Exploring the Search Space of Neural Network Combinations obtained with Efficient Model Stitching

Guijt, Arthur; Thierens, Dirk; Alderliesten, Tanja; Bosman, Peter A.N.

DOI
10.1145/3638530.3664131

Publication date
2024

Document Version
Final published version

Published in
GECCO '24 Companion

Exploring the Search Space of Neural Network Combinations obtained with Efficient Model Stitching

Arthur Guijt
Arthur.Guijt@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands

Dirk Thierens
D.Thierens@uu.nl
Utrecht University
Utrecht, The Netherlands

Tanja Alderliesten
T.Alderliesten@lumc.nl
Leiden University Medical Center
Leiden, The Netherlands

Peter A.N. Bosman
Peter.Bosman@cwi.nl
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
Delft University of Technology
Delft, The Netherlands

ABSTRACT

Machine learning models can be made more performant and their predictions more consistent by creating an ensemble. Each neural network in an ensemble commonly performs its own feature extraction. These features are often highly similar, leading to potentially many redundant calculations. Unifying these calculations (i.e., reusing some of them) would be desirable to reduce computational cost. However, splicing two trained networks is non-trivial because architectures and feature representations typically differ, leading to a performance breakdown. To overcome this issue, we propose to employ stitching, which introduces new layers at crossover points. Essentially, a new network consisting of the two basis networks is constructed. In this network, new links between the two basis networks are created through the introduction and training of stitches. New networks can then be created by choosing which stitching layers to (not) use, thereby selecting a subnetwork. Akin to a supernetwork, assessing the performance of a selected subnetwork is efficient, as only their evaluation on data is required. We experimentally show that our proposed approach enables finding networks that represent novel trade-offs between performance and computational cost compared to classical ensembles, with some new networks even dominating the original networks.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; • **Computing methodologies → Neural networks**.

KEYWORDS

Neuroevolution, Neural Architecture Search, Stitching, Ensembles

ACM Reference Format:
Arthur Guijt, Dirk Thierens, Tanja Alderliesten, and Peter A.N. Bosman. 2024. Exploring the Search Space of Neural Network Combinations obtained with Efficient Model Stitching. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3638530.3664131

1 INTRODUCTION

In recent years, deep neural networks have shown to be very powerful machine learning models in many fields. A prime example is in the field of computer vision, where training such a network from raw data has superseded manual feature engineering [47]. More often than not, these networks have been found to perform at their best when made as large as possible [47] and combined into an ensemble [8, 43].

Obtaining the necessary deep neural networks for an ensemble can be done in a number of ways. One can use a training procedure which produces a variety of networks using a reduced number of training runs, that can then be turned into an ensemble. Examples of such approaches are MotherNets [43], Snapshot Ensembles [19] and Batch Ensembles [44]. An alternative, highly effective method is to use multiple already-trained networks and employing transfer learning [7, 30, 32]. By doing so, multiple specialized networks are constructed which re-use knowledge gained from training on external datasets.

An ensemble constructed using these networks can be costly to use, and will incur this cost every time it is used. Expending some additional computational resources up front to reduce the costs in the long tail of using this network can make sense. For this, one can use a network pruning technique to lower the computational costs of each network individually [26] or even reduce a large variety of hyperparameters [28]. However, this reduction in cost does come with a trade-off. The computational cost can only be reduced so much before network performance starts to degrade [28].

Beyond the redundancy and complexity of a single network, there still exists significant redundancy across the networks in an ensemble. Given similar tasks networks likely develop similar features, repeating the same feature extraction performed by the other networks in the ensemble. For example, take networks trained on a computer vision task, using a dataset such as ImageNet [35]. The features in the first layers of a network for such a task represent aspects such as edges, whereas later layers represent higher level

GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0495-6/24/07.
https://doi.org/10.1145/3638530.3664131

features, such as eyes [31]. Unifying the calculations for these features between the networks in an ensemble could further reduce the computational costs associated with *using* this ensemble.

In this work we propose a new approach in order to find other, potentially better, trade-offs, by splicing two basis networks. For example, through the aforementioned redundancy reduction, and where possible through the use of a cheaper path to these features.

However, splicing two neural networks is a difficult problem due to the compatibility issues that appear between different networks, as will be discussed in Section 2. These issues are only compounded if the networks to be spliced exhibit different architectures. In this work, we perform splicing in a way that overcomes these issues.

2 SPLICING NEURAL NETWORKS

Splicing two neural networks without breaking them is a nontrivial task [3, 38, 42]. Due to representational differences, simple splicing is ineffective, and will commonly result in broken networks. These differences can be separated into two categories. First, there is the problem that different parts of the network may encode different features. For example, in the case of networks trained on ImageNet, features in the first layers of a network are generally lower level, representing aspects such as edges and textures, whereas features represented in later layers include higher level features - such as eyes or other complex objects [31]. To perform crossover without disruption, one will need to know *where* matching features are.

Second, even if similar features are present in a part of the network, they may be represented differently. This is caused by structural-functional redundancies induced by the networks' structure, also referred to as the competing conventions problem [36, 38, 41]. This problem concerns, for example, the ordering of features in a hidden layer of a Multi-Layer Perceptron (MLP) or classical neural network. Here, the weights of the corresponding and next layer can be permuted to reorder the features without affecting the end result. This causes a permutation symmetry among the weights. During initialization and training symmetries, such as this ordering property, will be determined at random, causing different networks to almost certainly adhere to different feature representations.

Contrary to MLPs, which consist of sequences of linear layers and their activation functions, modern deep neural networks consist of a large variety of layers, potentially with parallel branches. It is no longer a given that the permutation symmetry holds for each kind of layer used in practice. By extension, different layers may also introduce new kinds of symmetries.

Therefore, for the approach to be generally applicable, we need to have the means to splice a large variety of deep neural networks, including those with different architectures and those with parallel branches. Splicing two such deep neural networks requires both (1) knowledge about matching layers and (2) the means to align their representation. In literature, (1) is either bypassed by assuming identical architectures and assuming positional equivalence, or by tracking transformations as a network is grown [38]. Such approaches are not helpful in the context of already trained networks with varying architectures. Common strategies to tackle (2) are to permute the neurons, so that their activations match up [3, 42], or by selecting and matching individual features and merging layers [39]. Yet, these approaches are restricted to correcting for specific symmetries, and do not account for other contextual symmetries. An example of such a symmetry is the use of the sigmoid activation function [41], where due to the symmetry of the activation function, the sign of the feature may be flipped.

