Self-Stitching: Widely Applicable and Efficient Transfer Learning Using Stitching Layer

Tanachai Anakewat[1], Yusuke Mukuta[1,2], Thomas Westfechtel[1], Tatsuya Harada[1,2]
[1] The University of Tokyo
[2] RIKEN AIP
{anakewat,mukuta,thomas,harada}@mi.t.u-tokyo.ac.jp

Abstract

Transfer learning is a widely used technique in deep learning to leverage pre-trained models for new tasks. A common approach to transfer learning under distribution shift is to fine-tune the last layer of a pre-trained model, preserving well-learned features from pre-training while also adapting to the new task or fully fine-tuning the whole model. While significant progress has been made, the increasing complexity of network designs, meta-learning algorithms, and differences in implementation details make a fair comparison difficult. Moreover, no one solution works well in all situations including situations with various target data sizes and types of domain distribution. Inspired by model stitching, we propose a simple but novel transfer learning methodology called 'Self-Stitching' that inserts one convolutional layer called 'stitching layer' inside the feature extractor of a pre-trained model. As a result, our proposed method shows an improvement compared to baselines like linear-wise and cosine-wise transfer learning. It also achieves competitive results to full fine-tuning, across various domain gaps and data sizes with fewer trainable parameters, making it widely applicable and efficient.

1 Introduction

Transfer learning allows the reuse of pre-trained models on new tasks, drastically reducing the need for large training sets and cutting down on training time for training a model on target domains. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. This approach has proven to be highly effective in fields where data is scarce or expensive to generate, such as medical imaging. By using a pre-trained model, we can take advantage of the features it has learned and apply them to the specific novel task, often achieving impressive results with significantly less data. Moreover, transfer learning can also significantly reduce the computational resources required, as the need for training from scratch is eliminated. This makes it a highly efficient and cost-effective method for machine learning tasks.

Moreover, the traditional machine learning approach struggles to generalize quickly from limited examples. As an extreme situation for transfer learning, Few-Shot Learning (FSL) [1, 2] enables learning from a few examples. The FSL model leverages its ability to learn from limited data, often by using prior knowledge or learned features obtained from a pre-trained model. The model might also use techniques like data augmentation to artificially expand the dataset.

Transfer learning and few-shot learning have applications in several fields including medical diagnosis, where data is scarce due to unavailability of data, privacy, safety, or ethics [3]. With FSL, a model can be trained on these few images to recognize the specific patterns or anomalies associated with the disease. FSL also has applications in robotics [4], like replicating human actions and tasks where data is scarce or hard to gather due to privacy or safety concerns, such as drug discovery [5] and language translation [6]. This approach is also highly valuable in reducing efforts in gathering data and

38th Workshop on Fine-Tuning in Machine Learning (NeurIPS 2024).

expensively retrain models in multiple scenarios such as image classification [7], image retrieval [8], object tracking [9], gesture recognition [10], image captioning, visual question answering [11], video event detection [12], language modeling [7] and neural architecture search [13].

Several machine-learning methods try to leverage the importance of pre-trained data, including linear-wise (linear probing) and cosine-wise transfer learning [14, 15], meta-learning [16, 17, 18], embedding learning [9, 19, 7], and generative modeling [20, 1, 21]. However, there's no one solution to all problems in this field as the current approaches are still limited in terms of applicability to different sizes of target data. For example, linear-wise transfer learning which retrains only the classifier usually works well with few-shot data but this approach doesn't guarantee that the model is optimized to the target task as the parameters of the feature extractor aren't updated. The other approach is to fully fine-tune the model which allows the model to be optimized for that task but is usually computationally expensive and high risk of overfitting when there is less data. Moreover, meta-learning requires a large number of tasks to train the model, and generative modeling is limited to a specific domain. Therefore, there's still a lot of room for improvement in transfer learning. This study introduces a widely applicable stitching-based approach to transfer learning, enhancing the model's plasticity to better adapt to new tasks across diverse domain gaps and data sizes.

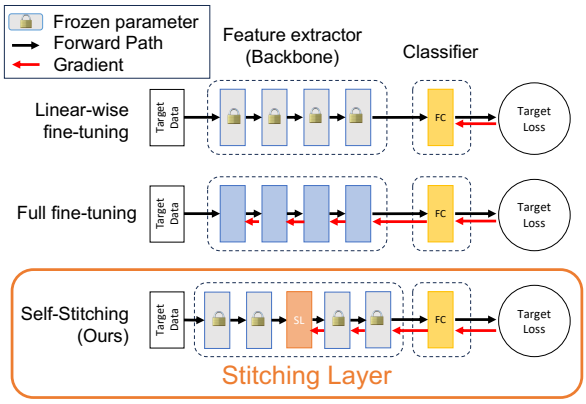


Figure 1: Comparison between traditional transfer learning and the proposed Self-Stitching approach, which adds an extra layer to enhance the model's flexibility and learning capacity.

In this work, we propose a novel transfer learning approach for adapting models to new tasks. Inspired by "model stitching" [22, 23], we enhance model plasticity by inserting a single convolutional "stitching layer" at a specific point during fine-tuning. We focus on fine-tuning image classification tasks across multiple datasets, varying data sizes, and diverse domain gap scenarios. Our main contributions are as follows:

- **Self-Stitching:** We present a novel experiment that adds an extra layer to a pre-trained feature extractor in the transfer learning process, enabling the adaptation of a pre-trained model to a new task using a single additional layer. This method enhances the model's flexibility and learning capacity for tasks beyond its initial training performance and is significantly better than linear-wise transfer learning and performs as well as full fine-tuning.
- **Comparison across various data sizes and domain gaps:** We evaluate the effectiveness of the stitching layer and other baseline methods across multiple data sizes and domain gaps. Our results demonstrate that Self-Stitching performs consistently well in transfer learning tasks, serving as a robust alternative to both linear-wise transfer learning and full fine-tuning. It adapts effectively to both small and large training datasets while maintaining strong performance across diverse domains.
- **Stitching position and pattern analysis:** We conduct a comprehensive exploration of stitching in different model positions. We found that in our task, Self-Stitching works best when added in the middle-later part of the model. Moreover, the accuracy of each Self-Stitching position doesn't vary much showing the robustness of the proposed method.

