Deep Model Reassembly

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper, we explore a novel knowledge-transfer task, termed as Deep Model Reassembly (DeRy), for general-purpose model reuse. Given a collection of heterogeneous models pre-trained from distinct sources and with diverse architectures, the goal of DeRy, as its name implies, is to first dissect each model into distinctive building blocks, and then selectively reassemble the derived blocks to produce customized networks under both the hardware resource and performance constraints. Such ambitious nature of DeRy inevitably imposes significant challenges, including, in the first place, the feasibility of its solution. We strive to showcase that, through a dedicated paradigm proposed in this paper, DeRy can be made not only possibly but practically efficiently. Specifically, we conduct the partitions of all pre-trained networks jointly via a cover set optimization, and derive a number of *equivalence set*, within each of which the network blocks are treated as functionally equivalent and hence interchangeable. The equivalence sets learned in this way, in turn, enable picking and assembling blocks to customize networks subject to certain constraints, which is achieved via solving an integer program backed up with a training-free proxy to estimate the task performance. The reassembled models, give rise to gratifying performances with the user-specified constraints satisfied. We demonstrate that on ImageNet, the best reassemble model achieves 78.6% top-1 accuracy without fine-tuning, which could be further elevated to 83.2% with end-to-end training. Our code will be made publicly available.

1 Introduction

The unprecedented advances of deep learning and its pervasive impact across various domains are partially attributed to, among many other factors, the numerous *pre-trained models* released online. Thanks to the generosity of our community, models of diverse architectures specializing in the same or distinct tasks can be readily downloaded and executed in a plug-and-play manner, which, in turn, largely alleviates the model reproducing effort. The sheer number of pre-trained models also enables extensive knowledge transfer tasks, such as knowledge distillation, in which the pre-trained models can be reused to produce lightweight or multi-task students.

In this paper, we explore a novel knowledge transfer task, which we coin as *Deep Model Reassembly* (DeRy). Unlike most prior tasks that largely focus on reusing pre-trained models as a whole, DeRy, as the name implies, goes deeper into the building blocks of pre-trained networks. Specifically, given a collection of such pre-trained heterogeneous models or *Model Zoo*, DeRy attempts to first dissect the pre-trained models into building blocks and then reassemble the building blocks to tailor models subject to users' specifications, like the computational constraints of the derived network. As such, apart from the flexibility for model customization, DeRy is expected to aggregate knowledge from heterogeneous models without increasing computation cost, thereby preserving or even enhancing the downstream performances.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

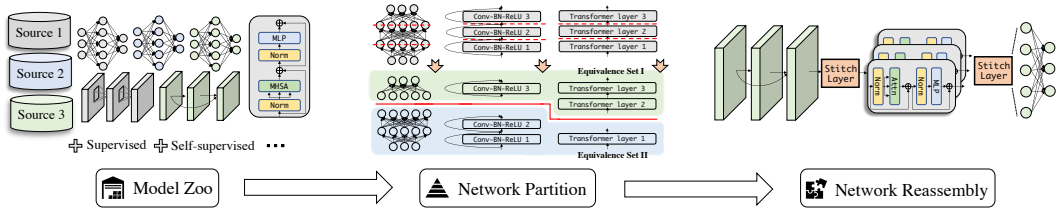

Figure 1: Overall workflow of DeRy. It partitions pre-trained models into equivalent sets of neural blocks and then reassemble them for downstream transfer. Both steps are optimized through solving constrained programs.

Admittedly, the nature of DeRy *per se* makes it a highly challenging and ambitious task; in fact, it is even unclear whether a solution is feasible, given that no constraints are imposed over the model architectures in the model zoo. Besides, the reassembly process, which assumes the building blocks can be extracted in the first place, calls for a lightweight strategy to approximate the model performances without re-training, since the reassembled model, apart from the parametric constraints, is expected to behave reasonably well.

We demonstrate in this paper that, through a dedicated optimization paradigm, DeRy can be made not only possible by highly efficient. At the heart of our approach is a two-stage strategy that first partitions pre-trained networks into building blocks to form *equivalence sets*, and then selectively assemble building blocks to customize tailored models. Each equivalence set, specifically, comprises various building blocks extracted from heterogeneous pre-trained models, which are treated to be functionally equivalent and hence interchangeable. Moreover, the optimization of the two steps is purposely decoupled, so that once the equivalence sets are obtained and fixed, they can readily serve as the basis for future network customization.

We show the overall workflow of the proposed DeRy in Figure 1. It starts by dissecting pre-trained models into disjoint sets of neural blocks through solving a cover set optimization problem, and derives a number of equivalence sets, within each of which the neural blocks are treated as functionally swappable. In the second step, DeRy searches for the optimal block-wise reassembly in a training-free manner. Specifically, the transfer-ability of a candidate reassembly is estimated by counting the number of linear regions in feature representations [43], which reduces the searching cost by 10^4 times as compared to training all models exhaustively.

The reassembled networks, apart from satisfying the user-specified hard constraints, give rise to truly encouraging results. We demonstrate through experiments that, the reassembled model achieves $>$ 78% top-1 accuracy on Imagenet with all blocks frozen. If we allow for finetuning, the performances can be further elevated, sometimes even surpassing any pre-trained network in the model zoo. This phenomenon showcases that DeRy is indeed able to aggregate knowledge from various models and enhance the results. Besides, DeRy imposes no constraints on the network architectures in the model zoo, and may therefore readily handle various backbones such as CNN, transformers, and MLP.

Our contributions are thus summarized as follows.

1. We explore a new knowledge transfer task termed Deep Model Reassembly (DeRy), which enables reassembling customized networks from a zoo of pre-trained models under certain user-specified constraints.
2. We introduce a novel two-stage strategy towards solving DeRy, by first partitioning the networks into equivalence sets and then reassembling neural blocks to customize networks. The two steps are modeled and solved using constrained programming, backed up with training-free performance approximations that significantly speed up the knowledge-trasnfer process.
3. The proposed approach achieves competitive performance on a series of transfer learning benckmarks, sometimes even surpassing than any candidate in the model zoo, which, in turn, sheds light on the the universal connectivity among pre-trained neural networks.

