research lies in combining existing tools such as net2net (Chen et al., 2016) and SVD with a genetic algorithm that modifies the given network in a decision based process instead of utilizing a black-box like decision module, while retaining a very high level of structural freedom. We repeatedly generate functionally equivalent but structurally different networks that are then trained for a short number of epochs, after which their performance is compared and the best candidates are retained. Ultimately this yields a fully trained neural network with an optimized topology. To our best knowledge no such method has yet been proposed.

3. Methods

This work introduces and utilizes three main modules:

- modification module: performs network modifications ("surgeries") so as to incur minimal changes to input–output behaviour.
- recommendation module: analyses network weights and gives recommendations on which operations are most likely to improve network accuracy.
- "the Surgeon": a genetic algorithm that links the above two modules, and gradually evolves a given starting network.

3.1. The modification module

We want to be able to carry out a number of different modifications that can restructure network architecture while keeping the input–output behaviour intact, and in particular avoid loss of prediction quality. In cases where this is not possible we aim for minimal impact changes instead. This yields network variations that are structurally different but functionally equivalent.

In mathematical terms, let the output of a dense layer be given by

$$f(x) = \Phi(\mathcal{A} \cdot x + b), \tag{1}$$

with activation function Φ, weight matrix $\mathcal{A}$ and bias vector b, as well as an arbitrary input x to that layer. We perform four different types of modifications, namely adding or removing neurons or whole layers. In particular, we are looking for $\tilde{f}(x)$ such that

$$\tilde{f}(x) \equiv f(x)\ \forall x, \text{ but } \tilde{f} \neq f. \tag{2}$$

Activation functions used in neural networks are usually nonlinear, or piecewise linear at best. Thus changing the activation function will in general not produce equal results for arbitrary input values. Finding modifications that still suffice Eq. (2) is therefore synonymous to adequately adapting the affected network weight matrices and bias vectors.

Adding layers. Under (at least) piecewise linear activation functions such as ReLu, one can always add neurons to a hidden layer, or even add whole layers, without any change to the overall network behaviour (Chen et al., 2016). Adding a whole layer is done by using an identity matrix as a new weight matrix for this inserted layer. In particular, the added layer will have the same

number n of neurons as the following layer. We initialize the weights as an identity matrix $\mathcal{I} \in \mathbb{R}^n$. Let Φ denote the activation function of a dense layer and x an arbitrary input, then

$$\Phi(x) = \Phi(\mathcal{I}\Phi(x)) \tag{3}$$

if and only if Φ is at least piecewise linear. In particular, for ReLu activation, adding layers without changing the input–output behaviour is possible.

Adding neurons. For adding neurons to an existing layer, consider the following example. Let $x \in \mathbb{R}^n$ be the input to a layer with weights $\mathcal{A} \in \mathbb{R}^{m\times n}$, and $\mathcal{B} \in \mathbb{R}^{k\times m}$ the next layer's weights, and let the layer's bias vector be zero w.l.o.g. Assume the activation function Φ acting between the two layers to be an identity mapping. Then the output $y \in \mathbb{R}^k$ is given by

$$y = \mathcal{B} \cdot \Phi(\mathcal{A}x) = \mathcal{B} \cdot (\mathcal{A}x). \tag{4}$$

In particular, the jth entry of y is

$$y_j = (b_{j1}, b_{j2}, \ldots, b_{jm}) \cdot \begin{bmatrix} \sum_{i=1}^n x_i a_{1i} \\ \sum_{i=1}^n x_i a_{2i} \\ \vdots \\ \sum_{i=1}^n x_i a_{mi} \end{bmatrix} \tag{5}$$

$$= b_{j1} \sum_{i=1}^n x_i a_{1i} + \cdots + b_{jm} \sum_{i=1}^n x_i a_{1m}, \tag{6}$$

where a_{ij} is the ij element of $\mathcal{A}$, and b_{ij} the ij element of $\mathcal{B}$. We now arbitrarily pick unit m and duplicate its incoming and outgoing weights. Note that we also have to divide by 2 in order to keep the total sum constant.

$$y_j = b_{j1} \sum_{i=1}^n x_i a_{1i} + \cdots + \frac{b_{jm}}{2} \sum_{i=1}^n x_i a_{1m} + \frac{b_{jm}}{2} \sum_{i=1}^n x_i a_{1m} \tag{7}$$

$$= (b_{j1}, b_{j2}, \ldots, \frac{b_{jm}}{2}, \frac{b_{jm}}{2}) \cdot \begin{bmatrix} \sum_{i=1}^n x_i a_{1i} \\ \sum_{i=1}^n x_i a_{2i} \\ \vdots \\ \sum_{i=1}^n x_i a_{mi} \\ \sum_{i=1}^n x_i a_{mi} \end{bmatrix} \tag{8}$$

This methods yields the exact same output $y \in \mathbb{R}^k$ given input $x \in \mathbb{R}^n$, but the weight matrices $\mathcal{A}$ and $\mathcal{B}$ are now of dimension $(m+1, n)$ and $(k, m+1)$ respectively, and can be extended to include a non-zero bias term in a similar fashion.

Recall now that we chose the activation Φ to be the identity mapping. For any other activation function, the equation in line (7) holds true, if and only if this activation is at least piecewise linear.

Chen et al. (2016) base their Net2WiderNet and Net2DeeperNet transformation on these steps.

Removing neurons. Removing neurons from a layer is rarely possible without changing the input–output behaviour unless some of the units are degenerate to begin with, i.e. the weight matrix is of reduced rank. For a modification with minimal impact on input–output behaviour we need the closest possible projection onto a lower rank subspace. As a measure for closeness we employ the *Frobenius norm*, defined as follows.