2 Related works

Transfer learning Prior research has explored the use of fine-tuning to adjust pre-trained models to new, specific tasks [24, 25, 26]. This involves slightly altering the features a model learned during its initial training to make it work better for a new type of data. To ensure that the valuable information learned in the initial training isn't lost, several researchers have suggested different ways to keep the fine-tuning process under control. Some of these methods include keeping certain parts of the model

unchanged to avoid fitting the model too closely to the new task, which can lead to poor performance. Full fine-tuning and linear-wise transfer learning or linear probing are widely used transfer learning methods. Specifically, there is a significant body of research, including recent large-scale studies, that demonstrates the superior performance of FT over LP in-distribution (ID) [27, 28, 29, 15]. There are a variety of other fine-tuning approaches including the following works both in the field of language and vision [30, 31, 32, 33, 34, 35]. Moreover, transfer learning is also being leveraged in few-shot learning task where training data is limited. Some of the approach are linear-wise (linear probing) and cosine-wise transfer learning [14, 15], meta-learning [16, 17, 18], embedding learning [9, 19, 7], and generative modeling [20, 1, 21]. Our work is built on these works, especially by [14] which deeply investigated linear-wise (linear probing) and cosine-wise transfer learning.

Model stitching Model stitching is a method primarily introduced as a way to study the internal representation of models [22]. The method itself was revisited and extended in 2021 as an approach to identify if two models learn similar representation [23]. The following are numerous works on applying model stitching in several ways to reuse pre-trained models. For instance, Deep Reassembly uses stitching layers as connectors between several amputated neural network modules [36]. Moreover, Stitchable Neural Network (SNNet) explores stitching transformer-based models together and studies the stitching strategies between different sizes of the model [37]. Previous works usually train the stitching layer on the same dataset as the model is trained. This work, however, expands the horizon of model stitching by using a stitching layer as an additional trainable layer and trains it on a different model distribution to see if the model can adapt to the new task. This way we can tackle domain adaptation while utilizing the learned representations of pre-trained models in the new task.

3 Self-Stitching

Transfer learning typically involves either full fine-tuning of target data or retraining only the classifier. The first option is computationally expensive and can overfit, while the second does not update the pre-trained model's parameters. We propose a method that integrates additional learnable parameters within the pre-trained feature extractor to efficiently adapt to new tasks, with performance comparable to full fine-tuning. Our system follows a standard transfer learning pipeline: (1) pre-training, (2) fine-tuning, and (3) evaluation.

3.1 Problem setup

Throughout this paper, we use the terms 'source' as the data for pre-training and 'target' as the data for fine-tuning. For given N classes with K labeled examples per class, the source data is denoted as $D_{\text{src}} = \{(x_i, y_i)\}_{i=1}^{N\times K}$ and the target data is denoted as $D_{\text{tgt}} = \{(x'_i, y'_i)\}_{i=1}^{N\times K}$. The goal is to adapt the model trained on the source data ($f^{src}(x)$) to the target data forming a target model ($f^{tgt}(x)$). Each model are comprised of a feature extractor f_θ and a classifier $C\left(\cdot \mid \mathbf{W}_b\right)$. The loss function is denoted as $\mathcal{L}_{\text{pred}}$. The objective is to minimize the loss on the target data D_{tgt}.

3.2 Pipeline

Our proposed system consists of 3 stages which are pre-training, fine-tuning, and evaluation. The objective is to achieve high accuracy with a low loss on target data by leveraging the knowledge from a pre-trained model from source data D_{src} and applying it to the target data D_{tgt}. The source distribution and target distribution can be from the same or different distributions.

Training stage We first trained a model's feature extractor f_θ^{src} and classifier $C\left(\cdot \mid \mathbf{W}_b\right)$ on a source dataset from scratch by minimizing a standard cross-entropy classification loss $\mathcal{L}_{\text{src}}$. The model is trained to minimize the loss on the source dataset D_{src}. In this setup, d represents the dimension of the encoded feature and c stands for the total number of output classes. The classifier, denoted as $C(.|W_b)$, is composed of a linear layer, represented as $W_b^T f_\theta(x_i)$, which is subsequently followed by a softmax function, denoted as σ.

Fine-tuning stage To adapt the model quickly to novel target classes, we fix the pre-trained model parameter θ of our feature extractor f_θ. Then we add a trainable convolutional layer at a specific position of the model called a 'stitching layer' initiated as an identity matrix and reinitialize the model's classifier $C\left(\cdot \mid \mathbf{W}_b\right)$ with random weights $C\left(\cdot \mid \mathbf{W}_n\right)$. We fine-tune this new model by only

updating the parameters of the stitching layer and the classifier layer by minimizing $\mathcal{L}_{\text{pred}}$ on target data D_{tgt}. In the few-shot setting, this fine-tuning stage is done by using only a few target data. In a vanilla setting, this fine-tuning stage is done by using all target data.

Evaluation stage To evaluate, the ability of our target model, we test our model on a test set and compare the accuracy. This process utilizes the top 1 accuracy metric which is the percentage of the time that the model's prediction is the same as the ground truth label. The accuracy used to compare each method is the maximum accuracy across all epochs.