2 Related Work

Transfer learning from Model Zoo. The availability of large-scale model repositories brings about a new problem of transfer learning from a model zoo rather than with a single model. Currently, there are three major solutions. One line of works focuses on select one best model

Problem	No need to retrain	Adaptive Architecture	No Additional Computation	Utilize All Models	Heterogeneous Architecture
Single Model Transfer	✓	✗	✓	✗	✗
Zoo Transfer by Selection	✓	✗	✓	✗	✓
Zoo Transfer by Ensemble	✓	✗	✗	✓	✓
Zoo Transfer by Parameter Fusion	✓	✗	✓	✓	✗
Neural Architecture Search	✗	✓	-	-	-
DeRy	✓	✓	✓	✓	✓

Table 1: Comparison of a series of transfer learning tasks and our proposed Deep Model Reassembly.

within the zoo for deployment, either by exhaustive fine-tuning [32, 56, 65] or quantifying the model transferability [68, 66, 45, 57, 3, 57, 4, 34] on the target task. However, due to the unreliable measurement of transferability, the best model selection may be inaccurate, possibly resulting in a suboptimal solution. The second idea was to apply ensemble methods [15, 70, 1], which inevitably leads to prohibitive computational costs at test time. The third approach is to adaptively fuse multiple pre-trained models into a single target model [54, 14, 55, 59]. However, those methods can only combine homogeneous network structures, whereas most model zoo contains a diverse range of neural architectures. In contrast to conventional approaches in Table 1, DeRy dissects the pre-trained models into building blocks and rearranges them in order to reassemble new pre-trained models.

Neural Representation Similarity. Measuring similarities between deep neural network representations provide a practical tool to investigate the forward dynamics of deep models. Let $\mathbf{X} \in \mathbb{R}^{n \times d_1}$ and $\mathbf{Y} \in \mathbb{R}^{n \times d_2}$ denote two activation matrices for the same n examples. A neural *similarity index* $s(X, Y)$ is a scalar to measure the representations similarity between X and Y, although they do not necessarily satisfy the triangle inequality required of a proper metric. Several methods including linear regression [64, 26], canonical correlation analysis (CCA) [51, 22, 50], centered kernel alignment (CKA) [33]generalized shape metrics [61]. In this study, we leverage the representations similarity towards function level to quantify the distance between two neural blocks.

Network Stitching. Initially proposed by [37], model stitching aims to "plug-in" the bottom layers of one network into the top layers of another network, thus forming a stitched network [2, 12]. It provides an alliterative approach to investigate the representation similarity and invariance of neural networks. A recent line of work achieves competitive performance by stitching a visual transformer on top of the ResNet [56]. Instead of stitching two identical-structured networks in a bottom-top manner, in our study, we investigate to assemble arbitrary pre-trained networks by model stitching.

3 Deep Model Reassembly

In this section, we dive into the proposed DeRy. We first formulate DeRy, and then define the functional similarity and equivalent sets of neural blocks to partition networks by maximizing overall groupbility. The resulting neural blocks are then linked by solving an integer program.

3.1 Problem Formulation

Assume we have a collection of N pre-trained deep neural network models $Z = \{\mathcal{M}_i\}_{i=1}^{N}$ that each composed of $L_i \in \mathbb{N}$ layers of operation $\{F_i^{(k)}\}_{l=1}^{L_i}$, therefore $\mathcal{M}_i = F_i^{(1)} \circ F_i^{(2)} \cdots \circ F_i^{(L_i)}$. Each model can be trained on different tasks or with varied structures. We call Z a *Model Zoo*. We define a learning task T composed of a labeled training set $D_{tr} = \{\mathbf{x}_j, y_j\}_{j=1}^{M}$ and a test set $D_{ts} = \{\mathbf{x}_j\}_{j=1}^{L}$.

Definition 1 *(Deep Model Reassembly) Given a task T, our goal is to find the best-performed L-layer compositional model $\mathcal{M}^*$ on T, subject to hard computational or parametric constraints.*

We therefore formulate it as an optimization problem

$$\mathcal{M}^* = \max_{\mathcal{M}} P_T(\mathcal{M}), \quad s.t. \mathcal{M} = F_{i_1}^{(l_1)} \circ F_{i_2}^{(l_2)} \cdots \circ F_{i_L}^{(l_L)}, |\mathcal{M}| \leq C \tag{1}$$

where $F_i^{(l)}$ is the l-th layer of the i-th model, $P_T(\mathcal{M})$ indicates the performance on T, and $|\mathcal{M}| \leq C$ denotes the constraints. For two consecutive layers with dimension mismatch, we add a single stitching layer with 1×1 convolution operation to adjust the feature size. The stitching layer structure is described in Appendix.

No Single Wins For All. Figure 2 provides a preliminary experiment that 8 different pre-trained

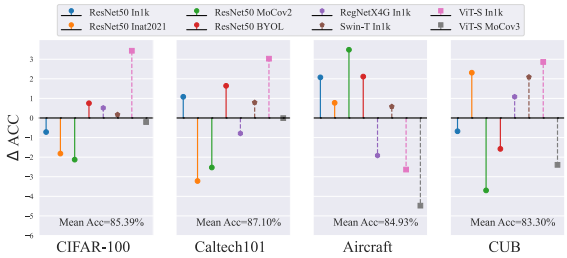

Figure 2: The top-1 accuracy difference between "off-the-shelf" pre-trained models on 4 down-stream tasks.