In this work, we will investigate the use of model stitching [5] for network splicing, as this approach allows us to operate agnostic of the layers used in the network itself. Model Stitching was originally intended to assess feature map similarity, much like canonical correlation analysis [27, 33] and centered kernel analysis [21, 46]. Model stitching essentially connects the output of one layer to the input of another layer in another network. Yet, model stitching has one major difference with naively making these connections directly: a simple trainable layer is inserted in between the layers to be connected. The parameters of this intermediate *stitching layer* are then determined by training the network with all other parameters frozen. This newly introduced stitching layer serves as a translator and corrects for the various misalignments and symmetries present in the representation of the layers to be connected. The chosen kind of stitching layer is important: a complex layer or even a small subnetwork, that can perform feature extraction, while powerful, yields more expensive networks and is costly to train. Yet, a layer that is too simple cannot correct for all desired symmetries, e.g., an affine transformation cannot reorder features. As such, in this work, we will focus on linear(-like) layers, such as convolutional layers with a 1x1 kernel, without an activation function.

While stitching can be an effective method to splice two sequential networks, the original approach trains and tests each stitch individually. This is problematic due to the presence of parallel branches, e.g., those in a ResNet [18]. Replacing a feature map via a stitch will leave any parallel branches untouched, causing a large portion of the other network to remain in use. In turn, more than one stitch may be necessary to save significant computational resources. In general, limiting to one splice is too restrictive to find interesting networks. To search through such a space, creating and training these stitches as necessary is wasteful, as redundant work is performed whenever a stitch at the same position is used again. A more efficient approach will be described in the following section.

3 EFFICIENT MODEL STITCHING

To improve efficiency, we take inspiration from supernetworks. Previously, supernetworks have been used to efficiently search for network architectures. Supernetworks avoid training a network from scratch for every configuration to be evaluated by creating a very large compound network. This network contains each possible architectural choice, such that each layer is already trained [6]. This allows a specific architecture to be evaluated without training.

A similar approach is utilized here for combining two provided basis networks. The following paragraphs explain how we construct this network. Unlike the supernetworks described in [6], we do not need to train the entire network. The weights of the two basis networks are inherited and frozen, and only the stitching layers need to be trained. Furthermore, unlike the original stitching approach, these stitches are trained concurrently.

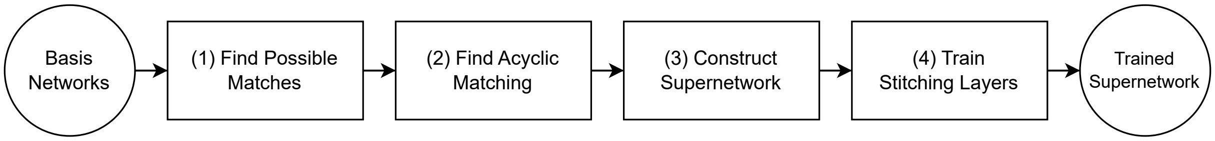
Basis Networks
(1) Find Possible Matches
(2) Find Acyclic Matching
(3) Construct Supernetwork
(4) Train Stitching Layers
Trained Supernetwork

Figure 1: An overview of the process to generate a single supernetwork based on two basis networks.

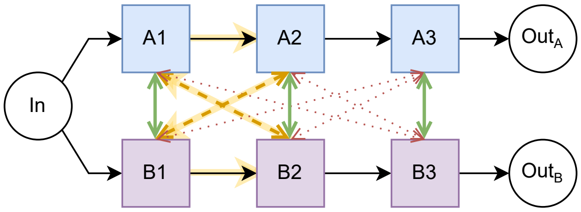

Figure 2: Not all pairwise matches between layers can be considered. Some transformations may be impossible given the kinds of stitching layers provided, e.g., the dotted red arrows. Furthermore, combinations of matches may introduce cycles after introducing the stitch and switch layers. E.g., the orange dashed pairs result in the highlighted cycle. Green edges represent a valid matching.

The procedure starts with a pair of basis networks. The networks used within this work are listed in Section 6. For a schematic representation of the steps, refer to Figure 1.

Finding possible matches. Before we can start splicing the networks using stitching layers, we need to consider what kinds of transformations we can perform; there needs to be a layer able to take both the right kind of input and produce the right kind of output. In this work, we use two kinds of stitching layers, which translate between the output of given a pair of layers A and B (and vice-versa). Both kinds require that the output of both of these layers are tensors of some shape sh_A and sh_B, which provide the size of each dimension.

First, we consider stitching using a linear layer if the dimensionality of the tensor $\dim(A) = \dim(B) = 2$. We label the dimensions of the tensor, $sh_A = [b, n]$, $sh_B = [b, m]$, where n and m represent the number of features per sample produced by this layer, and b the batch dimension. The layer is then configured to be a linear layer taking n features and producing m new features. Second, a stitching with a Conv2D layer is considered if $\dim(A) = \dim(B) = 4$, $sh_A = [b, n, w, h]$, and $sh_B = [b, m, w, h]$, where w and h represent the *identical width and height* of the last two dimensions of both tensors. Here we create a stitching layer that takes n channels and produces an output with m channels. In the remainder of this process we *can* only match two layers A and B if one of the layers above can convert between the output of layers A and B.

Finding an acyclic matching. However, we usually cannot consider all possible stitches simultaneously: this would result in a cycle in the constructed network. The orange dashed edges in Figure 2 show an example of a matching that would cause the highlighted cycle. As the computation to calculate a feature map in a cycle is reliant on itself, no forward pass can be performed. This renders a network containing a cycle unusable. Therefore, only matchings that does not generate cycles in the resulting graph are allowed. To obtain such a matching, we solve a matching problem with ordering constraints using a branch and bound approach as described in the supplementary material, aiming to maximize the number of matches between the two networks. The resulting acyclic matching under these constraints indicates where similar features are based on the structure of the network.

Constructing supernetwork. Given a set of matchings, i.e., pairs of layers A and B (and vice-versa) for which a kind of stitching layer is defined, we can merge and transform the networks into a supernetwork. This transformation introduces stitching layers to transform the output of layer B into what the output of layer A is expected to be, and a switch s_A which allows for the selection of the output. This switch represents a decision variable. The original input and the result of the stitch are provided as input to the switch, such that the switch can select either the original output of A or the output of the aforementioned stitch. All original connections to the output of A are replaced with the output of this switch s_A. By convention, we ensure that the first input to any switch represents the original output of the corresponding layer. Finally, the output is a special case. As we assume both networks to perform an identical task, and it is given that the format of the output is defined by the task itself, no stitches are necessary for the output switch. In addition to picking one of the outputs of the two networks, we include the option of creating an ensemble using these outputs as a third input to the output switch, as part of our focus on simplifying an ensemble.