3.3 Detailed analysis of the Self-Stitching

Self-Stitching is a technique specifically designed to add a trainable parameter to a pre-trained model in domain adaption tasks. We first train the model on a source dataset and then add a stitching layer to the pre-trained model. The goal of this model is to allow a single pre-trained model an extra trainable layer that can quickly adapt to new tasks without interrupting any information learned from the previous model. The proposed method can be structured as follows: Concretely, we begin with a model pre-trained on a source dataset, denoted as f^{src}. We can expand f^{src} into $f^{src} = f_n^{src} \circ f_{n-1}^{src} \circ ... \circ f_2^{src} \circ f_1^{src}$, where each layer f_i has parameters θ_i. We then introduce a stitching layer that acts as a fast adapter to new tasks, effectively transforming the pre-train model f^{src}into f^{tgt}. The goal is to minimize the loss of the target data as follows:

$$\mathcal{L}_{tgt}(f^{tgt}) = \mathbb{E}_{(x,y)\sim D_{tgt}}[\ell(f^{tgt}(x), y)] \tag{1}$$

Denote s as a stitching layer that includes 2 batch normalization and a convolutional layer in between, as proposed in [22, 23], we aim to achieve this by constructing a stitched model as follows:

$$f^{tgt}(x) := f_n^{src} \circ \cdots \circ f_{\ell+1}^{src} \circ s \circ f_\ell^{src} \circ \cdots \circ f_1^{src} \tag{2}$$

$$f^{tgt}(x) = f_{>\ell}^{src} \circ s \circ f_{\leq \ell}^{src} \tag{3}$$

where we use ℓ that can minimize the loss of the target data $\mathcal{L}_{tgt}(f^{tgt})$ solving the following optimization problem:

$$\min_{\theta_s, \ell \in \{1,\ldots,n\}} \hat{\mathcal{L}}_{tgt}\left(f(\theta_1, \ldots, \theta_s, \ldots, \theta_n)\right), \tag{4}$$

where $\hat{\mathcal{L}}_{tgt}$ represents the empirical loss on the target task, and $f(\theta_1, \ldots, \theta_s, \ldots, \theta_n)$ symbolizes the fine-tuned model with the stitching layer integrated. The parameters θ_i for layers not selected for fine-tuning ($i \notin S$) remain at their pre-trained values.

Where and what to stitch Previous work on surgical fine-tuning [38] has shown that the optimal layer to fine-tune depends on the type of distribution shift. Specifically, fine-tuning earlier layers is most effective for input-level shifts (e.g., image corruption), while fine-tuning later layers works best for feature-level shifts, such as subgroup shifts, where differences exist between subgroups of the same class. In our study, we redefine transfer learning within CIFAR-100 as a subgroup shift, as it involves adapting a model to different classes within a similar distribution. Consequently, fine-tuning later layers, as we do in Self-Stitching, results in improved performance compared to adjusting earlier layers. Our results align with the findings from surgical fine-tuning, confirming that later layers play a crucial role in handling distribution shifts.

Previous work on SNNET has also mentioned stitching position and stitching direction in the paper [37]. They mentioned that stitching from the smaller model to the larger model, no skip stitching, and no stitching backward (meaning that they emphasize the importance of stitching to the next layer). However, they didn't mention the analysis of the transfer learning ability of the stitching layer.

Can Self-Stitching outperform standard fine-tuning? In this part, we discuss the reason why Self-Stitching can outperform baselines like linear-wise and full fine-tuning. Since linear-wise transfer learning only optimizes the last layer, we can't say that the feature extractor is optimized

for the target dataset. In contrast, Self-Stitching dynamically updates the feature extractor, ensuring better adaptation to new tasks, especially in the many-shot regime.

Compared to full fine-tuning, Self-Stitching offers a more stable approach, especially when data is limited. The reason is that full fine-tuning of the whole model can lead to overfitting where the feature extractor couldn't generalize properly to the test datasets. This instability can prevent the model from converging to an optimal solution during fine-tuning. On the other hand, Self-Stitching is a more stable method as it only adds a small number of trainable parameters to the model and is less likely to lead to overfitting as the model has a lower number of trainable parameters. We utilize this property to realize both the efficiency and the generalization ability of the model making Self-Stitching widely applicable across different transfer learning tasks.

4 Experiments

This study aims to assess whether the proposed method results in a model with improved performance compared to the normal transfer learning approach such as linear-wise transfer learning, cosine-wise transfer learning, and full fine-tuning in various settings.

4.1 Experimental settings

We used the CIFAR100 [39] and CUB-200-2011 [40] datasets to examine adaptability in in-domain transfer learning. CIFAR100 was split into two: 50% of classes for source and the rest for target, with different shots across various experiments.

To test the proposed method's ability in cross-domain transfer learning, we also conducted transfer learning experiments using mini-ImageNet [41] as the source dataset and Caltech101 [42], GTSRB [43], MNIST [44], and Food101 [45] as target datasets. These datasets were chosen based on the varying levels of domain gaps highlighted in [46], allowing us to demonstrate that our method works effectively across both large and small domain shifts.

For the low-shot experiment, we used the CUB dataset, which contains 11,788 images for fine-grained classification. The model was pre-trained on ImageNet and evaluated in a 5-way classification task, extending to 400 epochs due to the task's complexity. The testing scenarios included 5, 20, and 30 shots.

Our experiments used ResNet architectures (18, 34, 50 layers), focusing on later-layer stitching to adapt the model for subgroup shifts. The 'stitching index' refers to specific insertion points for the stitching layer in the ResNet structure, applied immediately before a set of residual blocks. A stitching index of 1 is applied before conv2_x, index 2 before conv3_x, index 3 before conv4_x, and index 4 before conv5_x. The citations conv1, conv2_x, conv3_x, conv4_x, and conv5_x are the names of the layers in the ResNet architecture following the naming convention in the original paper [47].