Backbone	Init.	#Params(M)	Acc(%)
ResNet50	in1k sup	23.71	84.67
	inat2021 sup	23.71	82.57
ResNet50	inat2021(Stage 1&2) in1k(Stage 3&4)	23.98	**85.30**
ResNet50	in1k sup	23.71	84.67
Swin-T		27.60	85.56
ResNet50(Stage 1&2) Swin-T(Stage 3&4)	in1k sup	27.94	**85.77**

Table 3: Accuracy on CIFAR-100 with the pre-trained networks and their reassembled ones.

models are fine-tuned on 4 different image classification tasks. It is clear that no single model universally dominants in transfer evaluations. It builds up our primary motivation to reassemble trained models rather than trust the "best" candidate.

Reassembly Might Win. Table 3 compares the test performance between the reassembled model and its predecessors. The bottom two stages of the ResNet50 iNaturalist2021 (inat2021 sup) [58] are stitched with ResNet50 ImageNet-1k (in1k sup) stage 3&4 to form a new model for fine-tuning on CIFAR100. This reassembled model improves its predecessors by 0.63%/2.73% accuracy respectively. Similar phenomenon is observed on the reassembled model between ResNet50 in1k and Swin-T in1k. Despite its simplicity, the experiment provides concrete evidence that the neural network reassembly could possibly lead to better model in knowledge transfer.

Reducing the Complexity. From the overall $M = \sum_{i=1}^{N} L_i$ layers, the search space of Eq 1 is of size L-permutations of M $P(M, L)$, which is undesirably large. To reduce the overall search cost, we intend to partition the networks into blocks rather than the layer-wise-divided setting. Moreover, it is time-consuming to evaluate each model on the target data through full-time fine-tuning. Therefore, we hope to accelerate the model evaluation, even without model training.

Based on the above discussion, the essence of `DeRy` lies in two steps (1) Partition the networks into blocks and (2) Reassemble the factorized neural blocks. In the following sections, we elaborate on *"what is a good partition?"* and *"what is a good assembly?"*.

3.2 Network Partition by Functional Equivalence

A network partition is a division of a neural network into disjoint sub-nets. In this study, we refer specifically to the partition of neural network $\mathcal{M}_i$ along depth into K blocks $\{B_i^{(k)}\}_{k=1}^{K}$ so that each block is a stack of p layers $B_i^{(k)} = F_i^{(l)} \circ F_i^{(l+1)} \cdots \circ F_i^{(l+p)}$ and k is its stage index. Inspired by the hierarchical property of deep neural networks, we aim to partition the neural networks according to their function level, for example, dividing the network into a "low-level" block that identifies curves and a "high-level" block that recognizes semantics. Although we cannot strictly differentiate "low-level" from "high-level", it is feasible to define *functional equivalence*.

Definition 2 *(Functional Equivalence) Given two functions B and B' with same input space $\mathcal{X}$ and output space $\mathcal{Y}$. $d : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is the metric defined on $\mathcal{Y}$. For all inputs $\mathbf{x} \in \mathcal{X}$, if the outputs are the equivalent $d(B(\mathbf{x}), B'(\mathbf{x})) = 0$, we say B and B' are functional equivalent.*

A function is then uniquely determined by its peers who generate the same output with the same input. However, we can no longer define functional equivalence among neural networks, since network blocks might have varied input-output dimensions. It is neither possible to feed the same input to intermediate blocks with different input dimensions, nor allow for a mathematically valid definition for metric space [10, 5] when the output dimensions are not identical. We therefore resort to recent measurements on neural representation similarity [22, 33] and define the functional similarity for neural networks. The intuition is simple: two networks are functionally similar when they produces similar outputs with similar inputs.

Definition 3 *(Functional Similarity for Neural Networks) Assume we have a neural similarity index $s(\cdot, \cdot)$ and two neural networks $B : \mathcal{X} \in \mathbb{R}^{n \times d_{in}} \to \mathcal{Y} \in \mathbb{R}^{n \times d_{out}}$ and $B' : \mathcal{X}' \in \mathbb{R}^{n \times d'_{in}} \to \mathcal{Y}' \in \mathbb{R}^{n \times d'_{out}}$. For any two batches of inputs $\mathbf{X} \subseteq \mathcal{X}$ and $\mathbf{X}' \subseteq \mathcal{X}'$ with large similarity $s(\mathbf{X}, \mathbf{X}') > \epsilon$, the functional similarity between B and B' are defined as their output similarity $s(B(\mathbf{X}), B'(\mathbf{X}'))$.*

This definition generalizes well to typical knowledge distillation (KD) [26] when $d_{in} = d'_{in}$, which we will elaborate in the Appendix. We also show in Appendix that Def.3 provides a necessary and insufficient condition for two identical networks. Using the method of Lagrange multipliers, the conditional similarity in Def.3 can be further simplified to $S(B, B') = s(B(\mathbf{X}), B'(\mathbf{X}')) + s(\mathbf{X}, \mathbf{X}')$, which is a summation of its input-output similarity. The full derivation is shown in the Appendix.

Finding the Equivalence Sets of Neural Blocks. With Def.3, we are equipped with the math tools to partition the networks into equivalent sets of blocks. Blocks in each set are expected to have high similarity, which are treated to be functionally equivalent and hence interchangeable.

