Deep Model Reassembly
–Supplementary Material–

Xingyi Yang[1] **Zhou Daquan**[1,2] **Songhua Liu**[1] **Jingwen Ye**[1] **Xinchao Wang**[1]
[1]National University of Singapore [2]Bytedance
{xyang,daquan.zhou,songhua.liu}@u.nus.edu, {jingweny,xinchao}@nus.edu.sg

In this document, we provide additional materials that cannot fit into the main manuscript due to the page limit. We first give the full deviation of the function similarity and the pseudo code for the DeRy framework, and then conduct additional experiments to validate the proposed DeRy. Next, we describe our implementation details, dataset settings, evaluation metrics, and hyper-parameter settings.

1 Pseudo-code for DeRy

Algorithm 1 provides the pseudo-code of our two-stage solution for DeRy. Given a model zoo Z, we run the joint network partition for R times. Each partition stage involves a tri-level optimization, with a block partition step, an anchor selection step, and a block assignment step. The optimal network partition yields a number of build blocks, which are then taken to reassemble a new network by solving a constraint integer program with a training-free proxy.

Algorithm 1 Deep Model Reassembly

# Network Partition by Functional Equivalence
for $run = 1, 2, \ldots, R$ **do**
  Initialize the network partition $\{B_i^{(k)}\}_{k=1}^K|_0$ and clustering $A|_0$
  for $t = 1, 2, \ldots, T$ and $J(A|_t, \{B_i^{(k)}\}_{k=1}^K|_t) - J(A|_{t-1}, \{B_i^{(k)}\}_{k=1}^K|_{t-1}) \leq \epsilon$ **do**
    Update the partition by network layer swapping $\{B_i^{(k)}\}_{k=1}^K|_t \leftarrow \arg\max_{B_i^{(k)}} J(A|_{t-1}, \{B_i^{(k)}\}_{k=1}^K|_{t-1})$
    Update the anchor blocks $B_{a_j}|_t \leftarrow \arg\max_{B_{a_j}} J(A|_{t-1}, \{B_i^{(k)}\}_{k=1}^K|_t)$
    Update the block assignment $A|_t \leftarrow \arg\max_A J(A|_{t-1}, \{B_i^{(k)}\}_{k=1}^K|_t)$
  end for
end for
Find the best partition in all runs $A^*, \{B_i^{*(k)}\}_{k=1}^K \leftarrow \arg\max_{A, B_i^{(k)}} J(A, \{B_i^{(k)}\}_{k=1}^K)$.
# Network Reassembly by Solving an Integer Program
for $c = 1, 2, \ldots, num\_candidate$ **do**
  Reassemble a network randomly $M(X, Y)$ under hard constraints.
  Compute the training-free score Scores[c] $\leftarrow$ NASWOT($M(X, Y)$)
end for
Find the best candidate with maximum score $c^* \leftarrow \arg\max_c$ Scores

2 Functional similarity

2.1 Full Derivation of Function Similarity

Following Def. 3, we are able to find the best proxy B' of a given neural network B by solving

$$B'^* = \arg\max_{B'} s(B(\mathbf{X}), B'(\mathbf{X}')), \quad s.t. \quad s(\mathbf{X}, \mathbf{X}') - \epsilon \geq 0 \tag{1}$$

Using Lagrange multipliers, our objective can be simplified to $L(B', \lambda) = s(B(\mathbf{X}), B'(\mathbf{X}')) + \lambda s(\mathbf{X}, \mathbf{X}') - \lambda\epsilon$. By omitting the last term (not related to our B') and setting $\lambda = 1$, we have $S(B, B') = s(B(\mathbf{X}), B'(\mathbf{X}')) + s(\mathbf{X}, \mathbf{X}')$ to characterize the function similarity of two neural blocks, which is a weighted summation of its input-output similarity.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

2.2 Function Similarity and Knowledge Distillation

Knowledge distillation (KD) [6] aims to learn a compact student network $S : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$ from the teacher network $T : \mathbb{R}^{d'_{in}} \rightarrow \mathbb{R}^{d'_{out}}$. For the typical KD problem where $d_{in} = d'_{in}$ and $d_{out} = d'_{out}$, given the same input, we would like to minimize the difference between two models' output logits. Specifically,

- If we adopt **mean-square-error** (MSE), it measures the functional similarity using linear regression R^2_{LR} [8].
- If we adopt **KL divergence** $\mathbb{D}_{KL}\big(S(\mathbf{X})|T(\mathbf{X})\big)$ as our distillation loss, it matches well with the functional similarity with mutual information $MI\big(S(\mathbf{X}), T(\mathbf{X})\big) = H\big(T(\mathbf{X})\big) - \mathbb{D}_{KL}\big(S(\mathbf{X})|T(\mathbf{X})\big)$. We can drop the first term since the teacher output entropy $H\big(T(\mathbf{X})\big)$ is considered as a constant.