As an example of the full procedure, using the (bidirectional) pairs given by the green bidirectional arrows in Figure 2, one would obtain a network similar to the one illustrated in Figure 3.

Training stitching layers. Finally, on the resulting network we perform a training procedure. This procedure is applied *once* as part of network creation and trains the weights of *all stitching layers* simultaneously. The resulting weights are used by all evaluations such that no training need to take place when evaluating a subnetwork.

First, all layers except for the stitching layers are frozen. We then train by adding together the Mean-Squared Error (MSE) loss at each switch, which is calculated between the output provided by the stitch, and the output of the original layer as target. As all layers of the original networks are frozen, only the stitches require a gradient. The output of the stitch is only used to compute the MSE loss, and does not affect the result of the network, including any other stitching layers. This allows us to train all stitching layers using the same forward and backward pass with little additional computational cost. This especially reduces computational costs compared to the original approach described in [5], where each stitching layer has its own *individual* task-specific (supervised) training procedure. Moreover, the newly proposed approach successfully trains stitches for which the original approach fails.

4 STITCHING FOR ENSEMBLE SIMPLIFICATION

From the process in Section 3 we obtain a supernetwork consisting of the layers of both basis networks, as well as the newly introduced stitch and switch layers that create new connections between the basis networks. An example of such a network is visible in Figure 3. The components for which we need to make a decision, are the switches. A switch picks the source of its output in the supernetwork. By deciding for every switch which input to take ('original' or 'stitched') we can select a subnetwork.

This is done by working backwards from the output, simplifying each switch to the selected input connection. Layers that are determined to no longer affect the output are then removed. This results in a subnetwork that can be evaluated without the requirement of additional training. For instance, by assessing the performance of a neural network on a validation set and/or by determining the number of computational operations required (e.g., multiply-adds).

We combine the decision for each switch into a fixed-size discrete representation. This representation is chosen as to relate to the structure of the underlying network. In particular, switches belonging to the same matching are grouped together; with the groups ordered according to a topological ordering of the network. This ordering can inform evolutionary algorithms, which will be introduced in Section 5, on what variables are closely related. For example, if both stitches in a group are used simultaneously there is a cost increase. This is because both inputs remain in use, while we introduce the additional cost of a stitch. To save computational resources only one of the two stitches should be used. This is an example of a form of correlation between two variables induced by the objective function, also referred to as linkage. Accounting for this linkage allows for the creation of offspring that are more likely to maintain good properties of their parents, increasing the likelihood of finding improvements.

The resulting search space allows us to search among combinations of the two networks, to merge computations, and to choose paths that trade-off computational efficiency and predictive performance in different ways.

While the sequential nature of a neural network generally requires us to re-evaluate a network from scratch every time, not all changes to the decision variables affect the chosen subnetwork. This is because some switches are inactive: their output is unused. For such a switch changing which input is used does not result in any change in the subnetwork that has been selected. By tracking which switches were active during a previous evaluation, and allowing this information to transfer to the offspring during variation, we can detect whether all changes are restricted to variables belonging to inactive switches. When all changes have occurred to inactive variables, the network has not changed - and the original objective value is still valid, allowing us to skip the evaluation step.

5 OPTIMIZING OVER SPLICING POINTS

With the aforementioned technique, we have constructed a search space which uses a fixed-length discrete representation. To investigate what types of networks can be found within this space, we employ Evolutionary Algorithms (EAs).

This provides us with knowledge about what kind of networks can be found, and potentially provides insight on what characteristics of search algorithms lead to searching more effectively in this space. Among networks that can be found, we are most interested in a subset of them, those for which one or more objectives of interest are optimized. Here, we have two objectives of interest. First, the number of computational operations, measured in the number of multiply-adds. Second, the performance of the network, measured using accuracy on a validation set.

We have empirically found that using many stitches simultaneously can be problematic. This is caused by the error introduced by each stitching layer, and this error accumulating over subsequent stitching layers. Eventually, this error can accumulate to the point that the network becomes nonfunctional.

To avoid starting with a pool of non-functional initial networks due to this issue, we initialize such that the resulting networks use fewer stitches. Let $1 - p$ be the probability to preserve original network links (and sample a 0), and p the probability of using the stitch at each switch. We performed a preliminary experiment using the first network pair, ImageNet (a), testing the values $p \in \{0.02, 0.10, 0.50\}$ to cover varying scales, we determined that $p = 0.02$ worked better than 0.10 or 0.50. As we hypothesize that the number of stitches used has a strong impact on the performance of a network, independent of string length, we pick p such that the number of 1's in the genotype is expected to be the same. i.e., as for ImageNet (a) the string length is $\ell = 309$, the number of expected 1's is $0.02 \times 309 = 6.18$. To keep this number identical across network pairs, we use $p = 6.18/\ell$, where ℓ is the string length.

While the cost of evaluating a network has been reduced greatly, it is still in the order of seconds: a network must still be applied to each element in the dataset to calculate the validation performance. To speed up the search, we can evaluate the performance of each subnetwork in parallel. Furthermore, to avoid idle resources the end of a generation, we opt to use an asynchronous variant [37]. In short, we use *asynchronous parallel evolutionary algorithms*, such that we can maximally utilize the available computational resources. All the EAs are made asynchronous parallel by having a loop for each of the n individuals in the population: generate new solution, request evaluation (add to queue), wait for evaluation to complete, process solution (e.g., selection, update archive), repeat. Consequently, there are approximately n solutions pending evaluation at any given time. All evaluations are tracked, and an elitist archive is maintained, the fronts shown originate from the archive.