With a graphical notion, we represent each neural network as a path graphs $G(V, E)$ [20] with two nodes of vertex degree 1, and the other $n-2$ nodes of vertex degree 2. The ultimate goal is to find the best partition of each graph into K disjoint sub-graphs along the depth, that sub-graph within each group has maximum internal functional similarity $S(B, B')$. In addition, we take a mild assumption that each sub-graph should have approximately similar size $|B_i^{(k)}| < (1+\epsilon)\frac{|\mathcal{M}_i|}{K}$, where $|\cdot|$ indicates the model size and ϵ is coefficient controls size limit for each block. We solve the above problem by posing a tri-level constrained optimization

$$\max_{B_{a_j}} \quad J(A, \{B_i^{(k)}\}) = \max_{A_{(ik,p)} \in \{0,1\}} \sum_{i=1}^{N} \sum_{j=1}^{K} \sum_{k=1}^{K} A_{(ik,j)} S(B_i^{(k)*}, B_{a_j}) \qquad \text{(Clustering)} \tag{2}$$

$$s.t. \quad \sum_{j=1}^{N_g} A_{(ik,j)} = 1, \quad \{B_i^{(k)*}\}_{k=1}^{K} = \arg\max_{B_i^{(k)}} \sum_{k=1}^{K} A_{(ik,j)} S(B_i^{(k)}, B_{a_j}) \qquad \text{(Partition)} \tag{3}$$

$$s.t. \quad B_i^{(1)} \circ B_i^{(2)} \cdots \circ B_i^{(K)} = \mathcal{M}_i, B_i^{(k_1)} \cap B_j^{(k_2)} = \emptyset, \forall k_1 \neq k_2 \tag{4}$$

$$|B_i^{(k)}| < (1+\epsilon)\frac{|\mathcal{M}_i|}{K}, k = 1, \dots K \tag{5}$$

Where $A \in \mathbb{R}^{KN \times K}$ is the assignment matrix, where $A_{(ik,j)} = 1$ denote the $B_i^{(k)}$ block belongs to the j-th equivalence set, otherwise 0. Note that each block only belongs to one equivalence set, thus each column sums up to 1, $\sum_{j=1}^{N_g} A_{(ik,j)} = 1$. B_{a_j} is the anchor node for the j-th equivalence set, which has the maximum summed similarly with all blocks in set j.

The inner optimization largely resembles the conventional *set cover problem* [27] or $(K, 1+\epsilon)$ *graph partition problem* [29] that directly partition a graph into k sets. Although the graph partition falls exactly in a NP-hard [23] problem, heuristic graph partitioning algorithms like Kernighan-Lin (KL) algorithm [31] and Fiduccia–Mattheyses (FM) algorithm [18] can be applied to solve our problem efficiently. In our implementation, we utilize a variant KL algorithm. With a random initialized network partition $\{B^{(k)}\}_{k=1}^{K}|_{t=0}$ for $\mathcal{M}$ at $t = 0$, we iteratively find the optimal separation by swapping nodes (network layer). Given the two consecutive block $B^{(k)}|_t = F_i^{(l)} \cdots \circ F_i^{(l+p_k)}$ and $B^{(k+1)}|_t = F_i^{(l+p_k+1)} \cdots \circ F_i^{(l+p_k+p_{k+1})}$ at time t, we conduct a forward and a backward neural network layer swap between successive blocks, whereas the partition achieving the largest objective value becomes the new partition

$$(B^{(k)}|_{t+1}, B^{(k+1)}|_{t+1}) = \arg\max\{J(B^{(k)}|_t, B_i^{(k+1)}|_t), J(B_i^{(k)}|_t^{\mathrm{f}}, B_i^{(k+1)}|_t^{\mathrm{f}}), J(B_i^{(k)}|_t^{\mathrm{b}}, B_i^{(k+1)}|_t^{\mathrm{b}})\} \tag{6}$$

$$\text{where } (B_i^{(k)}|_t^{\mathrm{f}}, B_i^{(k+1)}|_t^{\mathrm{f}}) = B^{(k)}|_t \xrightarrow{F_i^{l+p_k}} B_i^{(k+1)}, (B_i^{(k)}|_t^{\mathrm{b}}, B_i^{(k+1)}|_t^{\mathrm{b}}) = B^{(k)}|_t \xleftarrow{F_i^{l+p_k+1}} B_i^{(k+1)} \tag{7}$$

For the outer optimization, we do a K-Means [40] style clustering. With the current network partition $\{B^{(k)*}\}_{k=1}^{K}$, we alternate between assigning each block to a equivalence set G_j, and identifying the anchor block within each set $B_{a_j} \in G_j$. It has been proved that both KL and K-Means algorithms converge to a local minimum according to the initial partition and anchor selection. We repeat the optimization for $R = 200$ runs with different seeds and select the best partition as our final results.

3.3 Network Reassembly by Solving an Integer Program

As we have divided each deep network into K partitions, each belongs to one of the K equivalence sets, all we want now is to find the best combination of neural blocks as a new pre-trained model under certain computational constraints. Consider K disjoint equivalence sets $G_1, \dots, G_K$ of blocks to be reassembled into a new deep network of parameter constraint C_{param} and computational constraint C_{FLOPs}, the objective is to choose exactly one block from each group G_j as well as from each network stage index j such that the reassembled model achieves optimal performance on the target task

without exceeding the capacity. We introduce two the binary matrices $X_{(ik,j)}$ and $Y_{(ik,j)}$ to uniquely identity the reassembled model $\mathcal{M}(X,Y)$. $X_{(ik,j)}$ takes on value 1 if and only if $B_i^{(k)}$ is chosen in group G_j, and $Y_{(ik,j)} = 1$ if $B_i^{(k)}$ comes from the k-th block. The selected blocks are arranged by the block stage index. The problem is formulated as

$$\max_{X,Y} P_T(\mathcal{M}(X,Y)) \tag{8}$$

$$\text{s.t. } |\mathcal{M}(X,Y)| \leq C_{\text{param}}, FLOPs(\mathcal{M}(X,Y)) \leq C_{\text{FLOPs}} \tag{9}$$

$$\sum_{i=1}^{N}\sum_{k=1}^{K} X_{(ik,j)} = 1, X_{(ik,j)} \in \{0,1\}, j = 1,\ldots,K \tag{10}$$

$$\sum_{i=1}^{N}\sum_{j=1}^{K} Y_{(ik,j)} = 1, Y_{(ik,j)} \in \{0,1\}, k = 1,\ldots,K \tag{11}$$

where P_T is again the task performance. Equation 10 and 11 indicates that each model only processes a single block from each equivalence set and each stage index. It falls exactly into a *0-1 Integer Programming* [47] problem with non-linear objective. Conventional methods train each $\mathcal{M}(X,Y)$ to obtain P_T. Instead of training each candidate till convergence, we estimate the transfer-ability of a network by counting the linear regions in the network as a training-free proxy.