The analogy can be brought up when distilling intermediate feature [16] from teacher to student. Here we assume $d_{in} = d'_{in}$ and $d_{out} \neq d'_{out}$. One typical solution is to add one linear layer $W \in \mathbb{R}^{d_{out} \times d'_{out}}$ on top of the student network to match the output dimensions and then minimize the $MSE\big(T(\mathbf{X}), S(\mathbf{X})W\big)$. Because the R^2_{LR} is invariant to invert-able linear transformation [8], as long as W is invert-able, $s\big(S(\mathbf{X})W, T(\mathbf{X})\big) = s\big(S(\mathbf{X}), T(\mathbf{X})\big)$. It again lies in our definition of functional similarity. If we consider the feature distillation by maximizing the mutual information [19, 1], the functional similarity with mutual information describes any form of non-linear transformation instead of a single linear layer.

In sum, KD can be considered as a special case for our functional similarly, when $d_{in} = d'_{in}$.

2.3 Function Similarity of Two Identical Networks

We now prove that function similarity is the necessary but insufficient condition for two identical neural networks. By the term *identical*, we mean the two networks have exactly the same architecture as well as the weights. Let us assume we have an arbitrary representation similarity $s(\cdot, \cdot) \in [0, 1]$ and its corresponding functional similarly $S(\cdot, \cdot) \in [0, 2]$, it is clear that $s(\mathbf{X}, \mathbf{X}) = 1$. We show that for two networks B and B', (1) [B, B' are identical $\Rightarrow S(B, B') = 2$] but (2) [$S(B, B') = 2 \not\Rightarrow B, B'$ are identical].

- **Necessity.** If two networks are identical, given the same input $\mathbf{X}$, two networks ought to produce the same output $B(\mathbf{X}) = B'(\mathbf{X})$, with $s(B(\mathbf{X}), B'(\mathbf{X})) = 1$. Then, we have $S(B, B') = s(B(\mathbf{X}), B'(\mathbf{X})) + s(\mathbf{X}, \mathbf{X}) = 2$.
- **Insufficiency.** We prove that [$S(B, B') = 2 \not\Rightarrow B, B'$ are identical] by contradiction. Suppose B is a single-layer linear neural network $B(\mathbf{X}) = W\mathbf{X}$, and B' is 3-layer network with a scaling and a descaling layer $B'(\mathbf{X}) = (kI)W(\frac{1}{k}I)\mathbf{X}$. Though the two networks have the same output given the same input, they are indeed not identical.

2.4 Computational Complexity Analysis and Solutions

In our formulation, we need to compute the feature similarity within each equivalence set. If we do it along side with the network partition, each network need to be forwarded to get the intermediate feature at every iteration, and then we compute the similarity between $K \times N$ blocks and K centroids for three times (one for clustering and two for a forward and a backward layer swapping). Assume there is n data points ($\frac{1}{20}$ of the full dataset). We run the optimization in Section 3.2 for t steps for each run and repeat the experiment for R times, the time complexity for representation similarity is $\mathcal{O}(K \times N \times K \times t \times R)$ and the time complexity for network inference is $\mathcal{O}(N \times n \times R)$. At the same time, similarity computation for each pair of block is huge. Taking linear CKA as an example, given two feature matrix $X \in \mathbb{R}^{n \times d_1}$ and $Y \in \mathbb{R}^{n \times d_2}$, the computation of linear CKA envolves 3 matrix multiplication

$$CKA(XX^\top, YY^\top) = \frac{||Y^\top X||^2_F}{||XX^\top||_F ||YY^\top||_F} \tag{2}$$

Since the number of samples and feature dimension is large ($n > 10000, d > 1000$), the memory consumption is extremely large. All above factors inevitably leads to huge computational over head.

As the **online computation** is cumbersome in practice, we take an alternative path to fill in the feature similarity table **offline**. We take 3 strategies to significantly reduces the computational complexity in our implementation.

1. **Reduce the Inference Time.** We perform data inference on n samples offline, and save the intermediate feature vectors to local files. This reduce the network inference complexity from $\mathcal{O}(N \times n \times R)$ to $\mathcal{O}(N \times n)$.
2. **Reduce the Similarity Computation.** We compute the similarity for all networks for $\sum_{i=1}^{N} \sum_{j=1}^{N} L_i \times L_j$ times for each pair of layers. This reduces the time complexity from $\mathcal{O}(K \times N \times K \times t \times R)$ to $\mathcal{O}(\sum_{i=1}^{N} \sum_{j=1}^{N} L_i \times L_j)$. This seems to be trivial, however, when we assume $L = 20$, $t = 100$, $R = 200$, $K = 4$, $N = 28$, the online version needs $4 \times 4 \times 28 \times 100 \times 200 = 8,960,000$ computations, and the offline version gets $28 \times 28 \times 20 \times 20 = 313,600$ times. In fact, for each block similarity in online computation, we need to do the pair-wise computation for three times (Line 58). There is a $1 \sim 2$ orders of magnitude speed up.
3. **Reduce the Memory for Large Matrix Multiplication.** Since there involves large matrix multiplication, the memory requirement is extremely large. We implement a mini-batch version full CKA algorithm to save memory and do multi-thresholding. In [13], it has been proved that the mini-batch CKA provides unbiased estimator of CKA value. We also implement a torch-version CKA on GPU to accelerate the computation.