Adam optimizer with a learning rate of 0.001 was used in the pre-training phase, while SGD with a learning rate of 0.01 momentum of 0.9, and weight decay of 0.001 was used for fine-tuning. We set the batch size of 128 for 100 epochs for pre-training and batch size of 32 with 200 epochs for fine-tuning. The top-1 accuracy served as our evaluation metric, gauging the model's prediction accuracy against actual labels, especially in few-shot settings.

For baselines, we compared our Self-Stitching approach to linear-wise transfer learning (linear probing), cosine-wise transfer learning, and full fine-tuning, noting the trade-offs between performance, computational demand, and overfitting risks.

For evaluation, we use the maximum accuracy achieved across all epochs. In episodic training, such as in few-shot learning, we report the average of the maximum accuracy from each episode. This approach reduces bias from selecting a specific set of n-shots by randomly sampling multiple sets of n-shots and retraining the model episodically.

Our experiments and codes are largely inspired by the implementation from [14]. For certain tasks, such as 5-shot classification using linear probing, our results are consistent with the reported results in the study. This ensures that our baseline comparisons are aligned with established methods in the

field, allowing for a fair evaluation of the proposed Self-Stitching approach against existing transfer learning methods.

4.2 Experimental results

In domain transfer learning: 50 classes as source and target The first experiment conducted in our study presents a comparative analysis of transfer learning methods applied to various ResNet architectures on the CIFAR100 dataset, shown in table 1. The dataset was randomly divided into two equal parts, each representing 50% of the classes. These subsets of data serve as the source and target dataset of transfer learning tasks in the ResNet-18, ResNet-34, and ResNet-50 models. These comparisons underscore the impact of network depth on the transfer learning ability of learned features.

For the pre-trained models trained on the source dataset (labeled as 'Pre-trained'), we observed that ResNet-18 and ResNet-34 showed similar accuracy levels, with ResNet-18 slightly outperforming ResNet-34 at 50.90% and 50.04%, respectively. The deeper ResNet-50 model initially had a lower accuracy of 48.76%, showing the relationship between the depth of each model and the power in the classification task.

These pre-trained models are further used for fine-tuning the target dataset which is the other 50% of classes. The 'Linear' column denotes the accuracy of the linear classifier trained on the target dataset. The 'Cosine' column represents the accuracy of the cosine similarity-based classifier trained on the target dataset. ResNet-18 achieved 39.56%, ResNet-34 achieved 37.30%, and ResNet-50 achieved 40.22% accuracy in linear-wise transfer learning. The use of cosine-wise transfer learning also resulted in lower accuracy than the linear-wise baseline, with accuracies of 31.98% for ResNet-18, 26.68% for ResNet-34, and 26.42% for ResNet-50.

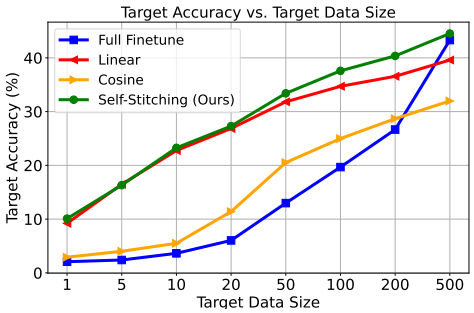
The 'Self-Stitching' column denotes the model's accuracy with the stitching layer trained on the target dataset. Our proposed Self-Stitching method, where we strategically insert trainable layers, outperformed both the linear-wise and cosine-wise baselines. ResNet-18 reached 43.00% accuracy, ResNet-34 reached 42.84%, and ResNet-50 showed a similar performance at 42.3%. Note that the result shown in this table yields from adding a stitching layer at the middle of the model which is before the third residual block.

We observe that the Self-Stitching method outperforms full fine-tuning in two out of three cases. Specifically, Self-Stitching achieves the highest accuracy for ResNet-34 and ResNet-50 models, compared to that of full fine-tuning with accuracies of 41.24% and 38.46% respectively. For the ResNet-18 model, full fine-tuning slightly outperforms Self-Stitching with an accuracy of 43.38% compared to 43.00%. These results demonstrate the effectiveness of the Self-Stitching method in transfer learning scenarios, particularly when using ResNet-34 and ResNet-50 models.

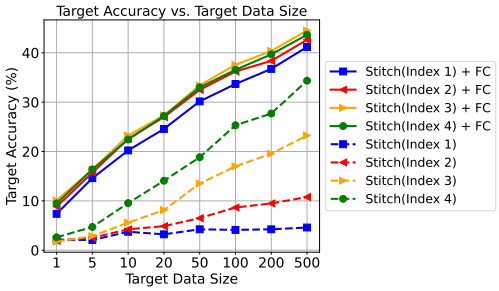
Table 1: Final accuracy of transfer learning methods on CIFAR100 using ResNet models. The first 50% of classes were used for pre-training, while the remaining 50% were used for fine-tuning as the target data.

	50% of classes	Other 50% of classes			
Model	Pre-trained	Linear	Cosine	Self-Stitching	Full fine-tune
ResNet-18	50.90	39.56	31.98	<u>43.00</u>	**43.30**
ResNet-34	50.04	37.30	26.68	**42.84**	<u>41.24</u>
ResNet-50	48.76	40.22	26.42	**42.30**	<u>38.46</u>

Widely applicable transfer learning Moreover, we compared our Self-Stitching method with other baselines across several sizes of target data (the other 50% of classes) including 1, 5, 10, 20, 50, 100, 200, and 500 shots. The objective of this experiment is to show that our Self-Stitching method is widely applicable across different sizes of data in comparison to linear-wise transfer learning which works well in few-shot data and full fine-tuning which works well when there is enough data. According to fig. 2a, The results show that in a setting where there isn't enough data including in 1 and 5 shots, the linear transfer learning outperforms other methods. But when there is enough data, such as more than 10 shots, our Self-Stitching outperforms the linear-wise transfer learning. These results align with the results on table 7.