Estimating the Performance with Training-Free Proxy The number of linear region [44, 21] is a theoretical-grounded tool to describe the expressivity of a neural network, which has been successfully applied on NAS without training [43, 7]. We, therefore, calculate the data-dependent linear region to estimate the transfer performance of each model-task combination. The intuition is straightforward: the network can hardly learn to distinguish inputs with similar binary codes.

We apply random search to get a generation of reassembly candidates. For a whole mini-batch of inputs, we feed them into each network and binarilize the features vectors using a sign function. Similar to NASWOT [43], we compute the kernel matrix $\mathbf{K}$ using Hamming distance $d(\cdot,\cdot)$ and rank the models using $\log(\det \mathbf{K})$. Since the computation of $\mathbf{K}$ requires nothing more than a few batches of network forwarding, we replace P_T in Equation 8 with NASWOT score for fast model evaluation.

4 Experiments

In this section, we first explore some basic properties of the the proposed `DeRy` task, and then evaluate our solution on a series of transfer learning benchmarks to verify its efficiency.

Model Zoo Setup. We construct our model zoo by collecting pre-trained weights from Torchvision [1], `timm` [2] and OpenMMlab [3]. We includes a series of manually designed CNN models like ResNet [25] and ResNeXt[63], as well as NAS-based architectures like RegNetY [49] and MobileNetv3 [28]. Due to recent popularity of vision transformer, we also take several well-known attention-based architecture into consideration, including Vision Transformer (ViT) [16] and Swin-Transformer [38]. In addition to the differentiation of the network structure pre-trained on ImageNet, we include models with a variety of pre-trained strategies, including SimCLR[6], MoCov2 [8] and BYOL [19] for ResNet50, MoCov3 [9] and MAE [24] for ViT-B. Those models are pre-trained on ImageNet1k [53], ImageNet21K [52] and iNaturalist2021 [58], Finally we result in 21 network architectures, with 28 pre-trained weights in total. We manually identify the atomic node to satisfy our line graph assumption. Each network is therefore a line graph composed of atomic nodes.

Implementation details. For all experiments, we set the partition number $K = 4$ and the block size coefficient $\epsilon = 0.2$. We sample 1/20 samples from each train set to calculate the linear CKA representation similarity. The NASWOT [43] score is estimated with 5-batch average, where each mini-batch contains 32 samples. We set 5 levels of computational constraints, with $C_{\text{param}} \in \{10, 20, 30, 50, 90\}$ and $C_{\text{FLOPs}} \in \{3, 5, 6, 10, 20\}$, which is denoted as `DeRy`(K, C_{param},C_{FLOPs}). For each setting, we randomly generated 500 candidates. Each reassembled model is evaluate under 2 protocols (1) FROZEN-TUNING. We freeze all trained blocks and only update the parameter for

[1] https://pytorch.org/vision/stable/index.html

[2] https://github.com/rwightman/pytorch-image-models

[3] https://github.com/open-mmlab

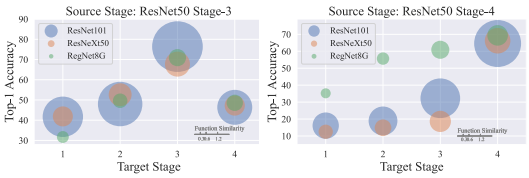

Figure 4: FROZEN-TUNING accuracy on ImageNet by replacing the 3nd and 4th stage of R50 to target blocks.

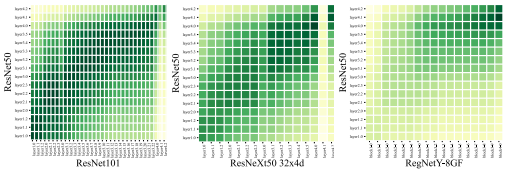

Figure 5: Pair-wise Linear CKA between pre-trained R50 and (1) R101 (2) RX50 and (3) Reg8G.

the stitching layer and the last linear classifier and (2) FULL-TURNING. All network parameter are updated. All experiments are conducted on a $8\times$GeForce RTX 3090 server.

4.1 Exploring the Properties for Deep Reassembly

Similarity, Position and Reassembly-ability. Figure 4 validates our functional similarity, reassembled block selection, and its effect on the model performance. For the ResNet50 trained on ImagNet, we replace its 3nd and 4th stage with a target block from another pre-trained network (ResNet101, ResNeXt50 and RegNetY8G), connected by a single stitching layer. Then, the reassembled networks are re-trained on ImageNet for a 20 epochs under FROZEN-TURNING protocol. The derived functional similarity in Section 3.2 is shown as the diameter of each circle. **We observe that**, the stitching position makes a substantial difference regarding the reassembled model performance. When replaced with a target block with the same stage index, the reassembled model performs surprisingly well, with $\geq 70\%$ top-1 accuracy, even if its predecessors are trained with different architectures, seeds, and hyperparameters. **It is also noted** that, though function similarity is not numerically proportional to the target performance, it correctly reflects the performance ranking within the same target network. It suggests that our function similarity provides a reasonable criteria to identify equivalence set. In sum, the coupling between the *similarity-position-performance* explains our design to select one block from each equivalence set as well as the stage index. We also visualize the linear CKA [33] similarity between the R50 and the target networks in Figure 5. An interesting finding is that *diagonal pattern* for the feature similarity. The representation at the same stage is highly similar. More similarity visualizations are provided in the Appendix.

Partition Results. Due to the space limitation, the partition results of the model zoo are provided in the Appendix. Our observation is that, the equivalent sets tend to *cluster the blocks by stage index*. For example, all bottom layers of varied pre-trained networks are within the same equivalence set. It provides valuable insight that neural networks learns similar patterns at similar network stage, despite of its architecture or training strategy.