Then, the offline computation for each pair of networks reduces to around $1 \sim 2$ min. For a zoo of 28 models, we need around $28 \times 28 \times 1 \sim 28 \times 28 \times 2$ minutes, with about 12-24 hour in total. Once the similarity computation is done, the partition and reassembly steps can be done without hassle. Please review our code at `similarity/get_rep.py` and `similarity/compute_sim.py` on the implementation details.

3 Limitation, Social Impact and Possible Solutions

Currently, our DeRyhas several limitations, especially with regards to *privacy concerns* and *model bias*. The following discussion of each of those factors provides a more comprehensive reflection on the current research.

- **Privacy Concern.** DeRy combines multiple trained models by reusing some of their parameters. Thus, the reassembled model could contain information from multiple models. Criminals may embezzle the private information or intellectual property of the upstream model by attacking the reassembled model. The problem might be solved by increasing the number of trained models so that the reserved parameters of each model are not sufficient to recover the complete model information.
- **Model Bias.** While DeRy inherits the privileged knowledge of its predecessors, the data bias and knowledge bias may also be transferred to the reassembled model. To tackle this problem, two techniques could be incorporated into the DeRy framework to mitigate the predecessor model bias. **For starters**, we can expand the model zoo size and limit the assembled size for each model. It ensures that no single model bias is dominant in the reassembled model. **It is also possible** to increase the diversity among the reassembled blocks instead of blindly optimizing the target performance. A diversity-promoting regularization term could be added to Equation 8 to provide unbiased predictions.

We will extend our study to those fields to eliminate the current limitation of DeRy.

4 Potential Applications

As DeRy provides a framework to integrate the trained networks and produce a new model, we discuss potential applications on *large model training* and *multi-task learning*.

- **Large model training.** Large models have recently shown their remarkable potential in building powerful AI systems with unprecedented generalization ability and robustness capability. However, training a large model is extremely cumbersome and computationally

intensive. Instead of training a network from scratch, DeRy allows us to take several pre-trained small models, partition them into building blocks and assemble them into the large model as an efficient method for network initialization. As suggested in the paper, reassembling pre-trained models provides faster convergence and reduces the training cost.

- **Multi-task Learning.** Typical multi-tasking learning requires training a model on a complex task combination. DeRy shed light on a very cheap alternative method for multi-task training. Given a bunch of trained single-task models, we can develop a method to aggregate their capacities into a reassembled model. For example, assemble a new model with a shared backbone and multiple task prediction components, each taken from a single task. As such, we reassemble a multi-task model at a very low cost using DeRy.

5 Additional Experimental Results

5.1 Heterogeneous Models VS Homogeneous Models

Although our DeRy was originally designed for fusing heterogeneous models, we would like extend it to homogeneous models to demonstrate its generality. We, therefore, adopt DeRy on homogeneous models and compare the results with those of the Zoo Tuning [18] on CIFAR-100, AirCraft and Cars using the same homogeneous zoo setting as in [18]. Note that we do not further pre-train DeRy on ImageNet to make sure the comparison is fair. As shown in Table 1, We improve the accuracy by 4.13% on Cars and 3.35% on AirCraft dataset, with marginal computational overhead. We indeed outperform [18] significantly with the same experimental setup.

	Param(M)	CIFAR-100	AirCraft	Cars
Zoo Tuning	23.71	83.39	85.51	89.73
DeRy(4, 30, 6)	24.89	**84.05**(+0.66)	**88.86**(+3.35)	**93.86**(+4.13)

Table 1: Performance and computation comparison on homogeneous model zoo.

In fact, a homogeneous model zoo is a simplified case for heterogeneous models. In our main paper, we have focused on the more challenging heterogeneous models.

5.2 Ablation Study on Partition Number, Performance and Complexity

In our main paper, we assume the network partition number $K = 4$. We repeat the network partition and reassembly steps with $K = 5$ and $K = 6$ to see how our method performs with different partition granularities. We set the configuration to DeRy(K, 30, 6). The network is trained on ImageNet for 100 epochs. All other experimental settings remain the same as those used in the main paper. As illustrated in Table 2, we observe that the partition number K has a minimal effect on the model performance. As for the computational complexity, a large K does increase the search time with less than 10% time growth. This ablation study suggests that our proposed DeRy is not sensitive to the selection of partition number K.

K	Param(M)	GFLOPs	Top1 Acc	Top 5 Acc
4	24.89	4.47	79.63	94.81
5	21.14	5.53	79.68	94.89
6	23.38	5.39	79.83	95.02

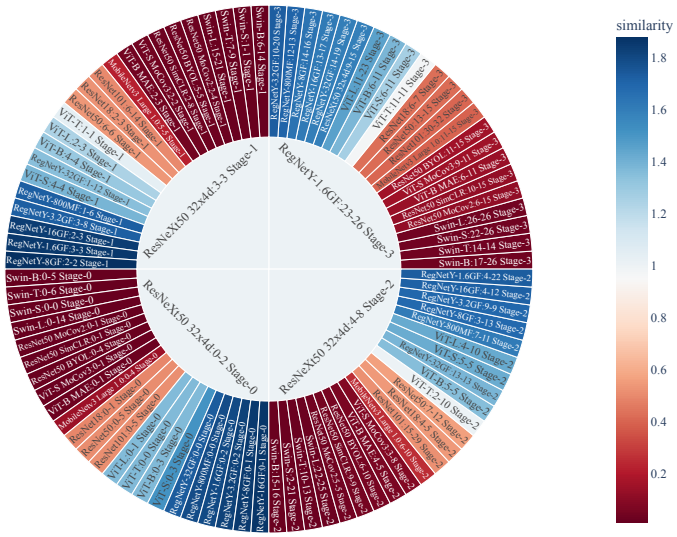
Table 2: Ablation study for the partition number K.

