
Deep Transfer Learning with Ridge Regression

^αShuai Tang ^{αβ}Virginia R. de Sa

^αDepartment of Cognitive Science, ^βHalicioğlu Data Science Institute
University of California, San Diego
{shuaitang93,desa}@ucsd.edu

Abstract

The large amount of online data and vast array of computing resources enable current researchers in both industry and academia to employ the power of deep learning with neural networks. While deep models trained with massive amounts of data demonstrate promising generalisation ability on unseen data from relevant domains, the computational cost of finetuning gradually becomes a bottleneck in transferring the learning to new domains. We address this issue by leveraging the low-rank property of learnt feature vectors produced from deep neural networks (DNNs) with the closed-form solution provided in kernel ridge regression (KRR). This frees transfer learning from finetuning and replaces it with an ensemble of linear systems with many fewer hyperparameters. Our method is successful on supervised and semi-supervised transfer learning tasks.

1 Introduction

Besides soaring high on tasks they are trained on, deep neural networks have also excelled on tasks where datasets are collected from similar domains. Prior work [34] showed that filters/parameters learnt in DNNs pretrained on ImageNet generalise better with slight finetuning than those learnt from random initialisations. Since then, applications in Computer Vision have had major breakthroughs by initialising DNNs with pretrained parameters and finetuning them to adapt to new tasks. Similarly, Natural Language Processing (NLP) welcomed its “ImageNet Era” with large and deep pretrained language models including Bert [11], and performance on downstream NLP tasks has achieved state-of-the-art on a daily basis by employing more data and deeper models during pretraining, and using smarter methods for finetuning.

These advances in transfer learning using pretrained DNNs and finetuning, however, come with a large computational cost. An essential step to boost the performance on a given new task is to finetune the pretrained DNN until it converges, which is computationally intense since these models tend to have hundreds of millions of parameters. An alternative approach is to freeze the parameters and treat the pretrained DNN model as a feature extractor which produces abstracted vector representations of data samples with the knowledge from pretraining, and then train a simple classifier on top of these extracted vectors. But as the parameters are not adapted to the new task, the latter approach provides inferior performance to finetuning.

We here propose a new way of augmenting the latter approach without finetuning the DNN. Our approach is to take an accumulation of feature vectors produced at different individual layers which encode various different aspects of the data. Since feature vectors are highly correlated with each other, as they are generated from a single DNN, only a few of them are needed to make predictions. We adopt the alignment maximisation algorithm for combining kernels [8], in which we first find a convex combination of linear kernels constructed from individual layers that gives maximal alignment with the target kernel constructed from one-hot encoding of the labels. Then, we take the ensemble of feature vectors of layers selected by non-zero elements in the sparse combination, and make predictions using kernel ridge regression (KRR).

2 Related Work

Transfer learning with classical machine learning methods has been studied for a couple of decades [27], including boosting [9], support vector machines [24], ridge regression [7], etc. These methods benefit from the transparency of classical machine learning models, and universal function approximators including boosting and kernel methods with strong theoretical guarantees. However, it is not easy to incorporate structural priors into regularising the learning process, such as our knowledge about images and text. This information is crucial in advancing machine learning systems.

Neural networks are also universal function approximators [20], and learnt vectorised representations are generalisable across tasks, with recent advances in various architecture designs specifically for individual types of inputs, including convolutional layers for image recognition [23], recurrent layers [12, 19] and transformers [31] for text processing, etc. Recent research has demonstrated that deep models pretrained on large amounts of training data give decent performance on unseen data sampled from relevant domains [34] by finetuning. With growing depth of networks, the cost for finetuning becomes non-negligible. Efforts in knowledge distillation from deep models to shallow ones [1, 18] and to simple ones [13] showed that neural networks can be simplified after learning, although the learnt transferable features can be potentially detrimented during distillation.

Our approach takes the best of both worlds by using feature vectors produced from multiple layers of a pretrained neural network but without explicit finetuning, and makes predictions with KRRs on a downstream task. With help from low-rank approximations, our approach only requires passing the training data once through a neural network without backpropagation.