Architecture or Pre-trained Weight. Since our proposed `DeRy` searches for the architecture and weights concurrently, a natural question arises that "Do both *architecture* and *pre-trained weights* lead to the final improvement? Or only *architecture* counts?" We provide the experiments in the Appendix that both factors contribute. It is observed that training the `DeRy` architecture from scratch leads to a substantial performance drop compared with `DeRy` model with both new structures and pre-trained weights. It validates our arguments that our reassembled models benefit from the pre-trained models for efficient transfer learning.

Verifying the training-free proxy. As the first attempt to apply the NASWOT to measure model transfer-ability, we verify its efficacy before applying it to `DeRy` task. We adopt the score to rank 10 pre-trained models on 8 image classification tasks, as well as the `timm` model zoo on ImageNet, shown in Figure 6. We also compute the Kendall's Tau correlation [30] between the fine-tuned accuracy and the NASWOT score. It is observed that the NASWOT score provides a reasonable predictor for model transfer-ability with a high Kendall's Tau correlation.

4.2 Transfer learning with Reassembled Model

Evaluation on ImageNet1k. We first compare the reassembled network on ImageNet [53] with current best-performed architectures. We train each model for either 100 epochs as SHORT-TRAINING or a 300 epochs as FULL-TRAINING. Except for `DeRy`, all models are trained from scratch. We optimize each network with AdamW [39] alongside a initial learning rate of $1e-3$ and cosine lr-decay, mini-batch of 1024 and weight decay of 0.05. We apply RandAug [13], Mixup [69] and CutMix [67] as data augmentation. All model are trained and tested on 224 image resolutions.

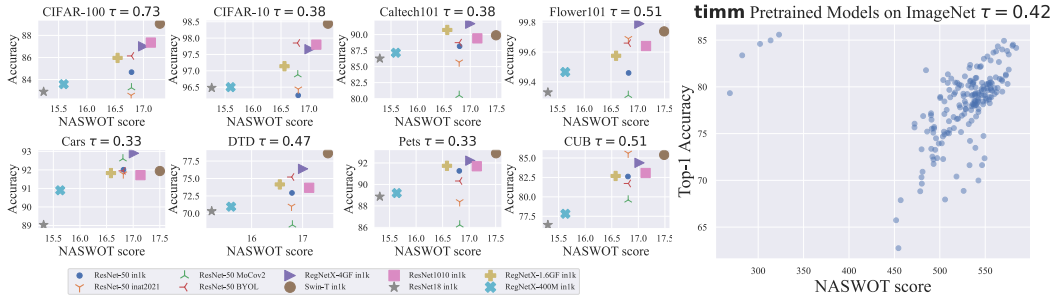

Figure 6: Plots of NASWOT [43] score and test accuracy for (Left) 10 pre-trained model on 8 downstream tasks and (Right) `timm` model zoo on ImageNet. τ is the Kendall's Tau correlation.

Architecture	Train/All Params (M)	FLOPs (G)	Top-1
RSB-ResNet-18	11.69/11.69	1.82	70.6
RegNetY-800M	6.30/6.30	0.8	<u>76.3</u>
ViT-T16	5.7/5.7	1.3	74.1
DeRy(4,10,3)-FZ†	1.02/7.83	2.99	41.2
DeRy(4,10,3)-FT†	7.83/7.83	2.99	76.9
DeRy(4,10,3)-FT	7.83/7.83	2.99	78.4
RSB-ResNet-50	25.56/25.56	4.12	79.8
RegNetY-4GF	20.60/20.60	4.0	79.4
ViT-S16	22.0/22.0	4.6	79.6
Swin-T	28.29/28.29	4.36	<u>81.2</u>
DeRy(4,30,6)-FZ†	1.57/24.89	4.47	60.5
DeRy(4,30,6)-FT†	24.89/24.89	4.47	79.6
DeRy(4,30,6)-FT	24.89/24.89	4.47	81.2
RSB-ResNet-101	44.55/44.55	7.85	81.3
RegNetY-8GF	39.20/39.20	8.1	81.7
Swin-S	49.61/49.61	8.52	<u>82.8</u>
DeRy(4,50,10)-FZ†	3.92/40.41	6.43	72.0
DeRy(4,50,10)-FT†	40.41/40.41	6.43	81.3
DeRy(4,50,10)-FT	40.41/40.41	6.43	82.3
RegNetY-16GF	83.6/83.6	16.0	82.9
ViT-B16	86.86/86.86	33.03	79.8
Swin-B	87.77/87.77	15.14	<u>83.1</u>
DeRy(4,90,20)-FZ†	1.27/ 80.66	13.29	78.6
DeRy(4,90,20)-FT†	80.66/ 80.66	13.29	82.4
DeRy(4,90,20)-FT	80.66/ 80.66	13.29	83.2

Table 7: Top-1 accuracy of models trained on ImageNet. † means the model is trained for 100 epochs. "FZ" and "FT" denote the reassembled blocks are frozen or fine-tuned. Trainable parameters are marked in red.

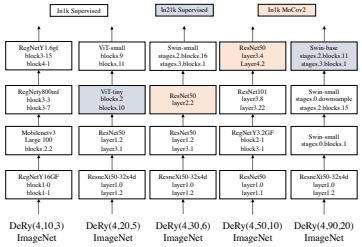

Figure 8: Reassembled structures on ImageNet.

Architecture	Params (M)	FLOPs (G)	Top-1	Top-5
ResNet-50	25.56	4.12	76.8	93.3
Swin-T	28.29	4.36	78.3	94.6
DeRy(30, 6)-FT	**24.89**	4.47	**79.6**	**94.8**
ResNet-101	44.55	7.85	79.0	94.5
Swin-S	49.61	8.52	80.8	**95.7**
DeRy(50, 10)-FT	**40.41**	**6.43**	**81.2**	95.6

Table 9: Top-1 and Top-5 Accuracy for the ImageNet 100-epoch FULL-TUNING experiment.