5.3 Partition Results

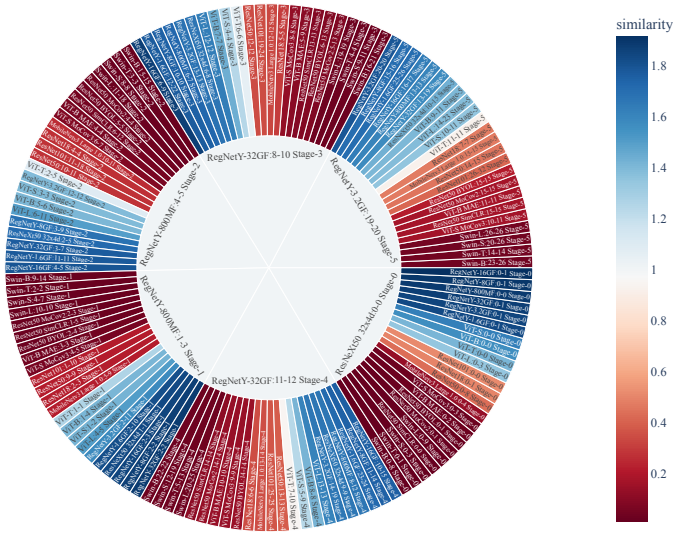
We visualize the network partition hierarchy in Figure 1 with linear CKA when the partition number $K = 4$ and $K = 6$. We name each block in the format of `ModelName-NodeRange-StageIndex`. The anchor blocks are shown in the inner circle, while the outside ring lists all of the blocks in each equivalence set. The panel color indicates the functional similarity between each block and its anchor block.

We make the following observations

1. The equivalent sets tend to *cluster the blocks by stage index*. For example, all Stage 0 blocks of various pre-trained networks are within the same equivalence set. It provides valuable insight that neural networks learn similar patterns at similar network stages, despite of its architecture or training strategy.
2. Our network partition prefers ResNeXt- and RegNet-structured network as its anchor blocks.



(a) $K = 4$



(b) $K = 6$

Figure 1: Model partition and assignment visualization.

5.4 Architecture or Pre-trained Weights?

We would like to investigate whether it is the searched architecture alone, or *both* the network structure *and* the pre-trained weights contribute to the performance improvement in our DeRy solution. Figure 2 compares the test accuracy between full DeRy and DeRy with random weight initialization on 9 transfer learning datasets. We observe that the DeRy model trained from scratch does not gain satisfying transfer performance. It indicates that the searched architecture and the pre-trained weights bring about the performance promotion collaboratively.