(a) Comparison of accuracies between several baselines and Self-Stitching on CIFAR100 dataset among different shots.



(b) Accuracy of two variants of Self-Stitching: one reinitializing the fully connected layer (Stitch+FC) and one freezing it (Stitch), measured across different stitching positions.

Figure 2: Side-by-side comparison of baselines and Self-Stitching variations.

When there is more data such as 500 shots, full fine-tuning where all parameters of the model are fine-tuned outperforms the linear where only those of the last layer got fine-tuned, aligning with the result from [15]. In contrast, when there are fewer data(1-200 shots), full fine-tuning which adjusts more parameters tends to overfit as the model fine-tunes more parameters than necessary resulting in forgetting the relevant information from pre-training and failure to generalize against the new data.

For Self-Stitching, the result shows that Self-Stitching consistently performs better than linear fine-tuning in the few-shot scenario especially when there is more data. Moreover, Self-Stitching can also perform at the same level as full fine-tuning even when there is enough data with less trainable parameters. This result shows that Self-Stitching is widely applicable across different sizes of data and can perform well in both few-shot and many-shot learning settings.

According to fig. 2b, the best stitching positions are in the later layers (index 3-4) aligning with the result in [38] which mentioned that fine-tuning later layers is most effective for handling feature-level distribution shifts. The accuracy across different stitching positions varies little, showing the robustness of Self-Stitching. However, freezing the FC layer leads to worse performance compared to reinitializing it, which contrasts with the source-free domain adaptation (SFDA) findings from [48], where freezing the FC layer is recommended. We believe the reason for this difference lies in the nature of the source and target domains. In SFDA, the source and target share the same classes, preserving the feature space structure and making FC layer fine-tuning unnecessary. In our case, however, the source and target have entirely different classes, causing misalignment in the feature space. Thus, reinitializing and fine-tuning the FC layer is essential for better adaptation to the target task.

For Self-Stitching with a frozen fully connected layer, performance improves with later stitching positions, though it still falls short of regular Self-Stitching. This may be because the stitching layer placed at a later stage has more parameters than the one placed at an earlier stage.

Cross-domain: Self-Stitching across various domain gap sizes To test the effectiveness of the proposed Self-Stitching method across a range of domain gaps, we performed transfer learning from a model trained on mini-ImageNet as a source dataset to four distinct target datasets: Caltech101, GTSRB, MNIST, and Food101, representing varying levels of domain shift. The results, shown in table 2, indicate that the Self-Stitching method outperforms the linear baseline and is competitive against full fine-tuning approaches across all datasets, with performance varying based on the stitching position (index 1 to 4). For example, in the Caltech101 dataset, the accuracy increases progressively from 75.12% (index 1) to 82.14% (index 4), exceeding the linear-wise baseline (72.75%) and converge to full fine-tuning (82.26%). For GTSRB and MNIST, the accuracy of Self-Stitching (particularly index 3 and 4) matches or closely rivals full fine-tuning, with Self-Stitching (index 4) reaching 99.57% for GTSRB and 99.39% for MNIST. In the Food101 dataset, which presents a larger domain gap, the performance gap is more evident, with Self-Stitching (index 4) achieving 43.91% comparable to full fine-tuning (45.61%) and better than linear-wise baseline (32.71%), demonstrating the method's adaptability across significant domain shifts.

Table 2: Accuracy of Self-Stitching across various target datasets.

Target	Self-Stitching (Ours)				Linear	Full fine-tune
	Index 1	Index 2	Index 3	Index 4		
Caltech101	75.12	73.91	79.72	82.14	72.75	**82.26**
GTSRB	94.73	97.15	99.38	99.57	84.25	**99.94**
MNIST	98.13	98.53	99.12	99.39	95.99	**99.47**
Food101	32.32	34.06	35.91	43.91	32.71	**45.64**

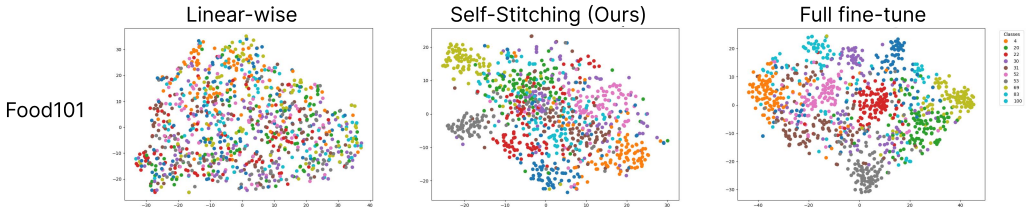


Figure 3: Visualization of the feature space of Food101 on cross-domain experiment

To qualitatively assess the adaptability of Self-Stitching, the visualization of each method is shown in fig. 3. Linear-wise learning struggles with high domain shifts, as evidenced by its less distinct clusters. Self-Stitching(Index 4) improves the separability of the classes, showing tighter, especially in the later stitching positions. Full fine-tuning achieves a similar distribution but at a much higher computational cost. This visual representation further supports the numerical results in table 2, where Self-Stitching, particularly at index 4, approaches full fine-tuning accuracy, demonstrating its capability to handle large domain gaps like those in Food101 while requiring fewer parameters.

These results suggest that the Self-Stitching method can perform comparably to full fine-tuning even in large domain gaps while requiring fewer trainable parameters. The ability to maintain high performance across diverse datasets with varying domain shifts showcases its robustness and wide applicability in different transfer learning scenarios.