Let $\|\cdot\|_F$ denote the *Frobenius norm*, with

$$\|\mathcal{A}\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 = \text{trace}(\mathcal{A}^T\mathcal{A}), \tag{9}$$

where $\mathcal{A} \in \mathbb{R}^{m\times n}$ is an arbitrary, real-valued matrix of rank $k \leq \min(m, n)$.

The Eckart–Young(–Mirsky) theorem states that the closest projection onto a lower rank subspace can be found by applying Singular Value Decomposition (SVD) to the weight matrix:

Let $\mathcal{A} \in \mathbb{R}^{m\times n}$ be the (weight) matrix of some layer L_j of interest in a neural network, then there exists a representation

$$\mathcal{A} = \mathcal{U}\Sigma\mathcal{V}^T \tag{10}$$

with orthogonal $\mathcal{U} \in \mathbb{R}^{m\times m}$, $\mathcal{V} \in \mathbb{R}^{n\times n}$ and rectangular diagonal $\Sigma \in \mathbb{R}^{m\times n}$ with $k \leq \min(m, n)$ non-negative real entries σ_i along its diagonal. This representation is called the *Singular Value Decomposition* of $\mathcal{A}$. The σ_i are called *singular values* of $\mathcal{A}$, and are usually given in descending order, i.e. $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$. It can be shown that $\|\mathcal{A}\|_F^2 = \sum \sigma_i^2(\mathcal{A})$, since $\sigma_i(\mathcal{A})$ corresponds to the square root of the ith non-zero eigenvalue of $\mathcal{A}^T\mathcal{A}$. Note that $\mathcal{U}$ and $\mathcal{V}$ are projections from the $(m \times n)$ vector space spanned by $\mathcal{A}$ to the $(k \times k)$ vector space spanned by Σ and back.

In order to reduce the rank of $\mathcal{A}$ to $r < k$, we set $\sigma_i = 0\ \forall i > r$, and drop associated columns/rows of $\mathcal{U}$ and $\mathcal{V}$, which is feasible since by definition $\sigma_i \geq \sigma_{i+1}$. We then project to a smaller matrix $\tilde{\mathcal{A}} \in \mathbb{R}^{m\times r}$ by computing $\mathcal{U}\Sigma$. In order to properly re-connect the reduced layer L_j to the following layer L_{j+1}, we have to modify L_{j+1}'s weight matrix to accept input of length r. In particular, we need to project back to the original layer size, to ensure that the layer weight shapes match again. We achieve this by multiplying from the left the weight matrix of L_{j+1} with $\mathcal{V}^T$.

Projecting onto a lower rank subspace by setting singular values to zero is sometimes called truncated SVD, and is used in network pruning (Denton et al., 2014; Girshick, 2015; Xue et al., 2013). This technique adds an intermediate layer of (potentially much fewer) neurons, which in case of very large weight matrices can drastically reduce the overall connection count.

Note that this projection method is not concerned at all with a potential activation function after the network's modified layer. As mentioned previously, changing a layer's activation function is in general not possible without changes to input/output behaviour, since activation functions are usually non-linear. We therefore do not perform any additional modifications to counterbalance a change of activation.

Removing layers. We remove whole layers in a similar fashion. Let $\mathcal{A}_j, b_j$ be the weight matrix and bias vector of some dense Layer L_j, and $\mathcal{A}_{j+1}, b_{j+1}$ those of the subsequent Layer L_{j+1}. We remove the Layer L_j by simply dropping it, and modify L_{j+1}'s weights by matrix multiplication, yielding $\tilde{L}_{j+1}$

$$\tilde{\mathcal{A}}_{j+1} = \mathcal{A}_j \cdot \mathcal{A}_{j+1}, \tag{11}$$

$$\tilde{b}_{j+1} = b_j \cdot \mathcal{A}_{j+1} + b_{j+1}. \tag{12}$$

This again ignores any activation function between the two layers, and will thus cause a change in input–output behaviour.

Recuperation from surgery. As we have seen, network modifications will in general cause a small change in input–output behaviour, typically leading to a loss in prediction accuracy. In order to make up for this, all network modifications are given a small amount of recuperative training (several batches) before any comparison is made. The retraining amount was determined by trials and statistical analysis based on a dataset[1] which we do not otherwise use in our results, to avoid overfitting the search algorithm to a specific dataset.

3.2. The recommendation module

For the decision *when* to execute *which* network modification, we perform a two-step analysis.

Analytical criterion for model selection. As a first order criterion, we compute the amount of information carried by each neuron by looking at the layer's singular values. The number of

[1] MNIST, LeCun, Bottou, Bengio, and Haffner (1998).

neurons that may be removed from a layer depends on the count of singular values that are (close to) zero, or are several orders of magnitude smaller than the layer's largest singular value:

$$n_r = \{\sigma_i,\ i = 1 \dots k \mid \sigma_i < \epsilon_1 \sigma_1 \vee \sigma_i < \epsilon_2,\ \epsilon_1,\ \epsilon_2 \in \mathbb{R}^+\} \tag{13}$$

This number is compared to the layer's total neuron count n_l. Let h be the number of hidden layers in the network, and i be the index of the layer in which the modification was performed. Then each modification candidate is given a score between 0 and 1, calculated as follows:

- add neurons: $1 - n_r/n_l$
- add layer: $(1 - n_r/n_t) \cdot i/h$
- remove neurons: n_r/n_l
- remove layer: 1, if $n_r = n_l$, otherwise $(n_r/n_l) \cdot i/h$

In particular, the higher the layer number, the more likely a layer is added or removed. For further refinement in the future, we aim to replace this selection criterion with a more sophisticated formula, which would improve the optimization process beyond the results presented herein.

Statistical criterion for model selection. The second order decision basis is derived from the Bayesian information criterion (BIC), also known as Schwarz information criterion. It was derived by Schwarz (1978) to address the problem of selecting between (statistical) models of different dimension. It takes into account the number of parameters of the given model, the sample size of the input data, as well as the a-posteriori model error computed from a likelihood function of the model given its parameters and input data. Since methods from Bayesian statistics are applied, it is assumed that the underlying data are independent and identically distributed from a family of allowed distributions.