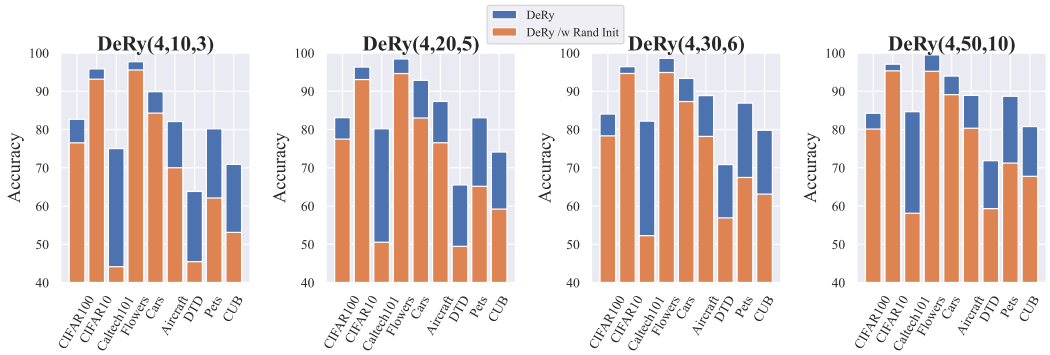
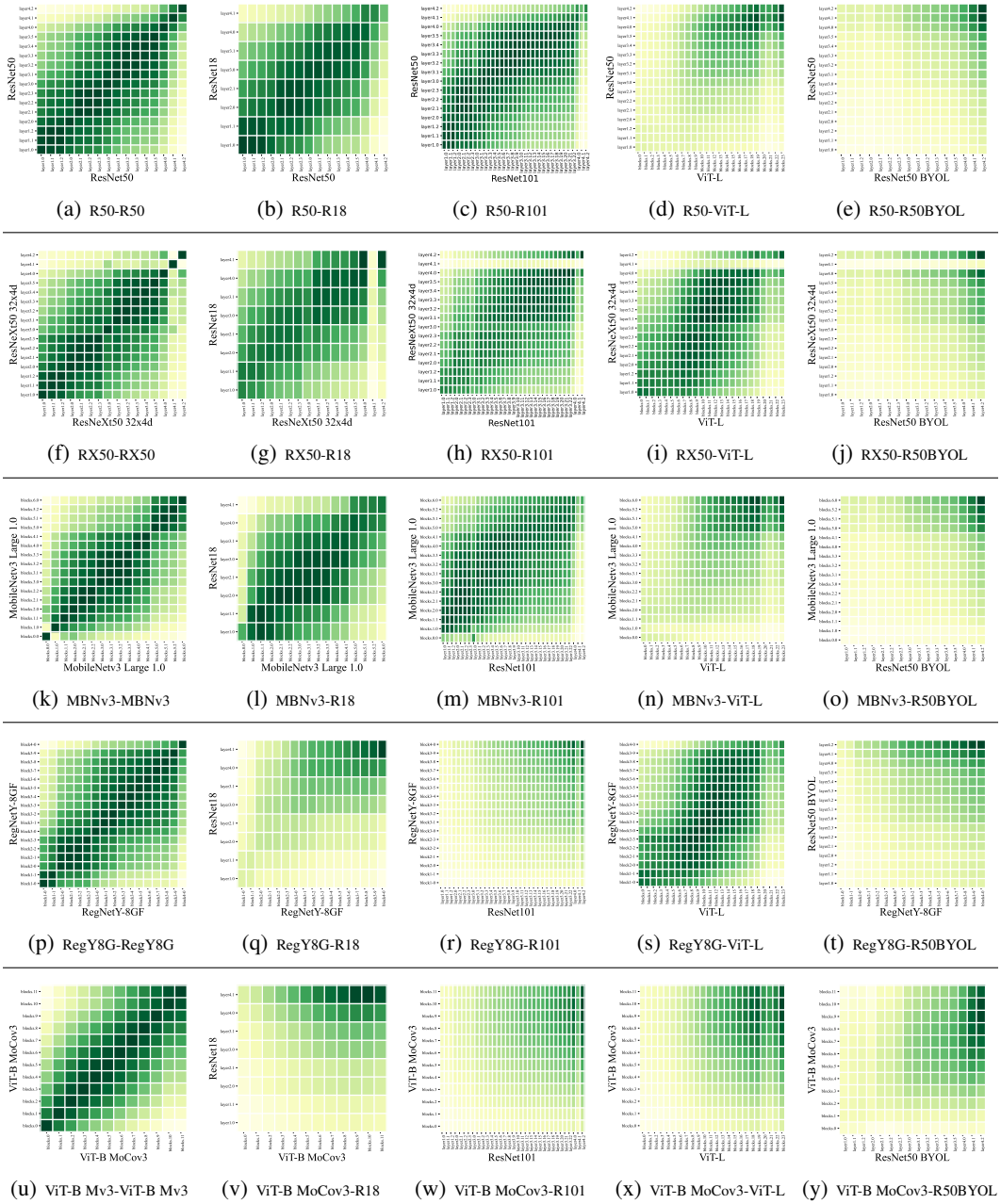


Figure 2: Performance comparison between `DeRy` and `DeRy` with random weight initialization.

5.5 Visualizing the Representation Similarity

Figure 3 visualizes the pairwise network representation similarity heatmap. We include ImageNet1k supervised ResNet-18 (R18), ResNet-50 (R50), ResNet-101 (R101), ResNeXt-50 (RX50), MobileNetv3 Large (MBNv3), Vision Transformer Large (ViT-L), ResNetY-8G (RegY8G), ImageNet1k BYOL ResNet-50 (R50BYOL), and ImageNet1k MoCov3 Vision Transformer Base (ViT-B MoCov3). We have the following observations

- **Diagonal Pattern.** For models trained with diverse network structures, random seeds, and training recipes, representations at similar depth generally have high similarity, resulting in the diagonal pattern in each pairwise heatmap. It again indicates that neural networks learn similar patterns at similar depths.
- **Self-supervised Models Learn High-level Semantics.** As shown in the last row (ViT-B MoCov3) and last column (R50 BYOL) in Figure 3, the self-supervised models achieve high representation similarity with the supervised models at the top layers (top-right on each heatmap). It suggests that the self-supervised learning might be able to learn the high-level semantics of the supervised pre-training. On the contrary, two training paradigms learn distinctive low-level patterns.



(a) R50-R50 (b) R50-R18 (c) R50-R101 (d) R50-ViT-L (e) R50-R50BYOL

(f) RX50-RX50 (g) RX50-R18 (h) RX50-R101 (i) RX50-ViT-L (j) RX50-R50BYOL

(k) MBNv3-MBNv3 (l) MBNv3-R18 (m) MBNv3-R101 (n) MBNv3-ViT-L (o) MBNv3-R50BYOL

(p) RegY8G-RegY8G (q) RegY8G-R18 (r) RegY8G-R101 (s) RegY8G-ViT-L (t) RegY8G-R50BYOL

(u) ViT-B Mv3-ViT-B Mv3 (v) ViT-B MoCov3-R18 (w) ViT-B MoCov3-R101 (x) ViT-B MoCov3-ViT-L (y) ViT-B MoCov3-R50BYOL

Figure 3: Feature similarity heatmap of heterogeneous pre-trained models on ImageNet.