3 Method

The key concept is to apply KRR with a few layers of feature vectors produced from a pretrained neural network to make predictions, classification in our case, on a downstream task. The notations include: $\mathbf{X} \in \mathbb{R}^{N \times d}$ is the data matrix with N samples with each sample in d -dimensional space, $\mathbf{Y} \in \mathbb{R}^{N \times c}$ is the corresponding labels with one-hot encoding, $\mathbf{X}_l \in \mathbb{R}^{N \times d_l}$ is the flattened feature vectors produced at the l -th layer from a pretrained neural network, $\mathbf{S} \in \mathbb{R}^{M \times N}$ is the random projection matrix that meets the requirement of subspace embedding with $M \ll N$, $\mathbf{I}_p \in \mathbb{R}^{p \times p}$ is the identity matrix, L is the number of layers in a pretrained neural network, and α is the regularisation term in ridge regression. Other notations will be introduced as needed.

3.1 Low-rank Approximation at Individual Layers

Flattened feature vectors generated from neural networks are generally high-dimensional and redundant, therefore, we adopt theoretical work [30] showing that big data matrices are approximately low rank, and use random projections to obtain low-rank approximations of high-dimensional feature vectors with many fewer dimensions. Given that the Nyström method is well-studied in approximating large-scale kernel matrices [14], we follow the formula to approximate a linear kernel $\mathbf{X}_l \mathbf{X}_l^\top$ as

$$\mathbf{X}_l (\mathbf{S} \mathbf{X}_l)^\top (\mathbf{S} \mathbf{X}_l \mathbf{X}_l^\top \mathbf{S}^\top)^\dagger (\mathbf{S} \mathbf{X}_l) \mathbf{X}_l^\top \quad (1)$$

where \dagger is the pseudo-inverse of a square matrix. If $M \ll d$, which is mostly the case for feature vectors generated from neural networks, and the eigendecomposition is written as $(\mathbf{S} \mathbf{X}_l \mathbf{X}_l^\top \mathbf{S}^\top)^\dagger = \mathbf{Q}_l \Lambda_l \mathbf{Q}_l^\top$, then the low-rank approximation of $\tilde{\mathbf{X}}$ can be obtained by $\mathbf{X}_l (\mathbf{S} \mathbf{X}_l)^\top \mathbf{Q}_l \Lambda_l^{-0.5}$ with each sample in at most M -dimensional space. As we aim to conduct layer-wise low-rank approximations, it is preferable to apply sparse random projections instead of dense ones. Therefore, we consider a stack of s CountSketch [4] to approximate the sparse Johnson-Lindenstrauss Transformation [33].

In CountSketch, the random projection matrix \mathbf{S} is considered as a hash table that uniformly hashes N samples into M buckets with a binary value randomly sampled from $\{+1, -1\}$ so there is no need to materialise \mathbf{S} . Successful applications of CountSketch including polynomial kernel approximation [28] and large-scale regressions are due to its scalability with theoretical guarantees when few hash tables are used [33]. Generally, larger s leads to better approximations, yet the performance improvement becomes marginal. Prior work [21] showed that $s = 4$ empirically works on real-world datasets, thus, we set $s = 4$, and it drastically reduces the cost for low-rank approximations at individual layers. The time complexity of Nyström is $\mathcal{O}((sN + M^2 + NM)d_l + (M + N)M^2)$.

With limited GPU memory, producing feature vectors for a downstream task given a pretrained neural network is often done in batches of samples. CountSketch is also well-suited in this situation as, technically, the approximation can be done in only one forward pass of \mathbf{X} .

3.2 Convex Combination of Features across Layers by Learning Kernel Alignment

Storing feature vectors at L layers has memory complexity at most $\mathcal{O}(NML)$, thus we aim to select only a few layers that give the maximum alignment with the target. Specifically, a vector $\boldsymbol{\mu} = [\mu_1, \mu_2, \dots, \mu_L]^\top$ is optimised to maximise the following alignment [8]:

$$\boldsymbol{\mu}^* = \arg \max_{\{\|\boldsymbol{\mu}\|=1, \mu \geq 0\}} \frac{\langle \mathbf{K}, \mathbf{Y}\mathbf{Y}^\top \rangle_F}{\|\mathbf{K}\|_F}, \text{ where } \mathbf{K} = \sum_{l=1}^L \mu_l \tilde{\mathbf{X}}_l \tilde{\mathbf{X}}_l^\top \quad (2)$$

Proposition 9 in [8] showed that it is equivalent to the quadratic programming problem: $\mathbf{v}^* = \arg \max_{\mathbf{v} \geq 0} \mathbf{v}^\top \mathbf{M} \mathbf{v} - 2\mathbf{v}^\top \mathbf{a}$, where $\mathbf{a}_l = \|\tilde{\mathbf{X}}_l^\top \mathbf{Y}\|_F^2$ and $\mathbf{M}_{k,l} = \|\tilde{\mathbf{X}}_k^\top \tilde{\mathbf{X}}_l\|_F^2$, then $\boldsymbol{\mu}^* = \mathbf{v}^* / \|\mathbf{v}^*\|$. Intuitively, Non-zero entries in $\boldsymbol{\mu}^*$ provide a weighted sparse combination of feature vectors from a few layers that gives the highest linear alignment with targets. The time complexity is dominated by materialising $\mathbf{M} \in \mathbb{R}^{L \times L}$, which is $\mathcal{O}(L^2 M^2 N)$ at worst.