The BIC is given by

$$\mathrm{BIC} = \ln n \cdot k - 2 \cdot \ln L, \tag{14}$$

where k is the number of model parameters, n the sample size, and L the likelihood function.

In our application, sample size is constant throughout the whole process. L needs to be estimated or calculated a-posteriori after a network operation has been performed. Our modifications are intended to keep the change in input–output behaviour minimal, therefore the difference in L will be very small between any two modifications. Eq. (14) thus becomes

$$\mathrm{BIC} \approx c_1 \cdot k + c_2 + \Delta L, \tag{15}$$

where c_1 and c_2 are constant and ΔL is the (small) change in error depending on the performed operation. Since the number of parameters k may be well above 10^6 and $c_1 = \ln n \gg 1$ is dependent on the sample size, we neglect ΔL and the constants, and directly use k as a second order constraint when deciding which modifications to apply.

Thus, all potential network modifications are ranked by two parameters. The Surgeon has two modes: it can either pick the top n ranking operations per decision step (with n being a tunable hyperparameter), or select the highest ranking operation of each type. Our experiments are performed with the latter option.

3.3. "The Surgeon": A genetic algorithm for neural architecture search

The final module is the Surgeon, the genetic algorithm that searches for an optimized network architecture while training network weights at the same time. We use the term genetic algorithm to describe a searching meta heuristic inspired by biological processes such as evolution, mutation, and selection. In this first paper about our new ansatz, we limit our focus on perhaps the most ubiquitous type of architecture: sequential networks (i.e. without recurrent or skip connections) consisting only of fully connected layers. As Cai et al. (2018) have shown, however, a number of the above described tools can be generalized easily also to non-sequential networks, as well as convolutional layers.

The rough idea behind the Surgeon is to alternate training and network optimization phases. Several competing topologies, called *branches*, may be retained concurrently. During the optimization phase, modification candidates are created for each such branch. From these, the n best performing ones are kept and put through another training step. One such training and optimization cycle is depicted in Fig. 1.

Algorithm 1 The Surgeon

function PERFORMSURGERY(m)
  input: model m
  output: optimized network m_{opt}

  initial modification $m_0 \leftarrow$ initialized from model m
  train m_0 for initial number of epochs
  list of current branches $B \leftarrow$ initialized with modif. m_0
  while termination criteria not met **do** ▷ e.g. max. epochs
    list of potential modification candidates $M_{cand} \leftarrow \varnothing$
    list of chosen modification candidates $M_{sel} \leftarrow \varnothing$
    for each branch in current branches B **do**
      determine possible modification candidates
      add candidates to M_{cand}
    repeat
      $M_{sel} \leftarrow$ chosen from M_{cand} ▷ e.g. 7 competitors
      re-train M_{sel} for small no. of batches ▷ e.g. 15 batches
      compare M_{sel}
      keep n top scoring as new current branches
      update current branches B ▷ e.g. best 2 branches
      train B to full epoch step ▷ e.g. 10 epochs
    until improvement achieved or max re-tries reached
  select final best scoring branch m_{opt} from B
  return fully trained optimized network m_{opt}

The overall structure of the Surgeon can be seen in algorithm 1. First, the provided model is pre-trained for an initial number of epochs, then the list of current branches is initialized with it. The choice of how many branches at most are being kept concurrently is a hyperparameter setting, and has a great influence on the total computational cost of the Surgeon.

We then evolve and continuously update the list of concurrent branches until termination criteria, such as the maximal number of epochs (if limited by computational resources) or a minimum accuracy threshold (if the topology optimization is used without a computational resource bottleneck), are met. At each decision point, the recommendation module analyses all networks in the list of current branches, and ranks all potential modifications. From these, in line 15 of algorithm 1, we select the most promising candidates, and perform the selected operations using the modification module.

The generated candidates are re-trained for several batches, to make up for lost performance. Our main objective function is to maximize the accuracy achieved by any given neural network through continuous network surgery. It is therefore sensible to retain those network candidates in algorithm 1, line 19 that reach the highest validation accuracy score. Note that we are scoring on validation accuracy instead of accuracy to avoid overfitting.

Focusing on accuracy alone is a greedy approach and carries the risk of getting stuck in local optima. We overcome this by additionally rewarding modification candidates that show a greater