Finally, in preliminary experiments, we found that for ImageNet (a) there are low performance, yet low computational cost networks on the approximation front, and have accuracies ranging from 0.10 to 0.30. In practice, the most interesting networks are often found at the higher end of the performance scale. Finding these low performance networks consumes a significant portion of the computational budget. To ensure that EAs allocate a larger portion of the budget towards finding high-performing networks, we utilize a constraint annealing scheme similar to Adaptive Steering [4] to steer the search towards higher accuracy values. This is implemented by introducing a threshold with respect to accuracy. This threshold linearly increases to 0.5, to gradually exclude these low performance solutions with some margin, until half the available budget has been used. Solutions below this threshold are

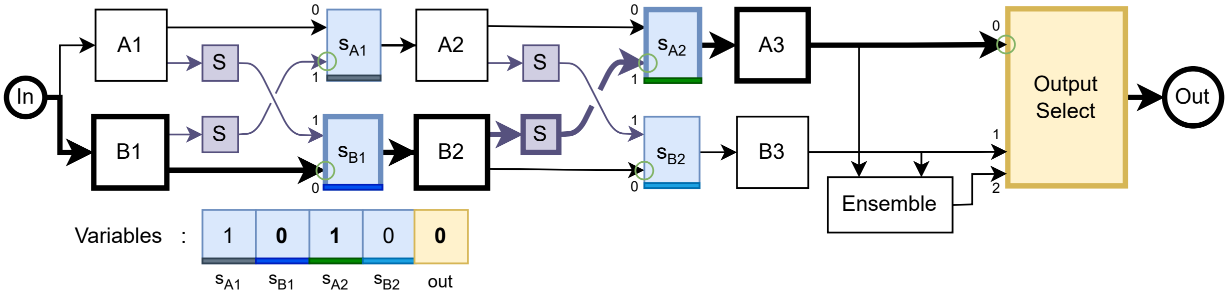

Figure 3: A diagram representing the network generated for the example in Figure 2, consisting of layers of the basis networks A and B, stitches, and switches. The fixed length genotype is a discrete decision vector which for each value determines which input to take for each switch. Chosen inputs have been marked with green circles. By working backwards from the output, taking only the selected input in the case of a switch, a subnetwork is selected. The layers and arrows involved for the example representation have been made bold. The output layer is a switch which selects between the networks, or the ensemble thereof.

considered always strictly worse than solutions above this threshold, even if other objectives would otherwise have allowed this solution to be a part of the approximation front.

Both approaches employ the same scalarization scheme, based on [23]. This approach performs Tschebysheff scalarization, where scalarization weights are reassigned by collecting and, in random order, assigning each of the weights to the individual with the lowest scalarization without newly assigned weights.

Source code and data are available via Zenodo at [14, 15], and at https://github.com/8uurg/Efficient-Model-Stitching.

5.1 Genetic Algorithm

The ordering of variables is chosen such that related variables are located near each other. We anticipate that a Genetic Algorithm (GA) could utilize this ordering with the right crossover. Both one-point and two-point crossover are reasonable choices, as both copy over neighboring variables together. However, only two-point crossover allows for fragments in the middle of the genotype to be copied over, resulting in a larger variety of offspring to be generated. We therefore create new offspring by performing two-point crossover and uniform mutation with $p = \frac{1}{\ell}$, where ℓ is the string length. Due to the asynchronous nature of the EA we use a replacement-like scheme, similar to the one used in MOEA/D [48], but restricted to the parents to preserve diversity. As such, selection is performed by randomly selecting a parent whose scalarized fitness is worse than the offspring, and replacing it. The offspring is not selected if no such parent exists. For the GA, scalarization weights are reassigned after every n completed evaluations.

5.2 (LK-)GOMEA

The provided ordering provides the GA with structure to be exploited. Yet, linkage in a problem can be complex and contextual [16]. Therefore, learning and adapting to the linkage of a specific problem in an online fashion, as is done linkage-learning EAs, can be useful. In [10] was shown that local-search-like approaches are a strong baseline for neural architecture spaces, which the space spanned by our proposal essentially also represents; with only MO-GOMEA outperforming a simple local searcher, in particular on the high-performance end of the found networks. They hypothesize that this is potentially due to linkage-learning.

We therefore utilize a variant of the aforementioned linkage-learning EA, Gene-pool Optimal Mixing Evolutionary Algorithm (GOMEA) [11]. In short, GOMEA employs a linkage tree, which is learned using UPGMA based on a mutual information matrix. This linkage tree contains sets of genotype indices, which are deemed linked. These indices are employed in a local-search like operation named Gene-pool Optimal Mixing (GOM) where for each of these indices, values are sampled from the population, replaced, and tested for improvement. We implement and apply an asynchronous parallel variant similar to that employed in [13] with improvements judged similar to MO-GOMEA using only Tschebysheff scalarizations [23], as described previously. Scalarization weight reassignment occurs after every n applications of GOM.

As for the given problem, the set of active variables changes depending on the values assigned to the variables themselves. The relationships and dependencies between variables can therefore be different in different parts of the search space. This presence of multiple linkage structures has shown to be detrimental to the performance of linkage learning approaches [16]. Therefore, we also investigate a linkage-kernel variant of GOMEA (LK-GOMEA) that was proposed to be more robust to such variable linkage structure. For LK-GOMEA the linkage is learned individually for each solution over its k-nearest neighbors. Due to the absence of a generational clock, we cannot use the original population sizing scheme. As k was tied to this scheme, we determine k differently. A value for k is chosen by randomly sampling a number of modes for each solution m_i uniformly between 1 and $n/c = m_{\max}$, where c is the minimum neighborhood size. Then, $k = \lceil n/m \rceil$. This per kernel neighborhood size should be more likely to sample a small value, as the expected value is $\log_2(n)$, yet also have the possibility to sample a larger neighborhood to avoid premature convergence.

6 NETWORKS

We perform experiments with respect to two tasks - classification on ImageNet/ImageNetV2 [34, 35] using pre-trained networks from timm [45] and Semantic Segmentation on the PASCAL VOC Dataset [12] using networks from TorchVision [24]. These models are provided as code, rather than a graph representation. During a sample execution we keep track of which module provides input to what other module. Providing us with the graph corresponding to the network. To ensure correctness, for each network imported, we have verified the output of the converted network to be the same as that of the original network on some sample input. Using these imported networks we create stitched networks following the procedure described in Section 3. For each dataset used, the train-validation-test split, and the input networks used for stitching differ, and are described in the following paragraphs.

ImageNet. The data-splits used for the ImageNet classification task are as follows. The training set is the training set of ImageNet [35], using only the samples that also have bounding box annotations (546 545 out of 1 283 166 images in the full dataset) validation is done using all unique samples of ImageNetV2 [34], the test set used here is the validation set of ImageNet [35]. The stitching layers are trained using Adam [20] with a learning rate of $lr = 10^{-3}$ and a batch size of 32 chosen to maximize hardware utilization, using 16 384 random samples (out of 546 545 samples), from the training set - noting that training has generally converged at this point. For evaluation, only the first 1000 samples of the validation set (out of 50 000) are used to keep evaluation costs lower, without going below the number of classes for this problem.