The kernel \mathbf{K} induces an embedding space which is a concatenation of feature vectors weighted by $\mu_l^{1/2}$, then the optimisation problem in Eq. 2 can be written in a weight-space perspective:

$$\boldsymbol{\mu}^* = \arg \max_{\{\|\boldsymbol{\mu}\|=1, \mu \geq 0\}} \frac{\|\mathbf{X}_\phi^\top \mathbf{Y}\|_F^2}{\|\mathbf{X}_\phi^\top \mathbf{X}_\phi\|_F}, \text{ where } \mathbf{X}_\phi = [\mu_1^{\frac{1}{2}} \tilde{\mathbf{X}}_1, \mu_2^{\frac{1}{2}} \tilde{\mathbf{X}}_2, \dots, \mu_L^{\frac{1}{2}} \tilde{\mathbf{X}}_L] \quad (3)$$

It is worth noting that the objective is not the ‘‘goodness-of-fit’’ measure for linear regression, R^2 statistics $\|\mathbf{X}_\phi^\top \mathbf{Q}_Y\|_F^2 / \|\mathbf{X}_\phi\|_F^2$, where \mathbf{Q}_Y contains eigenvectors of $\mathbf{Y}\mathbf{Y}^\top$. Optimising $\boldsymbol{\mu}$ to maximise R^2 will lead to drastic overfitting by accumulating all layers, and subsequently meaningless $\boldsymbol{\mu}$. The aforementioned objective finds a convex combination of features that maximises the alignment between the subspace spanned by the concatenated features and that by the onehot encoded label space, so it prevents \mathbf{X}_ϕ from accumulating more feature vectors once an optimal subset is obtained. Therefore, the alignment-based objective prevents overfitting to a certain degree.

3.3 [Optional Step] Nyström for Large-scale Kernel Approximation

We denote L_s as the number of layers with positive μ_l ’s. Since, in the end, the predictions are made by kernel ridge regression, if N is at a manageable order, then there is no need to conduct kernel approximation through Nyström. However, low-rank approximation can potentially help reduce the noise in data, which leads to a better generalisation compared to computing the exact kernel function.

We consider approximating an RBF kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2)$ with the Nyström method using the same subsampling in Sec. 3.1, CountSketch, to further promote fast computation on accumulated feature vectors \mathbf{X}_ϕ . We denote the number of buckets in m hash functions as M_s , then the time complexity of this step is $\mathcal{O}((m + M_s)NM + 2M_s^2 M + M_s^3)$. Since $N \gg M$ and $N \gg M_s$, the dominating term in the complexity is $M_s NM$. The hyperparameter σ^2 is heuristically set to $\max\{\|\mathbf{x}_i\|^2 / 2 : i = 1, 2, \dots, N\}$. One could cross-validate σ^2 as well, however, for the sake of reducing of the complexity of transfer learning, we stick to the heuristic value.

3.4 Ridge Regression for Predictions

The approximated low-rank feature map of an RBF function is denoted as $\tilde{\mathbf{X}}_\psi \in \mathbb{R}^{N \times M_s}$. Given a new data sample $\tilde{\mathbf{x}}_\psi$, the prediction is given the closed-form solution of ridge regression in the table. Then

condition	$N \gg M_s$	$N \ll M_s$
prediction \mathbf{y}	$\tilde{\mathbf{x}}_\psi (\tilde{\mathbf{X}}_\psi^\top \tilde{\mathbf{X}}_\psi + \alpha \mathbf{I}_{M_s})^{-1} \tilde{\mathbf{X}}_\psi^\top \mathbf{Y}$	$\tilde{\mathbf{x}}_\psi \tilde{\mathbf{X}}_\psi^\top (\tilde{\mathbf{X}}_\psi \tilde{\mathbf{X}}_\psi^\top + \alpha \mathbf{I}_N)^{-1} \mathbf{Y}$

the label of a test sample is the index of the maximum value in predicted \mathbf{y} . The time complexity of ridge regression is determined by the inverse of a square matrix and the matrix multiplication that gives the square matrix, and it is $\min(N^3 + M_s N^2, M_s^3 + N M_s^2)$.