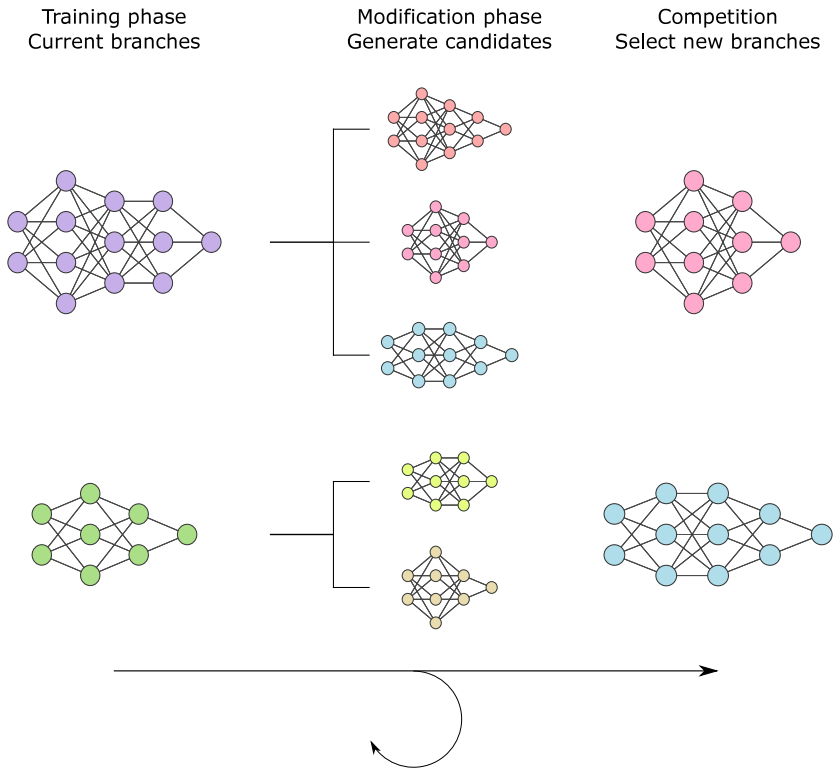

Fig. 1. Graphical representation of the Surgeon. Each cycle starts with a training phase, where all current branches are trained for a fixed number of epochs. Then the modification module generates new candidates for each branch based on the results provided by the analysis module. These candidates receive an appropriate amount of recuperative training, after which the winners are selected out of the set of all candidates, thus forming the new current branches. Network graphs created using LeNail (2019).

accuracy gain. However we need to make a distinction when rewarding this gain. We share Cai et al. (2018)'s rationale that an increase of 1% needs to be weighed higher in case it happens from 90 to 91% accuracy rather than 70 to 71%. At the same time, if an operation keeps the accuracy constant at e.g. 95% we can assume that a local optimum may have been reached. Therefore an operation that leads to an accuracy increase from 90% to 94%, showing potential for further improvement, should be regarded higher even though the reached accuracy is lower. Lastly we do not wish to neglect network size. A 1% accuracy increase might never be favourable at all, if the required increase in network size is "too big".

Note that it is hard to define when a network has indeed become "too big", a fact that is emphasized by the large number of publications dealing with pruning techniques (Blalock et al., 2020).

We need to find a way to balance these three components (accuracy, accuracy gain, network size), and to create a composite score by which we can rank the performance of current branches. As an additional restriction, the composite score should not depend on any global candidate statistics, nor do we want to set a global limit for network size. Therefore we cannot in any meaningful way regard the total number of parameters of a modification candidate, since we lack overall comparison. Instead, for each candidate, we store the network size as a fraction of the candidate's parent's network size. Thus, a network size fraction greater than 1 indicates growth, a fraction smaller than 1 indicates shrinking, and the identity operation yields a size fraction of exactly 1.

We want the scoring function as well as its first order derivative to be strictly increasing with accuracy gain, but decreasing with size fraction. This behaviour needs to hold even when accuracy gain becomes 0 or network size fraction becomes 1. The following scoring function fulfils all specified requirements:

$$s = a + \frac{\exp(\Delta a)}{\exp(\Delta f)} \tag{16}$$

where a is the current accuracy, Δa the current accuracy gain, and Δf the current network size fraction, and is adapted from Cai et al. (2018). A scoring example can be seen in Table 1.

Table 1
Scoring example. **A** denotes accuracy, **AG** accuracy gain, **PF** parameter fraction and **S** the calculated score. The layer number is given in reference to the hidden layers. Winning operations, that are kept as new concurrent branches, are indicated with an asterisk.

Operation	Position	A	AG	PF	S
Remove layer*	Layer 5	0.6023	0.0274	0.9977	0.9812
Identity*	–	0.6016	0.0268	1.0000	0.9795
Remove 1 Neuron	Layer 5	0.5829	0.0081	0.9997	0.9539
Add layer	Before layer 4	0.5773	0.0025	1.0030	0.9450
Add 20 Neurons	Layer 2	0.5678	−0.0070	2.6544	0.6377

As an option to alleviate greediness, the Surgeon can keep a one cycle memory. In this case, the newly selected concurrent branches are compared to previously best scoring ones, and retained only if their achieved accuracy is at least as good as the previous value. Should this not be the case, they are discarded and we backtrack one step. New potential candidates are provided by the recommendation module, and we train and compare these. Finally the best scoring branch is returned as the optimized network.

Table 2
Starting topologies for the Surgeon. Each topology additionally has a reshape layer as its input layer, as well as a dense layer without activation function as its output layer. The neuron count in the input and output layers is dependent on the dataset.

Name	Hidden layer count	Hidden layer sizes	Activation
Small	1	10	ReLu
Medium	3	10 - 10 - 10	ReLu
Large	3	300 - 100 - 100	ReLu

4. Experiments

4.1. Data

We evaluate the performance of the Surgeon on several standard benchmark datasets (SVHN, CIFAR-10, CIFAR-100, EuroSAT, EMNIST, Fashion-MNIST), which are described below. The SVHN dataset was downloaded manually from http://ufldl.stanford.edu/housenumbers/. The CIFAR-10 dataset was fetched from the `keras.datasets` catalogue. All others are fetched from the `tensorflow-datasets` catalogue, batched, and shuffled. As an additional preprocessing step, we normalize the data to be within the range [0, 1].

We pick three starting topologies that are described in Table 2, and perform several runs of the Surgeon on each one. Note that Table 2 omits the input layer, which is always a reshape layer that ensures the input data is formatted as a 1-dimensional vector instead of a multi-dimensional array, as well as the final dense layer. The Surgeon never adapts these two layers, as they are inherently determined by the dataset through the shape of the training data and number of output classes.

The small starting topology is consciously chosen to be insufficient for most of the regarded datasets, to mimic a case where a model is unknowingly trained with an inadequate network architecture. We average our results over several runs and several random seeds, and compare to results achieved by simply training the starting topology for the same number of epochs. Detailed machine properties and hyperparameter settings are listed in Appendix.

The SVHN dataset. The (Google) Street View House Number (SVHN) dataset was published by Netzer et al. (2011). It contains 73,257 training, as well as 26,032 validation colour images. We use the cropped version, where images are of size 32×32 pixels, and fall into 10 classes according to the numbers 0–9. Additionally, 531,131 extra images of lower difficulty are available but not currently used.