On ImageNet we consider stitching the following two pairs of networks, all originating from the timm model library [45]. The first pair, which we will refer to as ImageNet (a) consists of resnet152 [18] and efficientnet_b4 [40]. The resulting network has a total of 154 matches, or $\ell = 2 \times 154 + 1 = 309$ switches and corresponding decision variables. With the available computational resources for this experiment, we were able to evaluate up to 90 solutions in parallel at a given time, evaluating up to 5 solutions simultaneously per GPU with a batch size of 32. Creating this supernetwork took a total of 1415.52s, approx 23.5 minutes. The majority of which is due to the training of the stitching layers (994.28s, step 4 in Figure 1). Other steps (1-3) added up to a total of 421.24s, most of which is due to finding an acyclic matching (417.02s, step 3).

Second, ImageNet (b) consists of resnet50 [18] and resnext50_32x4d [47], which are two architectures that are more similar, with the latter modifying the repeated block by grouping channels. Stitches are trained identically to (a). The resulting network has 206 matches, resulting in a total of 413 decision variables. Stitching these networks took 1463.62s, about 23.5 minutes, most of which was training the stitching layers (1460.24s, step 4). With the available computational resources for this experiment, we were able to evaluate up to 72 solutions in parallel at a given time, or 4 / GPU.

VOC. To show that this stitching approach can be applied to other tasks with different architectures, we include a different kind of task. This second task is Semantic Segmentation according to the Visual Object Classes (VOC) [2] protocol. For this task, the VOC [2] dataset is used. The splits used are the 2012 training and validation sets, and the 2007 test set. For this task we consider stitching one pair of networks, deeplabv3_mobilenet_v3_large [9] and deeplabv3_resnet50 [9]. The original weights were trained using both the ImageNet [35] and Common Objects in Context (COCO) [1, 22] datasets. Adam is used to train the stitching layers with a learning rate of $lr = 10^{-2}$ for 10 epochs (1464 samples / epoch) - noting that training has generally converged at this point. Evaluation of a network is restricted to the first 64 (out of 1449) samples of the validation set to lower evaluation costs to the similar degree as ImageNet. Creation of this network took 791.37s, using 46.17s for steps (1-3), and 745.20s to train the stitching layers (step 4). The resulting network has a total of 56 matches. This provides a total of $\ell = 113$ decision variables. With the available computational resources for this experiment, we were able to evaluate up to 40 solutions in parallel at a given time, with a batch size of 4, evaluating up to 10 solutions simultaneously per GPU.

7 EXPERIMENTAL SETUP

For ImageNet (a) and VOC, we apply all approaches in Section 5. For ImageNet (b), we only used GA and LK-GOMEA, the best two approaches according to the median hypervolume on ImageNet (a) to save computational resources and time. Our key focus is to assess whether the search space contains high quality networks of interest. All runs are performed with an evaluation budget of 200,000 and a time limit of 24 hours, where the strictest limit applies. We determine the population size for each approach by performing a single run for each $n \in [128, 256, 512, 1024, 2048]$ the chosen population size is then that of the configuration with the highest normalized hypervolume. Using this population size, we perform 5 runs per approach. Evaluations that have been skipped due to no change happening to active variables, as explained in Section 4, still count towards the evaluation budget. The percentage of evaluations skipped will be discussed accordingly. Experiments are either performed on (1) a cluster with 5 nodes. Where each node has 2x Intel Xeon Bronze 3206R CPU @ 1.90GHz, for a total of 16 CPU cores, 93 GB of RAM, and 3x NVIDIA RTX A5000 per machine. Or (2), a single node containing 2x Intel Xeon Platinum 8360Y (2x) @ 2.4 GHz, and 4x NVIDIA A100. All networks were prepared on (1). Network optimization for both the ImageNet tasks was both performed on (1), while VOC was performed on (2). Evaluations performed during the optimization procedure are done using the validation set. The networks on the approximation front, as determined using accuracy on the validation set, also had their accuracy determined on a test set. Plots of the obtained fronts are obtained after applying a threshold on accuracy. This threshold is the accuracy of the worst performing reference network rounded down to the nearest multiple of 0.10. For ImageNet (a and b), $\text{accuracy}_{\text{min}} = 0.7$. For VOC, $\text{accuracy}_{\text{min}} = 0.9$. Hypervolume of the obtained fronts and convergence graphs, including corresponding statistical tests, are provided in the supplementary material. The population size n determined for each task / approach are as follows, where applicable. For ImageNet (a), $n_{\text{GA}} = 256$, $n_{\text{GOMEA}} = 512$ and $n_{\text{LK-GOMEA}} = 512$. For ImageNet (b), $n_{\text{GA}} = 128$ and $n_{\text{LK-GOMEA}} = 512$. For VOC, $n_{\text{GA}} = 256$, $n_{\text{GOMEA}} = 2048$, and $n_{\text{LK-GOMEA}} = 2048$.

8 RESULTS AND DISCUSSION

Skipped Evaluations. Evaluations were sometimes skipped in GOMEA and LK-GOMEA due to all changes being restricted to inactive variables, respectively with a median of 45% of evaluations being skipped for GOMEA and 39% for LK-GOMEA. This resulted in the runs of GOMEA and LK-GOMEA completing much earlier than those of the GA. This high percentage is due to the structure of the linkage tree, which contains many small subsets. In fact, half of the subsets contain only a single variable. As such, the changes made to a solution with respect to its previous evaluated state are

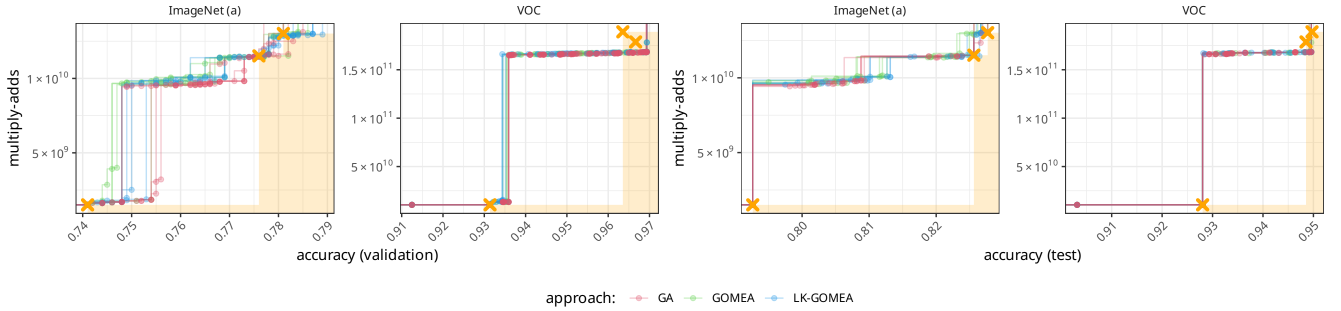

Figure 4: Approximation fronts obtained in each individual run, evaluated on the validation set as evaluated during a run (left) and evaluated on the full test set (right). The basis networks and their ensemble (referred to as 'reference networks') are labeled with an 'x', with the region in which networks dominate them marked in light orange.