Table 1: **Dataset details.** Individual cell indicates (# Training Samples / # Test Samples / [# Classes]).

Training ImageNet	In-domain Transfer			Out-of-domain Transfer		
	CIFAR10	CIFAR100	STL10	SVHN	CUB200	Kuzushiji49
1.2m / - [1000]	50k / 10k [10]	50k / 10k [100]	5k / 8k [10]	73k / 26k [10]	6k / 6k [200]	116k / 38k [49]

Table 2: **Results of supervised transfer learning.** Median accuracy of five trials is reported in each cell, and each cell has two accuracy terms of transferring from [ResNet-18 / ResNet-34]. Expect for CUB200, our method outperforms LogReg significantly since the variance of five trials is very small as presented in figures.

	CIFAR10 [In]	CIFAR100 [In]	STL10 [In]	CUB200 [Out]	SVHN [Out]	Kuzushiji49 [Out]
LogReg	87.45 / 89.94	69.08 / 72.76	95.08 / 96.55	60.80 / 61.60	64.36 / 59.47	74.56 / 71.08
Ours	90.77 / 92.31	71.31 / 74.63	96.30 / 97.31	58.78 / 61.70	88.76 / 88.53	88.12 / 88.00

In summary, our proposed method has four steps including 1) **CountSketch** to obtain low-rank feature vectors at individual layers to a manageable size, 2) **convex combination** to take weighted accumulation of feature vectors, 3) **Nyström** for approximating an RBF kernel, and 4) **KRR** to make predictions. Compared to multiple forward and backward passes required in finetuning or training classifiers, our method drastically reduces the computational cost.

4 Experiments

We demonstrate the effectiveness of our method through experiments on transferring ResNet-based models [16, 17] pretrained on the ImageNet dataset [10, 29] to downstream tasks, including three in-domain datasets, CIFAR-10, CIFAR-100 [22], STL10 [5], and three out-of-domain ones, Street View House Number (SVHN) [25], Caltech-UCSD-200 (CUB200) [32], Kuzushiji49 [3]¹. Basic statistics of each dataset are presented in Table 1.

Hyperparameter Settings: We report results with $M = \{512, 1024, 2048\}$, and $M_s = 2M$. ResNet-18 and ResNet-34 pretrained on ImageNet are selected as base models to transfer from. To reduce the memory cost, instead of hashing all layers, we only hash feature vectors from every residual block in a model as each block usually has two or three convolutional layers. The regularisation strength α is cross-validated on the training set of the downstream task with values ranging from $\{1e-1, 1e-2, 1e-3, 1e-4\}$.

Comparison Partner: Finetuning the top layer on each downstream task with softmax regression. Models are finetuned for 30 epochs with Adam optimiser, and the learning rate decays by a factor of 2 every 10 epochs. Cross validation is conducted to optimise the following hyperparameters and their associated values: data augmentation= $\{\text{with}, \text{without}\}$, weight decay rate= $\{1e-3, 1e-4, 1e-5\}$, initial learning rate= $\{1e-3, 2.5e-4\}$. Note that finetuning with data augmentation tremendously increases the training time as the neural network needs to be kept during finetuning, while for others one can store feature vectors from the last layer prior to finetuning. Results are marked with **LogReg** in the following tables and figures.

Trials: Since our method involves random projections and comparison partners require initialisation, for fair comparison, we run each method five times with different random seeds, and each marker in each plot presents the mean of five trials along with a vertical bar indicating the standard deviation. It is noticeable that vertical bars are often invisible as hyperparameters of each method are cross-validated on the training set. The main results are presented in Tab. 2.

4.1 Supervised Transfer with Varying Portions of Training Samples

Since individual downstream tasks have ample samples in the training set, it encourages us to study our method and its comparison method when varying the portion of training samples. Specifically, the

¹The full Kuzushiji49 dataset has 232k training images, which takes too long to cross-validate hyperparameters for LogReg. Thus, the same half of the dataset is used.