The CIFAR-10 dataset. The Canadian Institute For Advanced Research (CIFAR)-10 dataset was published by Krizhevsky (2009). It contains 60,000 32×32 pixel colour images that are evenly divided into 10 classes. 10,000 of the images are set aside for validation purposes.

The CIFAR-100 dataset. The CIFAR-100 dataset is equivalent to the CIFAR-10 dataset, except that samples are evenly divided into 100 classes.

The EuroSAT dataset. The EuroSAT dataset was published by Helber, Bischke, Dengel, and Borth (2018, 2019). It is based on Sentinel-2 satellite images consisting of 27000 labelled and geo-referenced colour images of size 64×64 pixels, that belong to 10 classes. We make use of the RGB version that contains only the optical red, green and blue frequency bands.

The EMNIST dataset. The EMNIST dataset was published by Cohen, Afshar, Tapson, and van Schaik (2017). It contains greyscale handwritten character digits of size 28×28 pixels that are derived from the NIST special database. They depict the numbers from 0–9 and thus fall into 10 classes. 697,932 training as well as 116,323 validation samples are provided.

Table 3
Statistics over all baseline and Surgeon runs. We report means and standard deviations.

Topology	Small	Medium	Large
SVHN			
Baseline accuracy	0.20 ± 0.00	0.34 ± 0.12	0.78 ± 0.01
Surgeon accuracy	0.29 ± 0.16	0.58 ± 0.04	0.79 ± 0.01
Rel. accuracy incr. [%]	45.00	70.59	1.28
Param. fraction incr. [%]	16.48 ± 52.0	1.97 ± 7.13	−7.23 ± 32.48
CIFAR-10			
Baseline accuracy	0.28 ± 0.01	0.32 ± 0.01	0.49 ± 0.00
Surgeon accuracy	0.34 ± 0.01	0.38 ± 0.06	0.51 ± 0.00
Rel. accuracy incr. [%]	21.43	18.75	4.08
Param. fraction incr. [%]	1.12 ± 2.85	203.87 ± 417.45	2.12 ± 3.13
CIFAR-100			
Baseline accuracy	0.12 ± 0.01	0.12 ± 0.01	0.20 ± 0.00
Surgeon accuracy	0.19 ± 0.00	0.14 ± 0.01	0.20 ± 0.00
Rel. accuracy incr. [%]	58.29	17.74	2.89
Param. fraction incr. [%]	865.44 ± 0.00	−0.18 ± 11.94	7.40 ± 27.69
EuroSAT			
Baseline accuracy	0.11 ± 0.00	0.24 ± 0.10	0.60 ± 0.02
Surgeon accuracy	0.54 ± 0.05	0.57 ± 0.03	0.66 ± 0.01
Rel. accuracy incr. [%]	383.09	215.00	10.37
Param. fraction incr. [%]	−14.31 ± 10.68	−12.87 ± 13.83	3.24 ± 4.15
EMNIST			
Baseline accuracy	0.67 ± 0.00	0.63 ± 0.01	0.83 ± 0.00
Surgeon accuracy	0.73 ± 0.00	0.71 ± 0.01	0.85 ± 0.00
Rel. accuracy incr. [%]	8.13	12.10	1.54
Param. fraction incr. [%]	470.44 ± 0.00	16.98 ± 47.30	13.02 ± 9.78
Fashion-MNIST			
Baseline accuracy	0.85 ± 0.01	0.85 ± 0.01	0.89 ± 0.00
Surgeon accuracy	0.85 ± 0.00	0.85 ± 0.00	0.89 ± 0.00
Rel. accuracy incr. [%]	0.47	0.65	0.06
Param. fraction incr. [%]	−0.36 ± 10.44	−3.83 ± 11.64	24.94 ± 86.73

The Fashion-MNIST dataset. The Fashion-MNIST dataset was published by Xiao, Rasul, and Vollgraf (2017). It contains greyscale fashion image data taken from Zalando's[2] article catalogue that fall into 10 categories. The dataset consists of 60,000 training as well as 10,000 validation samples of size 28×28 pixels.

4.2. Results

We apply the Surgeon to each combination of the above datasets and starting topologies (cf. Table 2). We do so a total of 15 times, re-initializing the `numpy` and `tensorflow` modules with a new random seed after every 3 runs, and report average statistics (cf. Table 3). As a baseline, we train the starting topology for the same number of epochs with each random seed. Note that throughout the entire section, unless otherwise stated, we report achieved validation accuracies rather than training accuracies.

To avoid overfitting on any specific dataset, we fix some hyperparameters for training (such as batch size, optimizer, and learning rate) before starting any runs with the Surgeon (cf. Appendix). No additional fine tuning is performed on any model.

Overall Surgeon performance. Table 3 and Fig. 2 both show an overview of average Surgeon performance for all starting topologies and datasets. We can see that in all cases, the Surgeon reaches or outperforms the result of the baseline with regard to validation accuracy. We are able to observe two types of behaviour. In cases where the baseline accuracy is already high to begin with, the relative accuracy increase achieved by the Surgeon is comparatively low. However the Surgeon is often able

[2] A fashion company specialized in online commerce.