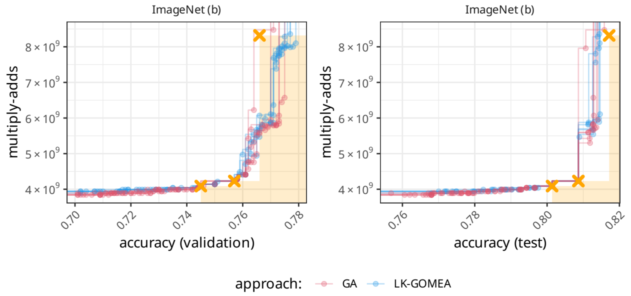

Figure 5: Fronts for ImageNet (b). Similar to Figure 4.

often small, and therefore have a much higher likelihood to consist of only inactive variables. In short, the efficiency improvements gained by this addition are highly dependent on the approach.

Budget Usage. On both ImageNet (a) and (b), all approaches excluding for LK-GOMEA were limited by their evaluation budget. LK-GOMEA terminates due to converged neighborhoods after using a median of 83% of the budget for ImageNet (a) and 93% for ImageNet (b). On the VOC dataset, time was a limiting factor for the GA, with 94% of the evaluation budget being spent in this case. By contrast, GOMEA and LK-GOMEA were limited by the number of evaluations rather than time, due to the time saved by skipping evaluations.

Comparing Approaches. Reviewing the fronts obtained in Figure 4 and 5, it is clear that all approaches find roughly similarly performing networks in terms of the objectives considered, especially so on the test set. In the convergence graphs in the supplementary material it appears that for many runs the GA improves hypervolume much faster, although some runs do stall. This is in part caused by the larger population size selected for the GOMEAs, which was tuned as to maximize end-of-run performance. Usually, GOMEA requires a smaller population size than a GA to reach similar results, which compensates for GOMEA performing many more evaluations per generation. Yet this is not the case here, both GOMEAs require a larger population size than the GA to obtain similar results in terms of hypervolume in the end. This is indicative that GOMEA is unable to (fully) exploit linkage, while the GA using the provided structure, is more capable of doing so. We suspect the presence of inactive variables to play a significant role in this, as detailed in the following section.

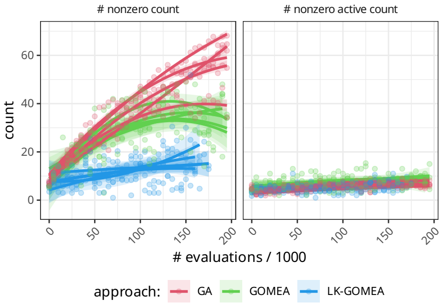

Figure 6: (Averaged) count of *potentially* used stitches (nonzero values), and the number of actually used stitches (active) for solutions evaluated on runs on ImageNet (a).

Inactive Variables. The number of nonzero variables, indicating the *potential* use of a stitch, increases over time, as seen in Figure 6. This may seem unexpected due to the decline in performance when using many stitches. However, this increase is restricted to inactive variables. As these variables do not influence the network that has been selected, these variables do not affect the objective values. In turn, these variables are not subject to selection pressure in the same way active variables are. These inactive variables can significantly affect the solutions being evaluated: if there are many inactive variables indicating the use of a stitch, deviating from this path will quickly lead to an encounter with a previously inactive switch, for which the corresponding variable value indicates the use of a stitch. This likely reroutes one back to the original path through the network. This inevitably affects small changes to the genotype, like those commonly done by GOMEA or a local searcher. In contrast, two-point crossover may simultaneously replace variable values for the previously inactive variables when activating them. This is in part due to the ordering of variables being consistent with a topological sorting of the computational graph of the neural network. These inactive variables are a key difference between this search space and the search space considered in [10]. Future work should therefore investigate the influence of inactive variables, and design approaches which account for them to obtain further performance improvements.

Found Networks. In Figures 4 and 5, we have plotted the approximation fronts and the boundary of the dominated space obtained in each individual run.

For all tasks, we find that the cheapest reference network is not dominated, while the other network is only barely dominated. Networks with a lower computational cost and higher accuracy than these basis networks are difficult to obtain in this search space. Reducing computational cost requires selection of the cheapest layers between the two networks, which, if not all part of the same network, may require the use of many stitches. This currently has a large negative impact on the performance of the resulting networks. Improving the training of the stitches may make finding improvements with a greater number of stitches possible.

Even so, we do find many networks that provide a trade-off of interest. For example, there are networks that provide more performance for a marginal increase in computational costs. Furthermore, the creation of simplified ensembles, i.e., ensembles sharing a portion of the feature extraction process, allow for the creation of networks that dominate the original ensemble, given accuracy on the validation set.

The distribution of points on the fronts obtained on ImageNet (a) and VOC in Figure 4 is quite different compared to those obtained for ImageNet (b) in Figure 5. For the first two the front is noticeably concave, whereas on ImageNet (b) the front is mostly convex. How two networks are matched in steps 1 and 2, has a significant effect on the distribution of multiply-adds of the subnetworks that can be found. When two networks with greatly differing depth are stitched, as is the case for ImageNet (a) and VOC, a large portion of layers in the deeper network may remain without a match in the other network. If these layers are consecutive, all of these layers will either be included or excluded as one unit. This can result in large portion of the computational costs being present or absent, with no potential networks in between. The approach used to match therefore has a large impact on the distribution of multiply-adds. The approach used in this work, matched as early as possible given the possible matches. In effect, for each group of compatible layers a large block is created. This resulted in the staircase for ImageNet (a), and the large cliff for VOC. Altering the matching approach for a better distribution of matches may therefore be necessary to obtain a less concave front.