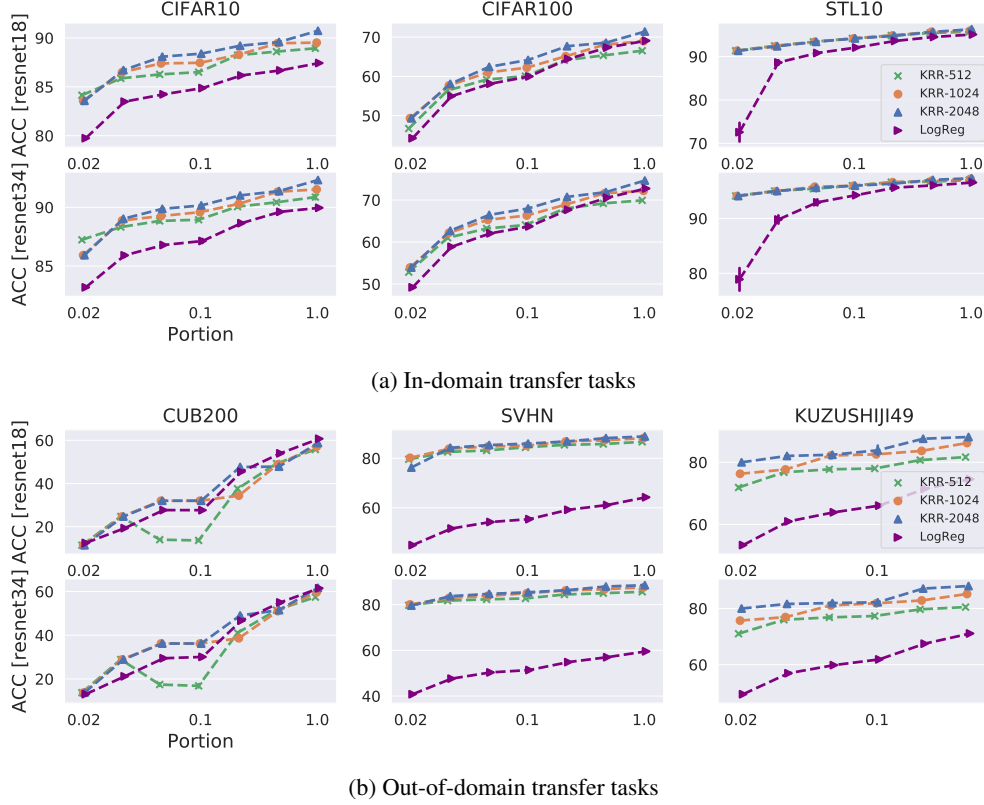


Figure 1: **Supervised transfer with varying portions of training samples from the transfer task.** Except for CUB200, our method with all three M 's generalises better than LogReg does (purple lines in plots) when the portion of training samples varies from 2% to 100%, and the observation is consistent across two different depths of ImageNet models.

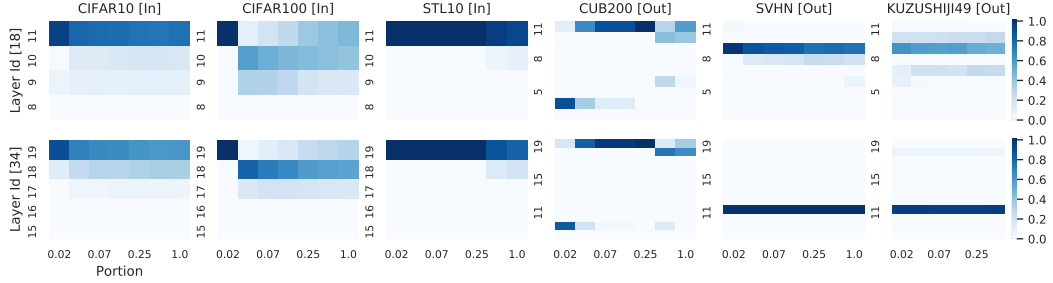


Figure 2: **Convex combination μ of layers vs. Varying portions of training samples.** In-domain transfer tasks assign higher values to top few layers, and out-of-domain ones tend to give higher values to slightly lower layers.

kept portion of training samples varies from 2% to 100%, and the interval is determined linearly in the log-space. The results are presented in Fig. 1.

Our method for $M = \{1024, 2048\}$ outperforms significantly finetuning the top layer on five out of six transfer tasks with different portions of training samples, and only performs relatively similar to finetuning on CUB200, which is a finegrained bird species recognition task.