5.6 Numerical Results for Transfer Learning

Table 3 and Table 4 provide the numerical performance on 9 tasks from pre-trained weights or from random initialization. The same results are also shown in Figure 11 in the main paper.

Network	Init.	#Param(M)	CIFAR100	CIFAR10	Caltech101	Flowers	Cars	Aircraft	DTD	Pets	CUB
ResNet-50	iNat2021	23.51	84.81	97.39	92.54	97.86	91.47	84.54	72.83	91.11	82.37
ResNet-50	BYOL	23.51	86.14	97.85	88.74	98.66	91.95	87.04	75.21	90.32	81.72
ResNet-50	MoCov2	23.51	83.26	96.88	80.52	98.31	92.6	88.41	68.35	86.23	79.60
ResNet-50	SimCLR	23.51	82.57	96.44	85.70	98.69	91.78	83.88	71.01	88.38	85.61
ResNet-18	In1K	11.69	83.00	96.17	91.70	97.26	91.00	82.70	71.60	89.90	80.75
ResNet-50	In1k	23.51	84.34	96.77	91.77	97.51	91.72	86.63	73.16	90.54	83.02
ResNet-101	In1K	42.52	86.81	97.68	93.12	97.94	91.69	85.57	74.21	92.02	83.88
ResNeXt 50 32x4d	In1K	25.03	84.62	97.18	91.91	97.18	91.47	85.39	72.92	90.94	82.63
ResNeXt 101 32x8d	In1K	88.79	85.75	97.41	93.47	97.68	91.03	84.66	72.5	90.83	82.63
MobileNet v3 Large 0.75	In1K	4.00	80.60	95.74	89.14	96.63	91.11	82.83	70.88	88.51	79.77
MobileNet v3 Large 1.0	In1K	5.40	82.31	96.13	91.07	97.52	91.80	86.80	71.03	88.99	81.70
RegNetY-800m	in1k	6.30	83.58	96.57	87.16	99.47	87.90	79.99	70.96	89.19	74.82
RegNetY-1.6GF	in1k	9.19	85.96	97.14	90.72	99.58	91.83	86.94	74.15	91.74	82.69
RegNetY-3.2GF	in1k	20.60	85.94	94.57	86.31	98.79	89.57	83.01	74.38	87.69	84.38
RegNetY-8GF	in1k	39.20	86.23	97.72	90.91	99.31	93.04	86.38	75.37	92.57	82.93
RegNetY-16GF	in1k	83.59	87.53	96.32	89.76	98.96	92.34	85.01	75.74	87.69	84.38
RegNetY-32GF	in1k	145.05	90.92	96.87	93.31	99.69	93.51	85.01	74.38	90.69	86.38
Swin-B	in21k	87.77	90.30	98.93	94.90	99.45	93.30	89.00	76.10	91.70	86.80
Swin-T	in1k	28.29	86.56	97.41	92.25	98.74	91.94	83.51	67.04	86.12	85.39
Swin-S	in1k	49.61	89.98	98.71	93.45	99.46	91.24	85.63	78.40	93.83	86.88
Swin-B	in1k	87.77	89.52	98.41	92.76	99.04	91.94	84.51	77.82	89.14	85.97
Swin-L	in1k	196.53	90.76	98.43	92.25	99.24	92.94	86.78	78.14	91.10	87.39
ViT-S	MoCov3	22.88	88.29	96.16	90.62	98.97	88.63	83.76	74.33	88.15	83.11
ViT-B	MAE	86.86	91.82	98.76	93.12	99.63	90.55	80.29	78.40	91.51	86.16
ViT-T	In1k	5.72	87.74	98.09	89.98	99.14	83.67	69.75	73.03	88.54	80.98
ViT-S	In1k	22.88	91.82	98.76	93.13	99.63	90.55	80.29	78.40	91.51	86.16
ViT-B	in1k	86.86	91.70	98.71	91.99	99.74	92.05	79.48	75.90	91.02	86.74
ViT-L	In1K	304.72	90.65	98.34	93.21	97.35	91.98	85.33	73.42	92.47	86.73
DeRy(4,10,3)-FT	DeRy+in1k	7.64	84.41	96.89	88.81	99.58	93.12	86.19	73.46	92.01	77.43
DeRy(4,20,5)-FT	DeRy+in1k	14.19	86.06	97.46	91.67	99.60	93.31	89.17	75.74	92.89	83.28
DeRy(4,30,6)-FT	DeRy+in1k	24.89	87.44	97.60	91.67	99.34	93.11	89.08	76.40	92.64	85.22
DeRy(4,50,10)-FT	DeRy+in1k	40.41	87.55	98.26	94.78	99.25	92.77	89.58	77.23	93.78	87.11

Table 3: Pre-trained transfer performance on 9 image classification tasks.

6 Extension to Other Vision Tasks

Apart from classification, DeRy can be indeed applied to other vision tasks. There are several advantages of DeRy when it is applied to downstream tasks.

First, as DeRy directly searches for a general backbone, we may readily apply the same network to other tasks without any hassle. **Second**, the training-free proxy of NASWOT does not depend on the ground-truth label and is therefore label-agnostic. It enables us to assemble new networks on any task and any modality of input. **Third**, DeRy is highly computationally efficient; it only requires several hours to search for the optimal structure on a large-sized dataset.