Overfitting. As stated previously, the performance on the test set indicates that overfitting is a potential issue. When re-evaluating the networks on the fronts on the test set, and redetermining the front, the improvements in accuracy seem to disappear for the cheaper networks, while the more expensive networks do generalize to the test set and remain on the approximation front.

Overfitting to accuracy is possible with minor changes in predictions. For example, a network predicting one of two classes may slightly increase the prediction for the true class. This would appear in the calibration of the predictions of the network, i.e., how well does the predicted probability represent the actual probability that the classification is correct. Furthermore, good calibration may be desirable for many real-world applications. We have therefore investigated the impact on the argmax calibration [17, 25] of the networks using the Expected Calibration Error (ECE) [29] for the best networks found on ImageNet (a). To compute ECE the probabilities of the predicted class are binned, and for each bin the accuracy is calculated. For a calibrated network, the probability that samples in the $0.3 - 0.4$ bin are predicted correctly, should be 0.35.

Instead, we observe a subset of the networks to be underconfident, having higher accuracies in bins than expected. Using an alternative objective, such as cross-entropy loss may help here. However, this is not the case for most of the networks found. It may be that the stitches are not robust enough. As such, improving the accuracy further, may be as simple as training the stitches of a network using the training set. A brief investigation of this is included in the supplementary material.

Finally, the networks found are often simplified ensembles. We find that this provides similar benefits as those obtained by normal ensembles, for example, the resulting networks tend to have a lower calibration error (ECE) than the original basis networks. Yet, their computational cost is still reduced compared to the full ensemble.

Given the aforementioned results we have shown that it is possible to combine two neural networks, without task-specific training, into a supernetwork, and through splicing obtain novel networks that are functional. While the resulting supernetwork is structurally complex, the obtained subnetworks can provide a novel trade-off when compared to the basis networks and their ensemble. We expect this approach to scale to more basis networks, once the matching procedure is adjusted accordingly.

9 CONCLUSION

In this work, we have investigated model stitching as an approach for splicing two neural networks, most notably to simplify the corresponding ensemble while preserving its original performance. The technique can employ already trained neural networks, and construct many new offspring networks without requiring additional training for each offspring. This method provides an alternative pathway for distributed training of neural networks. Using parallel EAs to search through the constructed search space, we found that the resulting networks can provide a novel trade-off between the performance and computational cost of the original basis networks, and potentially even dominate the basis networks.

ACKNOWLEDGMENTS

This publication is part of the project "DAEDALUS - Distributed and Automated Evolutionary Deep Architecture Learning with Unprecedented Scalability" with project number 18373 of the research programme Open Technology Programme which is (partly) financed by the Dutch Research Council (NWO). Other financial contributions as part of this project have been provided by Elekta AB and Ortec Logiqcare B.V.. Furthermore, we thank NWO for the Small Compute grant on the Dutch National Supercomputer Snellius.

REFERENCES

[1] [n.d.]. COCO - Common Objects in Context. https://cocodataset.org/#home

[2] [n.d.]. The PASCAL Visual Object Classes Homepage. http://host.robots.ox.ac.uk/pascal/VOC/

[3] Samuel K. Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. 2023. Git Re-Basin: Merging Models modulo Permutation Symmetries. https://doi.org/10.48550/arXiv.2209.04836 arXiv:2209.04836 [cs]

[4] Tanja Alderliesten, Peter A. N. Bosman, and Arjan Bel. 2015. Getting the Most out of Additional Guidance Information in Deformable Image Registration by Leveraging Multi-Objective Optimization. In *Medical Imaging 2015: Image Processing*, Vol. 9413. SPIE, 469–475. https://doi.org/10.1117/12.2081438

[5] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. 2021. Revisiting Model Stitching to Compare Neural Representations. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 225–236. https://proceedings.neurips.cc/paper/2021/hash/01ded4259d101feb739b06c399e9cd9c-Abstract.html

[6] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for All: Train One Network and Specialize It for Efficient Deployment. In *Eighth International Conference on Learning Representations*. https://iclr.cc/virtual_2020/poster_HylxE1HKwS.html

[7] Rich Caruana. 1997. Multitask Learning. *Machine Learning* 28, 1 (July 1997), 41–75. https://doi.org/10.1023/A:1007379606734

[8] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble Selection from Libraries of Models. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML '04)*. Association for Computing Machinery, New York, NY, USA, 18. https://doi.org/10.1145/1015330.1015432

[9] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. 2017. Rethinking Atrous Convolution for Semantic Image Segmentation. https://doi.org/10.48550/arXiv.1706.05587 arXiv:1706.05587 [cs]

[10] Tom Den Ottelander, Arkadiy Dushatskiy, Marco Virgolin, and Peter A. N. Bosman. 2021. Local Search Is a Remarkably Strong Baseline for Neural Architecture Search. In *Evolutionary Multi-Criterion Optimization*, Hisao Ishibuchi, Qingfu Zhang, Ran Cheng, Ke Li, Hui Li, Handing Wang, and Aimin Zhou (Eds.). Springer International Publishing, Cham, 465–479. https://doi.org/10.1007/978-3-030-72062-9_37

[11] Arkadiy Dushatskiy, Marco Virgolin, Anton Bouter, Dirk Thierens, and Peter A.N. Bosman. 2023. Parameterless Gene-pool Optimal Mixing Evolutionary Algorithms. *Evolutionary Computation* (June 2023), 1–28. https://doi.org/10.1162/evco_a_00338

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision* 88, 2 (June 2010), 303–338.

[13] Arthur Guijt, Dirk Thierens, Tanja Alderliesten, and Peter A.N. Bosman. 2023. The Impact of Asynchrony on Parallel Model-Based EAs. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '23)*. Association for Computing Machinery, New York, NY, USA, 910–918. https://doi.org/10.1145/3583131.3590406

[14] Arthur Guijt, Dirk Thierens, Tanja Alderliesten, and Peter A.N. Bosman. 2024. Exploring the Search Space of Neural Network Combinations Obtained with Efficient Model Stitching - Results Data. https://doi.org/10.5281/zenodo.11120103

[15] Arthur Guijt, Dirk Thierens, Tanja Alderliesten, and Peter A.N. Bosman. 2024. Exploring the Search Space of Neural Network Combinations Obtained with Efficient Model Stitching - Source Code. Zenodo. https://doi.org/10.5281/zenodo.11120074

[16] Arthur Guijt, Dirk Thierens, Tanja Alderliesten, and Peter A. N. Bosman. 2022. Solving Multi-Structured Problems by Introducing Linkage Kernels into GOMEA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*. Association for Computing Machinery, New York, NY, USA, 703–711. https://doi.org/10.1145/3512290.3528828