Low-shot transfer learning on CUB In this section, we delve into the experiment focusing on low-shot transfer learning, utilizing the CUB dataset. The model, pre-trained on the ImageNet [41] dataset, is evaluated for its proficiency in generalizing to new, sparse data scenarios - these are the 'shots'. A 'shot' represents the limited examples available for fine-tuning in each class of the target dataset.

Table 3: Performance Comparison of Self-Stitching and Linear Transfer Learning in Low-Shot Settings on CUB

Method	5-Shot	20-Shot	30-Shot
Linear	**81.17**	88.85	90.51
Self-Stitching (Ours)	78.96	**90.4**	**91.09**

According to table 3, the stitching layer performs better in low-shot settings when a moderate amount of data (e.g., more than 5-10 shots) is available. However, it is less effective than the linear layer in extremely low-data contexts, like 1 or 5 shots, due to the linear layer's fewer parameters and simpler functions.

5 Conclusion

This study presents Self-Stitching, a new transfer learning method that improves model adaptability by adding an extra convolutional layer to a pre-trained model. The method has shown strong results, surpassing baselines and matching full fine-tuning performance, while offering wide applicability to different data sizes. Specifically, Self-Stitching is better than linear-wise transfer learning in the few-shot region and is competitive with full fine-tuning in the many-shot region. Despite requiring manual selection of the stitching position, future work will automate the selection process and expand validation to tackle data imbalance in image classification.

Acknowledgement

This work was partially supported by JST Moonshot RD Grant Number JPMJPS2011, CREST Grant Number JPMJCR2015 and Basic Research Grant (Super AI) of Institute for AI and Beyond of the University of Tokyo.

References

[1] Li Fei-Fei, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[2] Michael Fink. Object classification from a single example utilizing class relevance metrics. *Advances in neural information processing systems*, 17, 2004.

[3] Jai Kotia, Adit Kotwal, Rishika Bharti, and Ramchandra Mangrulkar. Few shot learning for medical imaging. *Machine learning algorithms for industrial applications*, pages 107–132, 2021.

[4] John J Craig. *Introduction to robotics*. Pearson Educacion, 2006.

[5] Han Altae-Tran, Bharath Ramsundar, Aneesh S Pappu, and Vijay Pande. Low data drug discovery with one-shot learning. *ACS central science*, 3(4):283–293, 2017.

[6] Łukasz Kaiser, Ofir Nachum, Aurko Roy, and Samy Bengio. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*, 2017.

[7] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29, 2016.

[8] Eleni Triantafillou, Richard Zemel, and Raquel Urtasun. Few-shot learning through an information retrieval lens. *Advances in neural information processing systems*, 30, 2017.

[9] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. *Advances in neural information processing systems*, 29, 2016.

[10] Tomas Pfister, James Charles, and Andrew Zisserman. Domain-adaptive discriminative one-shot learning of gestures. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pages 814–829. Springer, 2014.

[11] Xuanyi Dong, Linchao Zhu, De Zhang, Yi Yang, and Fei Wu. Fast parameter adaptation for few-shot image captioning and visual question answering. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 54–62, 2018.

[12] Wang Yan, Jordan Yap, and Greg Mori. Multi-task transfer methods to improve one-shot learning for multimedia event detection. In *BMVC*, pages 37–1, 2015.

[13] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[14] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

[15] Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. *arXiv preprint arXiv:2202.10054*, 2022.

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[17] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International conference on learning representations*, 2016.

[18] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[19] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1199–1208, 2018.

[20] Harrison Edwards and Amos Storkey. Towards a neural statistician. In *International Conference on Learning Representations*, 2017.

[21] Ruslan Salakhutdinov, Joshua Tenenbaum, and Antonio Torralba. One-shot learning with a hierarchical nonparametric bayesian model. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 195–206. JMLR Workshop and Conference Proceedings, 2012.

[22] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. *CoRR*, abs/1411.5908, 2014.

[23] Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting model stitching to compare neural representations. *CoRR*, abs/2106.07682, 2021.

[24] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[25] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.

[26] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[27] 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.

[28] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers, 2021.

[29] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[30] Weifeng Ge and Yizhou Yu. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1086–1095, 2017.

[31] Yunhui Guo, Honghui Shi, Abhishek Kumar, Kristen Grauman, Tajana Rosing, and Rogerio Feris. Spottune: transfer learning through adaptive fine-tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4805–4814, 2019.

[32] Jeffrey O Zhang, Alexander Sax, Amir Zamir, Leonidas Guibas, and Jitendra Malik. Side-tuning: a baseline for network adaptation via additive side networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 698–714. Springer, 2020.

[33] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. Freelb: Enhanced adversarial training for natural language understanding. *arXiv preprint arXiv:1909.11764*, 2019.

[34] Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. Better fine-tuning by reducing representational collapse. *arXiv preprint arXiv:2008.03156*, 2020.

[35] Zhiqiang Shen, Zechun Liu, Jie Qin, Marios Savvides, and Kwang-Ting Cheng. Partial is better than all: revisiting fine-tuning strategy for few-shot learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9594–9602, 2021.

[36] Xingyi Yang, Daquan Zhou, Songhua Liu, Jingwen Ye, and Xinchao Wang. Deep model reassembly. *NeurIPS*, 2022.

[37] Zizheng Pan, Jianfei Cai, and Bohan Zhuang. Stitchable neural networks. In *CVPR*, 2023.

[38] Yoonho Lee, Annie S Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. Surgical fine-tuning improves adaptation to distribution shifts. *International Conference on Learning Representations*, 2023.

[39] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research).