We look forward to extending DeRy to other vision tasks in our future work.

7 Implementation details

7.1 Source Code

We implement the model training with the mmClassification[1] framework alongside the Pytorch backend. To support the extraction of sub-model, we require the Pytorch version > 1.10 with the support for `torch.fx`. The NASWOT score is taken from the official implementation[2]. We apply

[1] https://github.com/open-mmlab/mmclassification

[2] https://github.com/BayesWatch/nas-without-training

Network	Init.	#Param(M)	CIFAR100	CIFAR10	Caltech101	Flowers	Cars	Aircraft	DTD	Pets	CUB
ResNet-18	Rand	11.69	77.08	94.03	64.66	90.94	90.11	83.71	61.93	79.10	75.85
ResNet-50	Rand	23.51	78.63	94.17	65.61	87.88	88.51	79.65	62.33	78.09	75.17
ResNet-101	Rand	42.52	79.87	94.81	73.33	87.84	88.21	78.67	62.63	76.22	75.35
ResNeXt 50 32x4d	Rand	25.03	78.34	95.13	72.48	90.51	89.64	81.06	63.11	75.40	75.77
ResNeXt 101 32x8d Large	Rand	88.79	80.23	96.06	74.61	91.77	90.88	88.23	65.51	77.82	77.61
MobileNet v3 Large 0.75	Rand	4.00	75.26	93.68	67.71	89.93	88.58	81.33	63.66	78.50	75.08
MobileNet v3 Large 1.0	Rand	5.40	75.50	94.07	73.11	91.88	89.02	82.69	63.14	80.12	75.44
RegNetY-800m	Rand	6.30	79.53	95.35	73.55	91.22	90.08	82.14	62.88	78.61	76.53
RegNetY-1.6GF	Rand	9.19	80.07	95.53	73.82	91.89	89.64	82.82	61.31	79.86	76.37
RegNetY-3.2GF	Rand	20.60	80.81	96.05	72.38	90.82	89.33	78.44	60.70	80.28	74.94
RegNetY-8GF	Rand	39.20	81.34	94.61	74.88	88.73	88.57	78.21	61.14	77.08	75.03
RegNetY-16GF	Rand	83.59	82.35	95.61	77.23	92.73	90.26	81.21	63.67	78.08	79.03
RegNetY-32GF	Rand	145.05	82.35	95.61	77.23	92.73	90.26	81.21	63.67	78.08	79.03
Swin-T	Rand	28.29	61.12	53.55	46.65	5.32	60.73	68.40	12.13	52.69	42.44
Swin-S	Rand	49.61	67.83	55.61	47.14	13.51	51.64	67.69	25.08	43.17	49.53
Swin-B	Rand	57.77	70.96	56.05	55.43	13.91	62.70	68.79	36.77	51.21	80.48
Swin-L	Rand	196.53	72.91	64.28	56,32	24.26	59.92	64.91	59.58	66.91	76.47
ViT-T	Rand	5.72	52.94	79.53	39.55	84.35	14.01	16.82	46.65	32.69	28.36
ViT-S	Rand	22.88	58.36	84.16	43.28	87.28	19.52	18.18	50.25	37.26	33.35
ViT-B	Rand	86.86	63.26	85.63	45.77	88.27	21.36	19.98	51.37	38.46	36.17
ViT-L	Rand	304.72	67.26	87.71	47.39	88.12	30.18	23.57	52.38	40.07	38.05
DeRy(4,10,3)-FT	DeRy	7.64	82.67	95.84	75.02	97.72	89.88	82.09	63.82	80.18	70.93
DeRy(4,20,5)-FT	DeRy	14.19	83.10	96.32	80.19	98.44	92.86	87.34	65.52	83.07	74.12
DeRy(4,30,6)-FT	DeRy	24.89	84.05	96.42	82.19	98.61	93.35	88.86	70.84	86.9	79.81
DeRy(4,50,10)-FT	DeRy	40.41	84.25	97.07	84.66	99.5	93.97	88.92	71.85	88.69	80.77

Table 4: Randomly-initialized transfer performance on 9 image classification tasks.

`distributed training` and `FP16 half-precision training` to accelerate the model training and save the GPU memory. The code is documented and attached in a zipped file.

7.2 Stitching Layer Structure

Our stitching layer structures are shown in Table 5. For any two network blocks with different input-output sizes, we use one layer of `Norm-Conv1` $\times$ `1-Activation` to align the dimensions and feed the output of the previous block to the next.

Mode	Structure
CNN-to-CNN	`BN` $\rightarrow$ `Conv1`$\times$`1` $\rightarrow$ `LeakyReLU`
CNN-to-Trans/MLP	`BN` $\rightarrow$ `Conv1`$\times$`1` $\rightarrow$ `LeakyReLU` $\rightarrow$ `Flatten`
Trans/MLP-to-CNN	`Reshape(H,W)` $\rightarrow$ `BN` $\rightarrow$ `Conv1`$\times$`1` $\rightarrow$ `LeakyReLU`
Trans/MLP-to-Trans/MLP	`LN` $\rightarrow$ `Linear` $\rightarrow$ `LeakyReLU`

Table 5: Stitching Layer Structure.