[17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On Calibration of Modern Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 1321–1330. https://proceedings.mlr.press/v70/guo17a.html

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778. https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html

[19] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. 2017. Snapshot Ensembles: Train 1, Get M for Free. https://doi.org/10.48550/arXiv.1704.00109 arXiv:1704.00109 [cs]

[20] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv:1412.6980 [cs] http://arxiv.org/abs/1412.6980

[21] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of Neural Network Representations Revisited. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 3519–3529. https://proceedings.mlr.press/v97/kornblith19a.html

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014 (Lecture Notes in Computer Science)*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 740–755. https://doi.org/10.1007/978-3-319-10602-1_48

[23] Ngoc Hoang Luong, Tanja Alderliesten, and Peter A. N. Bosman. 2018. Improving the Performance of MO-RV-GOMEA on Problems with Many Objectives Using Tchebycheff Scalarizations. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '18)*. Association for Computing Machinery, New York, NY, USA, 705–712. https://doi.org/10.1145/3205455.3205498

[24] TorchVision maintainers and contributors. 2016. TorchVision: PyTorch's Computer Vision Library. https://github.com/pytorch/vision

[25] Matthias Minderer, Josip Djolonga, Rob Romijnders, Frances Hubis, Xiaohua Zhai, Neil Houlsby, Dustin Tran, and Mario Lucic. 2021. Revisiting the Calibration of Modern Neural Networks. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 15682–15694. https://proceedings.neurips.cc/paper/2021/hash/8420d359404024567b5aefda1231af24-Abstract.html

[26] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2017. Pruning Convolutional Neural Networks for Resource Efficient Inference. https://doi.org/10.48550/arXiv.1611.06440 arXiv:1611.06440 [cs, stat]

[27] Ari Morcos, Maithra Raghu, and Samy Bengio. 2018. Insights on Representational Similarity in Neural Networks with Canonical Correlation. In *Advances in Neural Information Processing Systems*, Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/hash/a7a3d70c6d17a73140918996d03c014f-Abstract.html

[28] Juan Pablo Munoz, Nikolay Lyalyushkin, Chaunte Willetta Lacewell, Anastasia Senina, Daniel Cummings, Anthony Sarah, Alexander Kozlov, and Nilesh Jain. 2022. Automated Super-Network Generation for Scalable Neural Architecture Search. In *Proceedings of the First International Conference on Automated Machine Learning*. PMLR, 5/1–15. https://proceedings.mlr.press/v188/munoz22a.html

[29] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. 2015. Obtaining Well Calibrated Probabilities Using Bayesian Binning. *Proceedings of the AAAI Conference on Artificial Intelligence* 29, 1 (Feb. 2015). https://doi.org/10.1609/aaai.v29i1.9602

[30] Behnam Neyshabur, Hanie Sedghi, and Chiyuan Zhang. 2020. What Is Being Transferred in Transfer Learning?. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 512–523. https://proceedings.neurips.cc/paper/2020/hash/0607f4c705595b911a4f3e7a127b44e0-Abstract.html

[31] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. 2017. Feature Visualization. *Distill* (2017). https://doi.org/10.23915/distill.00007

[32] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (Oct. 2010), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[33] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. 2017. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/hash/dc6a7e655d7e5840e66733e9ee67cc69-Abstract.html

[34] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. 2019. Do ImageNet Classifiers Generalize to ImageNet?. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 5389–5400. https://proceedings.mlr.press/v97/recht19a.html

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (Dec. 2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[36] J.D. Schaffer, D. Whitley, and L.J. Eshelman. 1992. Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. 1–37. https://doi.org/10.1109/COGANN.1992.273950

[37] Eric O. Scott and Kenneth A. De Jong. 2015. Understanding Simple Asynchronous Evolutionary Algorithms. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15)*. Association for Computing Machinery, New York, NY, USA, 85–98. https://doi.org/10.1145/2725494.2725509

[38] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation* 10, 2 (June 2002), 99–127. https://doi.org/10.1162/106365602320169811

[39] George Stoica, Daniel Bolya, Jakob Bjorner, Taylor Hearn, and Judy Hoffman. 2023. ZipIt! Merging Models from Different Tasks without Training. https://doi.org/10.48550/arXiv.2305.03053 arXiv:2305.03053 [cs]

[40] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 6105–6114. https://proceedings.mlr.press/v97/tan19a.html

[41] D. Thierens. 1996. Non-Redundant Genetic Coding of Neural Networks. In *Proceedings of IEEE International Conference on Evolutionary Computation*. 571–575. https://doi.org/10.1109/ICEC.1996.542662

[42] Thomas Uriot and Dario Izzo. 2020. Safe Crossover of Neural Networks Through Neuron Alignment. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference* (June 2020), 435–443. https://doi.org/10.1145/3377930.3390197 arXiv:2003.10306

[43] Abdul Wasay, Brian Hentschel, Yuze Liao, Sanyuan Chen, and Stratos Idreos. 2020. MotherNets: Rapid Deep Ensemble Learning. *Proceedings of Machine Learning and Systems* 2 (March 2020), 199–215. https://proceedings.mlsys.org/paper_files/

paper/2020/hash/4a420924d20bc025ebb37849169e6ebd-Abstract.html

[44] Yeming Wen, Dustin Tran, and Jimmy Ba. 2020. BatchEnsemble: An Alternative Approach to Efficient Ensemble and Lifelong Learning. https://doi.org/10.48550/arXiv.2002.06715 arXiv:2002.06715 [cs, stat]

[45] Ross Wightman. 2019. PyTorch Image Models. https://doi.org/10.5281/zenodo.4414861

[46] Alex H Williams, Erin Kunz, Simon Kornblith, and Scott Linderman. 2021. Generalized Shape Metrics on Neural Representations. In *Advances in Neural Information Processing Systems*, Vol. 34. Curran Associates, Inc., 4738–4750. https://proceedings.neurips.cc/paper/2021/hash/252a3dbaeb32e7690242ad3b556e626b-Abstract.html

[47] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. 2017. Aggregated Residual Transformations for Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1492–1500. https://openaccess.thecvf.com/content_cvpr_2017/html/Xie_Aggregated_Residual_Transformations_CVPR_2017_paper.html

[48] Qingfu Zhang and Hui Li. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (Dec. 2007), 712–731. https://doi.org/10.1109/TEVC.2007.892759