[40] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[41] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[42] 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. *Computer Vision and Pattern Recognition Workshop*, 2004.

[43] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

[44] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[45] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

[46] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[48] Jian Liang, Dapeng Hu, and Jiashi Feng. Do we really need to access the source data? source hypothesis transfer for unsupervised domain adaptation. In *International Conference on Machine Learning (ICML)*, pages 6028–6039, 2020.

Supplementary Material

5.1 Ablation study

Cross testing with 10 classes We established a nuanced experimental setting to rigorously test the models' capabilities. We designated classes 0-9 of the CIFAR-100 dataset as the source model, which serves as the foundational knowledge base. We then systematically targeted this model to a series of class segments, each encompassing a successive group of 10 classes. Specifically, we iteratively set classes 10-19 as the first target segment, followed by classes 20-29 for the second target, and so on. This segmentation created a structured framework that allowed for incremental cross-testing across a diversified class range, with the final target segment comprising classes 90-99. This allows us to test the original model without making changes to the original learned feature of the source model as we only reinitialize the new stitching layer when we move to a new target segment.

The following table presents the outcomes of these methodological applications, offering a detailed comparison of classification accuracies between the Self-Stitching technique and a linear baseline approach.

As shown in table 4, our model can iteratively adapt to new tasks without interrupting the previously trained parameters. Our model also yields better results compared to changing only the last linear layer by average over 10% in top 1 accuracy.

Table 4: Comparison of Self-Stitching with linear-wise transfer learning on a small but wide dataset (%)

Target Class	Self-Stitching	Linear
0-9 (Source)	**72.6**	72.4
10-19	**56.0**	45.6
20-29	**67.6**	44.9
30-39	**57.8**	55
40-49	**60.4**	49.3
50-59	**60.9**	50.4
60-69	**64.1**	50.4
70-79	**63.3**	54
80-89	**59.4**	50.7
90-99	**68.5**	48.6

Visualization of feature space We visualized the feature space for linear-wise transfer learning, Self-Stitching, and full fine-tuning during evaluation, using models from the cross-domain experiment in table 2. This visualization highlights how adding a stitching layer significantly improves the adaptability of the feature extractor, bringing it closer to the performance of full fine-tuning while surpassing linear transfer learning.

The purpose of this visualization is not to show that Self-Stitching is necessarily superior to full fine-tuning, but rather to demonstrate that adding a stitching layer can dynamically adapt the feature extractor, making it nearly as effective as full fine-tuning. In fig. 4, which visualizes the feature space using full-shot training, Self-Stitching achieves feature clustering that is much tighter and more separated compared to linear transfer learning, and comparable to full fine-tuning.

As seen in the fig. 4, for datasets like Caltech101, MNIST, Food101, and GTSRB:

- Linear-wise transfer learning shows scattered and poorly defined clusters, indicating that the feature extractor isn't optimized for the target task.
- Self-Stitching, particularly in the later stitching indices, demonstrates significantly improved feature clustering, with clear class separations across all datasets, approaching the performance of full fine-tuning.
- Full fine-tuning produces well-separated clusters with minimal overlap, which is the expected outcome when the entire model is optimized for the target dataset.

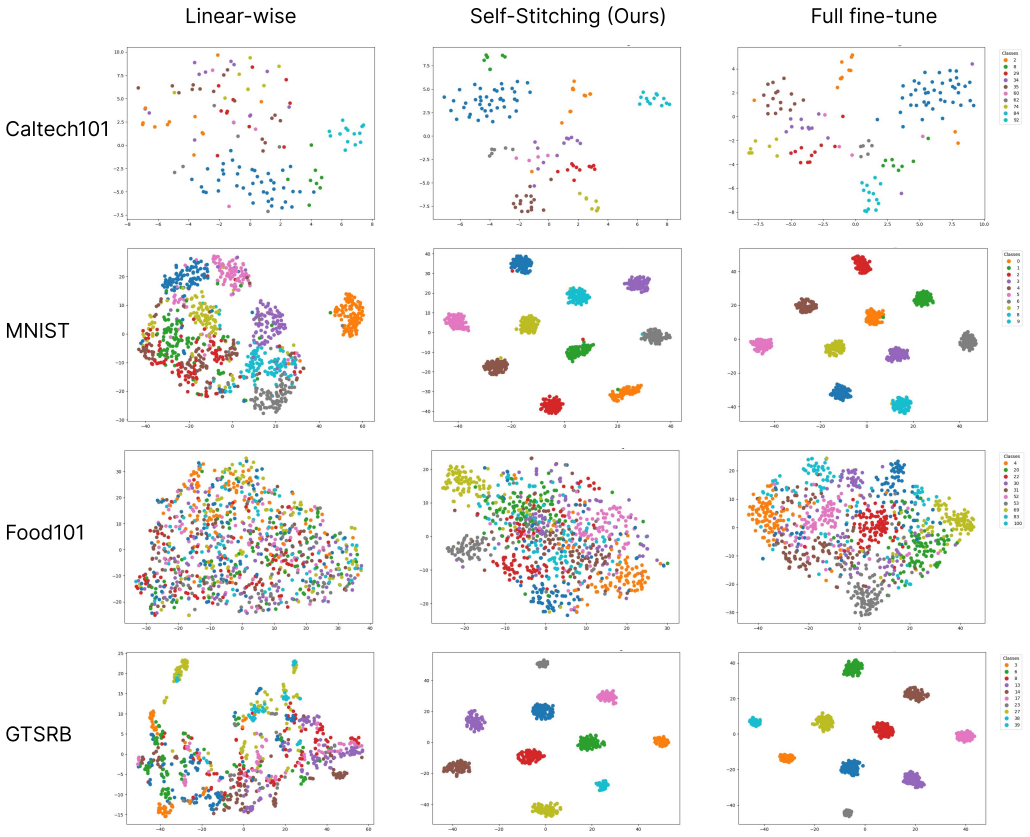


Figure 4: Visualization of the feature space from cross-domain transfer learning experiment.