7.3 Node Definition

Since `DeRy` assumes that all models are line graphs, we need to manually specify the axiom node in each neural network. In fact, not every operation in the DNN can be treated as an atomic node. Consider a `Conv->ReLU` with skip-connection; we cannot make the single convolution layer as our node because a skip-connection breaks the line graph assumption. `DeRy`, in this current form, can not cut off multiple parallel paths at the same time. Therefore, we need to specify the node in each network. For example, a ViT-B contains 12 transformer blocks and hence has 12 nodes, and a ResNet-18 has 8 residual blocks and hence has 8 nodes.. The node definition can be found in our source code at file

`blocklize/block_meta.py`

7.4 Dataset and Pre-processing

Dataset. We evaluated the DeRy models on 10 image classification datasets in Table 6. These datasets cover a wide range of image classification tasks, including 4 object classification tasks CIFAR-10 [10], CIFAR-100 [10], Caltech-101 [5] and ImageNet1k [17]; 5 fine-grained classification tasks Flower-102 [14], Stanford Cars [9], FGVC Aircraft[12], Oxford-IIIT Pets [15], and CUB-Bird [20] alongside 1 testure classification task DTD [2]. The evaluation metric is either the top-1 accuracy or the per-class mean accuracy, as listed in Table 6.

Dataset	#Classes	#Train	#Test	Accuracy metric
CIFAR-10	10	50,000	10,000	top-1
CIFAR-100	100	50,000	10,000	top-1
Stanford Cars	196	8,144	8,041	top-1
FGVC Aircraft	100	6,667	3,333	mean per-class
Describable Textures (DTD)	47	3,760	1,880	top-1
Oxford-IIIT Pets	37	3,680	3,669	mean per-class
Caltech-101	102	3,060	6,084	mean per-class
Oxford 102 Flowers	102	2,040	6,149	mean per-class
CUB-200-Bird	200	5,994	5,794	top-1
ImageNet1k	1000	1,281,167	50,000	top-1

Table 6: Statistics and evaluation metric of datasets.

Data Pre-processing. Following previous works [7, 4], we train and evaluate on all datasets at the image resolution of 224×224 to align the pre-training and fine-tuning input size. For CIFAR-10 and CIFAR-100 images with original size of 32×32, we first resize them to 224×224. For the other 8 datasets, a crop of random resize ratio $r \in [0.08, 1.0]$ of the original image size and a random aspect ratio $a \in [0.75, 1.33]$ of the original aspect ratio are applied to each training image, and we resize the crop to the 224×224. We apply the same data augmentation on 9 transfer learning evaluations, with a random horizontal image flip with probability of 0.5, a RandAug [3], a random Gaussian noise with maximum kernel size of 10 and probability of 0.1, a RandomErasing [23] with a relative area size $s \in [0.02, 0.33]$ and probability of 0.2, a Mixup [22] operation and a CutMix [21] operation. On the ImageNet evaluation, we utilize a similar data augmentation, just without the Gaussian noise and RandomErasing.

7.5 Hyper-parameter Setting

ImageNet. Following recent training recipe for ImageNet, we train all models using the following setting

- **Batch Size.** We utilize 8 GPUs for distributed training, each contains 128 samples. The overall `batch-size` is 1024.
- **Optimizer.** The networks are optimized with AdamW [11] with initial learning rate 0.001, weight decay of 0.05. Weight decay is not applied to normalization layers and bias term.
- **Training Time.** The models are optimized for 100 epochs for SHORT-TRAINING and 300 epochs for FULL-TRAINING.
- **Learning rate schedule.** We apply the cosine learning decay, with warm-up of 5 epochs.
- **Exponential Moving Average (EMA).** We utilize EMA technique with decay 0.9995.
- **Label Smoothing.** We apply label smoothing with $\epsilon = 0.1$.

Nine Transfer Learning Tasks. We apply grid search to fine the best hyper-parameter setting for each model-task combination.

- **Batch Size.** The `batch-size` is selected from $\{64, 128, 256\}$.

- **Optimizer.** The networks are either optimized with AdamW [11] with initial learning rate of $\frac{0.0005}{512} \times$ `batch-size` or SGD with initial learning rate of $\{0.1, 0.01\}$ and Nesterov momentum of 0.9.
- **Training Time.** The models are optimized for $\{20k, 40k\}$ iterations.
- **Learning rate schedule.** We apply the cosine learning decay, without warm-up.
- **Exponential Moving Average (EMA).** We utilize EMA technique with decay 0.9995.
- **Label Smoothing.** We apply label smoothing with $\epsilon = 0.1$.

References

[1] Sungsoo Ahn, Shell Xu Hu, Andreas Damianou, Neil D Lawrence, and Zhenwen Dai. Variational information distillation for knowledge transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9163–9171, 2019.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

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

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

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

[13] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. *arXiv preprint arXiv:2010.15327*, 2020.

[14] 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.

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

[16] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[17] 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.

[18] 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.

[19] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive representation distillation. *arXiv preprint arXiv:1910.10699*, 2019.

[20] 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.

[21] 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.

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

[23] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13001–13008, 2020.