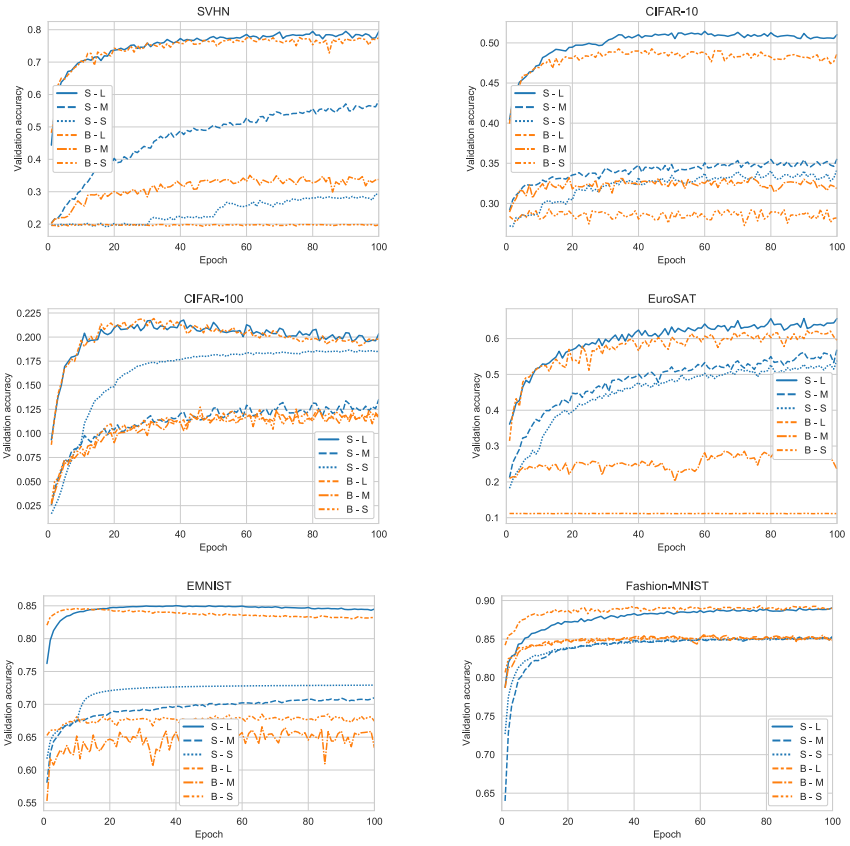

Fig. 2. Average performances of the Surgeon (S) compared to the baseline (B) for all three starting topologies (L — large, M — medium, S — small), depicted per dataset.

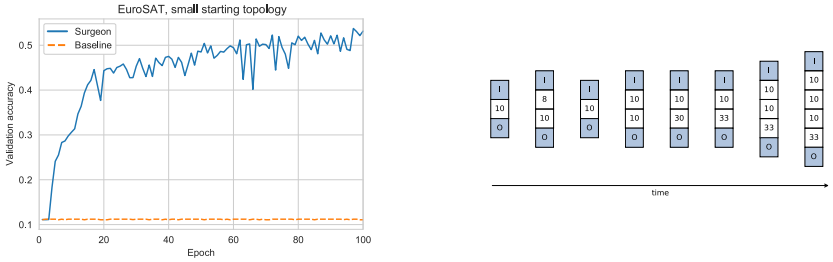

Fig. 3. Left: Single run of the Surgeon on the EuroSAT dataset and small starting topology. Right: Evolution of the applied topology during the run. Boxes with marked the letters I and O designate the input and output layers, the numbers in the remaining boxes indicate the number of neurons in the respective hidden layers.

to decrease the required amount of network parameters without loss in accuracy. On the other hand we are able to observe quite significant improvements in accuracy in cases where the starting topology is sub optimal. In fact, as can be seen in Fig. 2, there are cases where the baseline is not learning at all whereas the Surgeon is able to overcome the initial local minimum.

We will subsequently highlight a few interesting cases.

Topology rescue. We recall that the small starting topology, consisting of only one hidden layer with 10 neurons, was purposefully chosen to be insufficient for convergence. In fact, as we can see in Fig. 2, the small topology baseline learns for neither SVHN, CIFAR-10, nor EuroSAT. The Surgeon on average manages to improve the topology and learn at least a little, even reaching a validation accuracy above 50% in case of the EuroSAT dataset. In case of the SVHN dataset, the global average over all runs contains several instances where even with the aid of the Surgeon, the model is not able to learn at all and stays stuck in the initial state, as well as a number of runs where the Surgeon is able to very quickly leave this local sink and then in fact provides a model that learns very well.

In Fig. 3 (left), we can see a single run of the Surgeon using the EuroSAT dataset and small starting topology, where an early *Add Layer* operation allows the network to train properly. The total parameter increase in this case is less than 1%, with the Surgeon preferring to add several small layers rather than widening existing layers. Note that due to the shape of our training data and choice of starting topology, a large portion of the network's parameters is required to connect the input layer to the first hidden layer. Adding a whole layer after the first 10-unit layer causes only a small overall increase, whereas widening the first

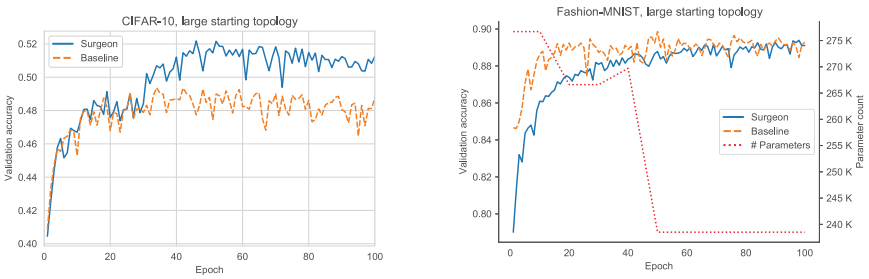

Fig. 4. Left: Single run of the CIFAR-10 dataset and large starting topology. The Surgeon manages to overcome the local optimum at around epoch 30. Right: Single run of the Surgeon on the Fashion-MNIST dataset and large starting topology. The Surgeon is able to reduce the total parameter count by around 14% while still reaching the same overall validation accuracy.

hidden layer might cause a greater increment in parameter count. Fig. 3 (right) shows the evolution of the topology over the course of the run.