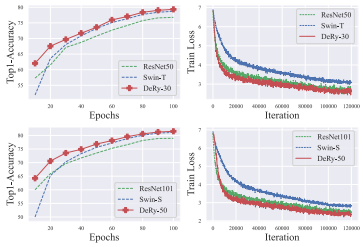

Figure 10: (Left) Test accuracy and (Right) Train loss comparison under the 100-epoch training on ImageNet.

Table 7 provides the Top-1 accuracy comparison on Imagenet with various computational constraint. We <u>underline</u> the best-performed model in the model zoo. **First**, It is worth-noting that DeRy provide very competitive model, even under FROZEN-TURNING or SHORT-TRAINING protocol. DeRy(4,90,20) manages to reach 78.6% with 1.27M parameter trainable, which provides convincing clue that the heterogeneous trained model are largely graftable. With only SHORT-TRAINING, DeRy models also match up with the full-time trained model in the zoo. For example, DeRy(4,10,3) gets to 76.9% accuracy within 100 epochs' training, surpassing all small-sized models. The performance can be further improved towards 78.4% with the standard 300-epoch training. **Second**, DeRy brings about faster convergence. We compare with ResNet-50 and Swin-T under the same SHORT-TRAINING setting in Table 9 and Figure 10. It is clear that, by assembling the off-the-self pre-trained blocks, the DeRy models can be optimized faster than the it competitors, achieving 0.9% and 0.2% accuracy improvement over the Swin-T model with less parameter and computational requirements. **Third**, as showcased in Figure 8, our DeRy is able to search for diverse and hybrid network structures. DeRy(4,10,3) learns to adopt light-weight blocks like MobileNetv3, while DeRy(4,90,20) gets to a large CNN-Swin hybrid architecture. Similar hybrid strategy has been proved to be efficient in manual network design [42, 62].

Transfer Image classification. We evaluate transfer learning performance on 9 natural image datasets. These datasets covered a wide range of image classification tasks, including 3 object classification tasks CIFAR-10 [36], CIFAR-100 [36] and Caltech-101 [17]; 5 fine-grained classification tasks

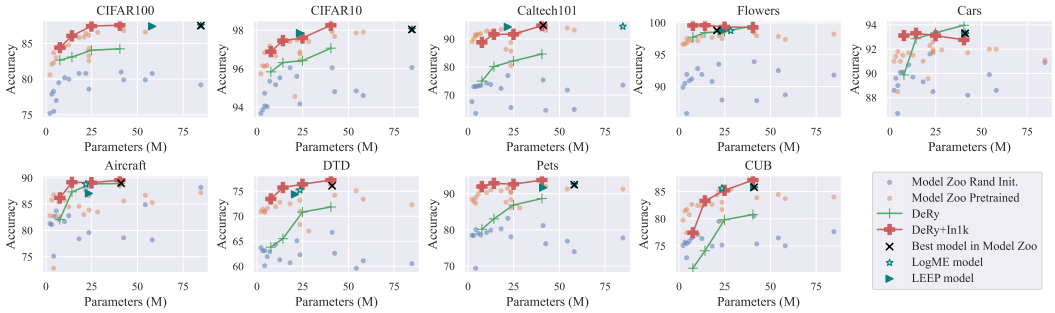

Figure 11: Transfer performance on 9 image classification tasks with the model zoo and our DeRy. Each blue or orange point refers to a single model trained from scratch or pre-trained weights.

Flower-102 [46], Stanford Cars [35], FGVC Aircraft[41], Oxford-IIIT Pets [48] and CUB-Bird [60] and 1 testure classification task DTD [11]. We FULL-TUNE all candidate networks in the model zoo and compare them with our DeRy model. Two model selection strategies LogME [66] and LEEP [45] are also taken as our baselines. For fair comparison, we further train the reassembled network on ImageNet for 100 epochs to further boost the transfer performance. Following [6, 34], we perform hyperparameter tuning for each model-task combination, which are elaborated in the Appendix.

Figure 11 compares the transfer performance between our proposed DeRy and all candidate models. By constructing models from building blocks, the DeRy generally surpasses all network trained from scratch within the same computational constraints, even beats pre-trained ones on Cars, Aircraft, and Flower. If allowing for pre-training on ImageNet (DeRy+In1k), we can further promote the test accuracy, even better than the best-performing candidate in the original model zoo (highlighted by $\times$). The performance improvement rises up as parameter constraints increase, which demonstrates the scalability of the proposed solution. Model selection approaches like LogME and LEEP may not necessarily get the optimal model, thus failing to release the full potential of the model zoo. These findings provide encouraging evidence that DeRy gives rise to an alternative approach to improve the performance when transferring from a zoo of models.

Ablation Study. To study the influence of each stage in our solution, we conduct ablation study by replacing the (1) cover set partition and (2) training-free reassembly with a random search individually. For the partition ablation, we randomly dissect each network into K partitions and reassemble the blocks in an order-less manner using our training-free proxy. For the reassembly ablation, we retain the cover set partition and fine-tune each randomly reassembled network for 100 epochs. Due to the computation limitation, we can only evaluate 25 candidates for reassembly ablation. We report the 100-epoch FROZEN-TUNING top-1 accuracy and the search time on ImageNet in Table 2 under the DeRy(4,50,10) setting. We see that the majority of the search cost comes from the fine-tuning stage. The training-free proxy largely alleviates the tremendous computational cost by 10^4 times, with marginal performance degradation. On the other hand, the cover set model partition not only improves the transfer performance but also reduces the reassembly search space from $O(\prod_{i=1}^{N} \binom{L_i-1}{K-1})$ to $O(1)$. Both stages are crucial.

Cover Set Partition	Train-Free Reassembly	Acc (%)	Search Cost (GPU days)
✓	✓	72.0	0.23
✗	✓	70.5	1.48
✓	✗	73.5	135
✗	✗	72.2	135

Table 2: Ablation study on partition and reassembly strategy.