This visualization confirms that Self-Stitching improves feature extractor adaptability with fewer trainable parameters, especially in scenarios involving larger domain gaps like those seen with Food101 and GTSRB.

Does increasing stitching position result in full fine-tuning accuracy? In this section, we investigate the behavior of the performance of the target model when we increase stitching layers throughout the model and fine-tune it based on the target data. As we compare our stitched model with the fully fine-tuned model, the question is whether increasing the number of extra stitching layers over the pre-trained model contributes to the increase in performance that would converge to the performance of full fine-tuning.

The experimental results from table 5 demonstrate that the performance of the stitched models on CIFAR100 varies depending on the number of layers stitched. When only one layer is stitched, performance rates vary widely from 40% to 46.70%. Introducing stitching at two layers leads to an average performance, as indicated by a 45% accuracy score. Adding stitching to three or four layers seems to offer more stability, consistently yielding around 46% performance. In contrast, models with two stitched layers exhibit more performance fluctuation, ranging from 42% to 46%. Notably, as the number of stitched layers increases, the model's performance progressively aligns more closely with that of a fully fine-tuned model.

Weakly supervised setting with noisy label Table 6 presents an experiment evaluating the robustness of transfer learning methods under various label noise conditions. The models are trained on one 50% subset of the CIFAR100 dataset and tested on a different 50% subset, with noise levels set at 0.1, 0.2, and 0.3. The experiment assesses the performance of pre-trained baseline, linear-wise and

Table 5: The accuracy of the model when increasing the number of stitching layers at different stitching index

Stitching Index	Fine-tuning accuracy
1	40.28
2	42.22
3	46.70
4	44.42
1,2	42.62
1,3	45.16
1,4	45.84
2,3	45.48
2,4	46.43
3,4	44.40
1,2,3	46.38
1,2,4	46.12
1,3,4	46.60
2,3,4	45.98
1,2,3,4	46.40
Full fine-tune	47.26

Table 6: Transfer learning performance with various levels of noise

	50% of classes	Other 50% of classes			
Noise Level	Pre-trained	Linear	Cosine	Self-Stitching	Full fine-tune
0.1	40.66	32.52	26.88	<u>37.00</u>	**37.50**
0.2	32.18	25.16	21.12	<u>27.60</u>	**28.80**
0.3	21.4	21.38	18.90	**23.82**	<u>23.58</u>

cosine-wise transfer learning method, our proposed Self-Stitching method, and the full fine-tuning approach under these conditions, highlighting their strengths and weaknesses in handling noisy data.

As shown in table 6, the results demonstrate the efficacy of Self-Stitching as a transfer learning strategy, particularly in noisy data conditions. As the noise level increases from 0.1 to 0.3, traditional methods like linear and cosine similarity approaches show a significant decline in performance. However, the Self-Stitching method consistently outperforms other strategies, indicating its robustness and potential as a reliable method in noisy environments. While stitching cannot outperform full fine-tuning, the performance of full fine-tuning is on par with the performance of the stitching layer.

5.2 Self-Stitching's variants

Adding ReLU after a stitching layer In our experiments, we investigated the impact of incorporating an activation function, specifically ReLU, after the stitching layer within the model. Since normal Self-Stitching introduces a linear convolutional layer, the introduction of ReLU was intended to inject non-linearity post-stitching, investigating the necessity of the activation function. The results show that the ReLU activation function doesn't contribute to the improvement of the performance of the model.

Parallel Stitching Instead of directly inserting the stitching layer, we also explored an alternative approach where the stitching layer operates alongside a specific stage of the original ResNet model, akin to creating a bypass route that merges with the main flow at a subsequent stage. This method, named 'Parallel Stitching' was designed to assess the impact of parallel stitching on the model's performance and to determine whether it could offer any advantages over the traditional stitching approach. The results from these experiments are presented in table 7. The parallel stitching method slightly outperforms the traditional stitching method when adding a stitching layer only in the earlier stage.

Table 7: Accuracy of Self-Stitching with and without ReLU activation function and Parallel Stitching

Stitching Index	Without ReLU	With ReLU	Parallel Stitching
1	41.18	41.50	**41.96**
2	42.64	41.96	**43.94**
3	**44.52**	43.14	44.10
4	**43.66**	43.02	40.36

5.3 Transfer learning efficiency

The Self-Stitching method significantly reduces the number of trainable parameters compared to full fine-tuning, yet achieves similar or better performance across wide domain ranges and data sizes. According to table 8, Self-Stitching requires only 30,066 to 115,6146 parameters for ResNet-18, ResNet-34, and ResNet-50 models, compared to the 11,202,162 to 23,610,482 parameters needed for full fine-tuning. Though Self-Stitching uses more parameters than linear and cosine transfer learning methods, Self-stitching consistently outperforms them. This reduction in parameters leads to more efficient use of computational resources, faster training times, and lower memory requirements, without compromising model performance.

Table 8: Parameter count comparison for different transfer learning strategies

Method		Time [order]	Trainable Parameter [-]		
			ResNet-18	ResNet-34	ResNet-50
Linear		1	25,650	25,650	102,450
Cosine		1	25,650	25,650	102,450
Self-Stitching at specific stitching index	1	1.3	30,066	30,066	106,866
	2	1.3	30,066	30,066	169,266
	3	1.3	42,674	42,674	367,154
	4	1.3	92,466	92,466	1,156,146
Full fine-tune		6.5	11,202,162	21,310,322	23,610,482