Accuracy increase. The Surgeon is able to detect and improve sub-optimally sized network architectures. This works in cases such as above (cf. Fig. 3 left), where it is very obvious that little to no learning happens, but also in less apparent ones, where the base topology does learn to a certain extent. In Fig. 4 (left) we can see a single run of the Surgeon trained on CIFAR-10 where both the large starting topology as well as the Surgeon start learning in a similar fashion and soon reach a plateau. After a while however, the Surgeon is able to perform an *Add Layer* operation that allows the topology to overcome the local optimum which had been reached.

Parameter reduction. As mentioned previously, using the large topology as a starting point for the Surgeon does not improve achieved accuracy by any large margin (cf. Table 3). In Fig. 4 (right) we can see a single run of the Surgeon on the Fashion-MNIST dataset using the large starting topology. The baseline in this case is already performing quite well given that we are regarding a very basic network architecture. In this case we can observe pruning by the Surgeon, such that an early *Remove Layer* operation allows parameter reduction of almost 15%.

For the large starting topology, the composite scoring function given in Eq. (16) prevents any big jumps in accuracy since they would most likely come at the cost of a (potentially rather large) increase in network size. The scoring function much rather prefers network operations that keep the accuracy more or less constant while reducing network size.

Computational costs. In our trials the Surgeon is configured to produce a resulting topology that has been trained for exactly 100 epochs, with decision points every 10 epochs. We allow for a maximum of two concurrent branches, as well as re-drawing from potential branches up to two times, cf. Appendix. On average, 0.56 re-draws are necessary per decision step, resulting in an average total training amount of around 290 epochs per run of the Surgeon. The additional overhead produced by the analysis and modification modules mostly relies on matrix calculations and manipulations, which are highly optimized by standard python modules such as `numpy`, and thus cause only negligible additional computation time. In particular, making use of large GPU clusters is not a necessary requirement for running the Surgeon. For verification purposes and in order to ensure scalability of our code we ran our experiments both on a standard office computer without making use of any GPU acceleration, as well as a GPU cluster (cf. Appendix for detailed hardware specifications).

We can see from Fig. 2 that 100 epochs in most cases seem to be longer than is necessary for the Surgeon to reach convergence. With appropriate early stopping techniques, and/or a more dynamic training schedule, the total training amount for the Surgeon could be considerably reduced, and the resulting model (including its trained weights) used either as is, or further fine tuned (cf. Fig. 5).

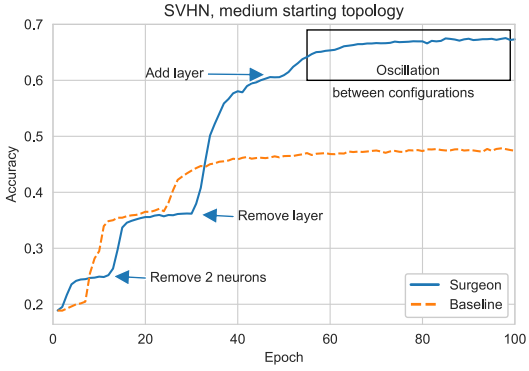

Fig. 5. Example of a topology evolution performed by the Surgeon on the SVHN dataset and medium starting topology. Early on, reducing the network size helps to increase the accuracy level (at epochs 10 and 30). Adding a whole layer in epoch 50 is still able to achieve some increase in accuracy. From epoch 60 on the network oscillates between very similar states. Ideally this behaviour can be used as an indicator for early stopping in future versions of the Surgeon.

5. Discussion and outlook

In this paper, we presented the Surgeon, a ANN/evolutionary algorithm hybrid optimization designed for neural architecture search. The algorithm utilizes a modification module, that is able to perform minimally invasive network surgeries, where the topology of the network is modified with as little change to overall input–output-behaviour as possible. Additionally, it uses a recommendation module, that analyses a given neural network and indicates which structural changes may be most beneficial. Those changes can be either increases of network width or depth in case of high information density, or respective decreases, in case network size can be reduced without fear of too high an accuracy loss. Both modules do not utilize any black box behaviour but are based on mathematical tools, which we presented in Section 3.

We put the Surgeon to the test on several combinations of starting topologies and datasets. We saw that the network generated by the Surgeon is able to outperform the baseline in case of suboptimal topologies, or reach comparable accuracies while pruning the underlying network structure to less resource-intensive topologies.

A very important feature of the Surgeon is that it itself is computationally cheap, with little overhead compared to simple baseline training. Via hyperparametre settings, it is possible to make use of larger computational power if required/available.

5.1. Limitations of this work and future goals

For this proof of concept work, we limited ourselves to the most basic and ubiquitous network structures — dense layers linked strictly in sequence. While such neural networks are best studied and easiest to understand and manipulate from a mathematical perspective, there are some draw backs.

Fully connected neural networks require a fixed input shape, thus they are not suited for a variety of classical deep learning tasks such as natural language processing (NLP) or image detection. Large tabular datasets often contain ordinal or categorical variables which are not well suited for deep learning tasks as gradient calculation becomes somewhat ill-defined. In particular, we are mostly limited to making use of image classification benchmark sets. For these, state of the art is driven by large, computationally expensive algorithms that allow more complex topological elements such as convolutional or recurrent layers, skip connections, etc., see for example Liu et al. (2019), Xu et al. (2020) and Zoph and Le (2017). In general, few benchmark results exist for fully connected neural networks trained on these datasets.

For future work we wish to extend and improve both the Surgeon as well as the underlying modules. Currently, the Surgeon follows a static routine, with hyperparameters such as epochs steps, number of selected candidates, or number of concurrent branches staying constant throughout the whole process. This could be changed to a more dynamic approach, and could maybe integrate further features such as adaptive learning rates, or a more sophisticated memory, as well as early stopping mechanisms to improve performance gains compared to baseline training even further. The recommendation module can be improved by finding a closer approximation for the BIC. For larger starting topologies, the scoring function given in Eq. (16) seems to be chosen too restrictively. The balancing between the different components can be adapted to allow for more drastic additions even when the network is already quite large to begin with.