5 Conclusion

In this study, we explore a novel knowledge-transfer task called Deep Model Reassembly (DeRy). DeRy seeks to deconstruct heterogeneous pre-trained neural networks into building blocks and then reassemble them into models subject to user-defined constraints. We provide a proof-of-concept solution to show that DeRy can be made not only possible but practically efficient. Specifically, pre-trained networks are partitioned jointly via a cover set optimization to form a series of equivalence sets. The learned equivalence sets enable choosing and assembling blocks to customize networks, which is accomplished by solving integer program with a training-free task-performance proxy. DeRy not only achieves gratifying performance on a series of transfer learning benchmarks, but also sheds light on the functional similarity between neural networks by stitching heterogeneous models.

References

[1] Andrea Agostinelli, Jasper Uijlings, Thomas Mensink, and Vittorio Ferrari. Transferability metrics for selecting source model ensembles. *arXiv preprint arXiv:2111.13011*, 2021.

[2] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. *Advances in Neural Information Processing Systems*, 34, 2021.

[3] Yajie Bao, Yang Li, Shao-Lun Huang, Lin Zhang, Lizhong Zheng, Amir Zamir, and Leonidas Guibas. An information-theoretic approach to transferability in task transfer learning. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2309–2313. IEEE, 2019.

[4] Daniel Bolya, Rohit Mittapalli, and Judy Hoffman. Scalable diverse model selection for accessible transfer learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[5] Valeriĭ Buldygin. *Metric characterization of random variables and random processes*.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[7] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021.

[8] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[9] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9640–9649, 2021.

[10] B. Choudhary. *The Elements of Complex Analysis*. New Age International Publishers, 1992.

[11] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[12] Adrián Csiszárik, Péter Kőrösi-Szabó, Ákos Matszangosz, Gergely Papp, and Dániel Varga. Similarity and matching of neural network representations. *Advances in Neural Information Processing Systems*, 34, 2021.

[13] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[14] Dong Dai and Tong Zhang. Greedy model averaging. *Advances in Neural Information Processing Systems*, 24, 2011.

[15] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[17] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pages 178–178. IEEE, 2004.

[18] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *19th design automation conference*, pages 175–181. IEEE, 1982.

[19] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems*, 33:21271–21284, 2020.

[20] Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[21] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, pages 2596–2604. PMLR, 2019.

[22] David R Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural computation*, 16(12):2639–2664, 2004.

[23] Juris Hartmanis. Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). *Siam Review*, 24(1):90, 1982.

[24] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv:2111.06377*, 2021.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7), 2015.

[27] Dorit S Hochba. Approximation algorithms for np-hard problems. *ACM Sigact News*, 28(2):40–52, 1997.

[28] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[29] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3):285–300, 2000.

[30] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[31] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[32] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pages 491–507. Springer, 2020.

[33] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[34] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2661–2671, 2019.

[35] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[36] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[37] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 991–999, 2015.

[38] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *International Conference on Computer Vision (ICCV)*, 2021.

[39] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[40] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[41] S. Maji, J. Kannala, E. Rahtu, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. Technical report, 2013.

[42] Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

[43] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, pages 7588–7598. PMLR, 2021.

[44] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.

[45] Cuong Nguyen, Tal Hassner, Matthias Seeger, and Cedric Archambeau. Leep: A new measure to evaluate transferability of learned representations. In *International Conference on Machine Learning*, pages 7294–7305. PMLR, 2020.

[46] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[47] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[48] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[49] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

[50] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. *Advances in neural information processing systems*, 30, 2017.

[51] JO Ramsay, Jos ten Berge, and GPH Styan. Matrix correlation. *Psychometrika*, 49(3):403–423, 1984.

[52] Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses, 2021.

[53] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[54] Yang Shu, Zhi Kou, Zhangjie Cao, Jianmin Wang, and Mingsheng Long. Zoo-tuning: Adaptive transfer from a zoo of models. In *International Conference on Machine Learning*, pages 9626–9637. PMLR, 2021.

[55] Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. *Advances in Neural Information Processing Systems*, 33:22045–22055, 2020.

[56] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

[57] Anh T Tran, Cuong V Nguyen, and Tal Hassner. Transferability and hardness of supervised classification tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1395–1405, 2019.

[58] Grant Van Horn, Elijah Cole, Sara Beery, Kimberly Wilber, Serge Belongie, and Oisin Mac Aodha. Benchmarking representation learning for natural world image collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12884–12893, 2021.

[59] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. *arXiv preprint arXiv:2002.06440*, 2020.

[60] Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. Technical Report CNS-TR-201, Caltech, 2010.

[61] Alex Williams, Erin Kunz, Simon Kornblith, and Scott Linderman. Generalized shape metrics on neural representations. *Advances in Neural Information Processing Systems*, 34, 2021.

[62] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better. *Advances in Neural Information Processing Systems*, 34:30392–30400, 2021.

[63] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[64] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the national academy of sciences*, 111(23):8619–8624, 2014.

[65] Xingyi Yang, Xuehai He, Yuxiao Liang, Yue Yang, Shanghang Zhang, and Pengtao Xie. Transfer learning or self-supervised learning? a tale of two pretraining paradigms. *arXiv preprint arXiv:2007.04234*, 2020.

[66] Kaichao You, Yong Liu, Jianmin Wang, and Mingsheng Long. Logme: Practical assessment of pre-trained models for transfer learning. In *International Conference on Machine Learning*, pages 12133–12143. PMLR, 2021.

[67] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[68] Guojun Zhang, Han Zhao, Yaoliang Yu, and Pascal Poupart. Quantifying and improving transferability in domain generalization. *Advances in Neural Information Processing Systems*, 34, 2021.

[69] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[70] Zhi-Hua Zhou. Ensemble learning. In *Machine learning*, pages 181–210. Springer, 2021.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...[No]
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...[No]
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]