Lastly we want to expand the modification module to allow more complex topologies as well, and include an option to cross architecture types. This would allow us to include e.g. convolutional or recurrent elements, change a network from one type to another, or even freely mix and match as required.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Appendix. Hyperparameter settings and machine specifications

Simulations for the SVHN and CIFAR-10 datasets were performed on a Windows 10 machine with an Intel(R) Core(TM) i7-9700K CPU 3.6 GHz processor and 64,0 GB RAM. The code was implemented in python 3.7.7 using tensorflow 2.1.0.

Simulations for the CIFAR-100, EuroSAT, EMNIST, and Fashion-MNIST datasets were performed on a virtual machine running on a 24-core 2.1 GHz Intel Xeon Scalable Platinum 8160 processor, which is equipped with a Tesla V100 GPU card with 16 GB memory. The code for these datasets was ported to python 3.8.2 using tensorflow 2.3.0 and tensorflow-datasets 4.1.0.

We chose the following hyperparameters:

The modification module.

- Amount of re-training per modification type:
  - Identity: 1 · re-training batches
  - Add Neuron: 1 · re-training batches
  - Add Layer: 10 · re-training batches
  - Remove Neuron: 10 · re-training batches
  - Remove Layer: 10 · re-training batches
  - Truncated SVD: 1 · re-training batches

The recommendation module.

- Absolute singular value threshold: $\epsilon_2 = 0.3$
- Relative singular value threshold: $\epsilon_1 = 0.005$
- Recommendation style: best scoring per modification type
- Include additional random draw: True

The Surgeon.

- Training time for winning branch: 100 Epochs
- Initial pre-training: 10 Epochs
- Interval between decision points: 10 Epochs
- Concurrent branches: 2
- Maximum re-draws per decision step: 2
- Number of batches for re-training: 25 batches

The testing setup.

- Random seeds: 6, 63, 72, 77, 97
- Loss function: Sparse categorical crossentropy
- Optimizer: SGD
  - learning rate: 0.005
- Metrics: accuracy
- Batch size: 16

References

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., & Guttag, J. (2020). What is the state of neural network pruning? In I. Dhillon and D. Papailiopoulos and V. Sze (Eds.), *Proceedings of machine learning and systems, Vol. 2* (pp. 129–146).

Cai, H., Chen, T., Zhang, W., Yu, Y., & Wang, J. (2018). Efficient architecture search by network transformation. In *32nd AAAI conference on artificial intelligence, AAAI 2018* (pp. 2787–2794). AAAI Press, arXiv:1707.04873.

Chen, T., Goodfellow, I. J., & Shlens, J. (2016). Net2Net: Accelerating learning via knowledge transfer. In Y. Bengio, & Y. LeCun (Eds.), *4th international conference on learning representations, (ICLR)*.

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. arXiv:1702.05373.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., & Fergus, R. (2014). Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 1269–1277). Curran Associates, Inc..

DiMattina, C., & Zhang, K. (2010). How to modify a neural network gradually without changing its input-output functionality. *Neural Computation*, *22*(1), 1–47. http://dx.doi.org/10.1162/neco.2009.05-08-781.

Dong, X., & Yang, Y. (2019). Searching for a robust neural architecture in four GPU hours. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, *20*(55), 1–21.

Frankle, J., & Carbin, M. (2019). The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th international conference on learning representations, (ICLR)*.

Girshick, R. (2015). Fast r-CNN. In *2015 IEEE international conference on computer vision (ICCV)* (pp. 1440–1448). IEEE Computer Society, http://dx.doi.org/10.1109/ICCV.2015.169.

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2018). Introducing eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. In *IGARSS 2018-2018 IEEE international geoscience and remote sensing symposium* (pp. 204–207). IEEE.

Helber, P., Bischke, B., Dengel, A., & Borth, D. (2019). Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*.

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*. http://dx.doi.org/10.1109/CVPR.2017.243.

İrsoy, O., & Alpaydın, E. (2020). Continuously constructive deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(4), 1124–1133. http://dx.doi.org/10.1109/TNNLS.2019.2918225.

Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., & Xing, E. P. (2018). Neural architecture search with Bayesian optimisation and optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems 31* (pp. 2016–2025). Curran Associates, Inc..

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images*: *Tech. rep.*, University of Toronto.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

LeNail, A. (2019). NN-SVG: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, *4*(33), 747. http://dx.doi.org/10.21105/joss.00747.

Li, L., Khodak, M., Balcan, N., & Talwalkar, A. (2021). Geometry-aware gradient algorithms for neural architecture search. In *9th international conference on learning representations (ICLR)*.

Li, L., & Talwalkar, A. (2019). Random search and reproducibility for neural architecture search. In *Conference on uncertainty in artificial intelligence (UAI)*.

Liu, H., Simonyan, K., & Yang, Y. (2019). DARTS: Differentiable architecture search. In *7th international conference on learning representations (ICLR)*.

Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *Proceedings of the third international conference on genetic algorithms* (pp. 379–384). Morgan Kaufmann Publishers Inc..

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464. http://dx.doi.org/10.1214/aos/1176344136.

Wang, R., Cheng, M., Chen, X., Tang, X., & Hsieh, C.-J. (2021). Rethinking architecture selection in differentiable NAS. In *9th international conference on learning representations (ICLR)*.

Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv:cs.LG/1708.07747.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., et al. (2020). PC-DARTS: Partial channel connections for memory-efficient architecture search. In *8th international conference on learning representations (ICLR)*.

Xue, J., Li, J., & Gong, Y. (2013). Restructuring of deep neural network acoustic models with singular value decomposition. In *Proceedings of the annual conference of the international speech communication association (INTERSPEECH)* (pp. 2365–2369).

Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *5th international conference on learning representations (ICLR)*.