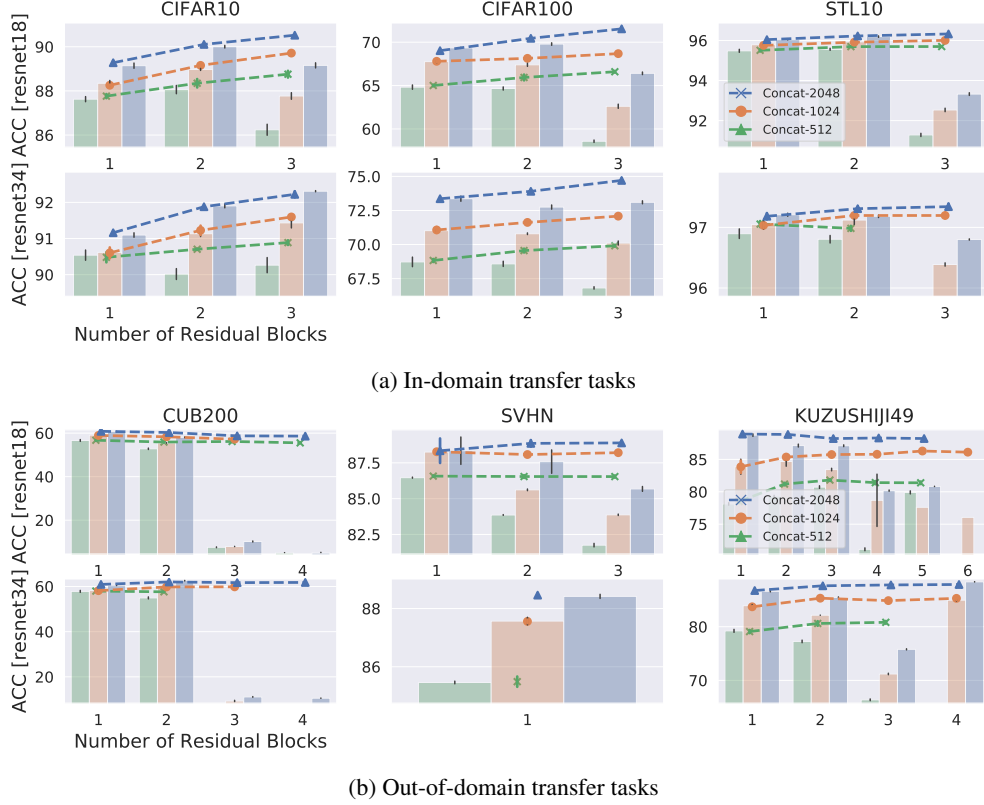


Figure 3: **Accuracy of accumulating Residual Blocks vs. that of individual Ones.** Line plots indicate accuracy of accumulating blocks until the exhaustion of non-zero μ_l , and bar plots indicates the performance of these blocks separately. **(I)** In-domain transfer tasks demonstrate increasing accuracy when blocks are accumulated gradually, so does Kuzushiji49, which validates that accumulating layers helps. **(II)** Note that values $\{\mu_l\}_{l=1}^L$ don't directly imply the importance of layers, and that explains why the bar plots don't have a monotonic trend.

4.2 Insights provided by μ_l

The solution to Eq. 2 provides insights on the number of accumulated layers and their weights. We plot a heatmap with y-axis indicating the index of layers, x-axis indicating the portion of training samples, and gradient colour scheme presenting the value of μ_l in Fig. 2.

As shown in [17], the penultimate layer (index 11 for resnet18 and 19 for resnet34) of a ResNet removes all spatial information by averaging outputs from the previous layer. As illustrated in Fig. 2, layers before the penultimate layer have been assigned non-zero μ_l 's across six tasks confirming that preserving spatial information helps in transfer learning.

For in-domain transfer tasks, it turns out that the top few layers are the most useful, and the improvement of our method is brought by the ability of identifying and accumulating these layers. STL10 contains images from the ImageNet dataset but with lower resolutions, so the penultimate layer provides adequately abstract information of the images, which explains the observation that our method assigns a very dominating μ towards the penultimate layer.

For SVHN and Kuzushiji49, clearly, the selected layers don't include the feature vectors generated from the penultimate layer, and lower layers give higher μ , which results in better performance than finetuning the top linear layer. However, our method doesn't provide better performance compared to finetuning the last layer on CUB200. A potential explanation comes from the fact that kernel ridge regression learns one-vs-all classifiers, and it is suitable when many classes are presented. This is a limitation of our method, but also a research direction for future study.

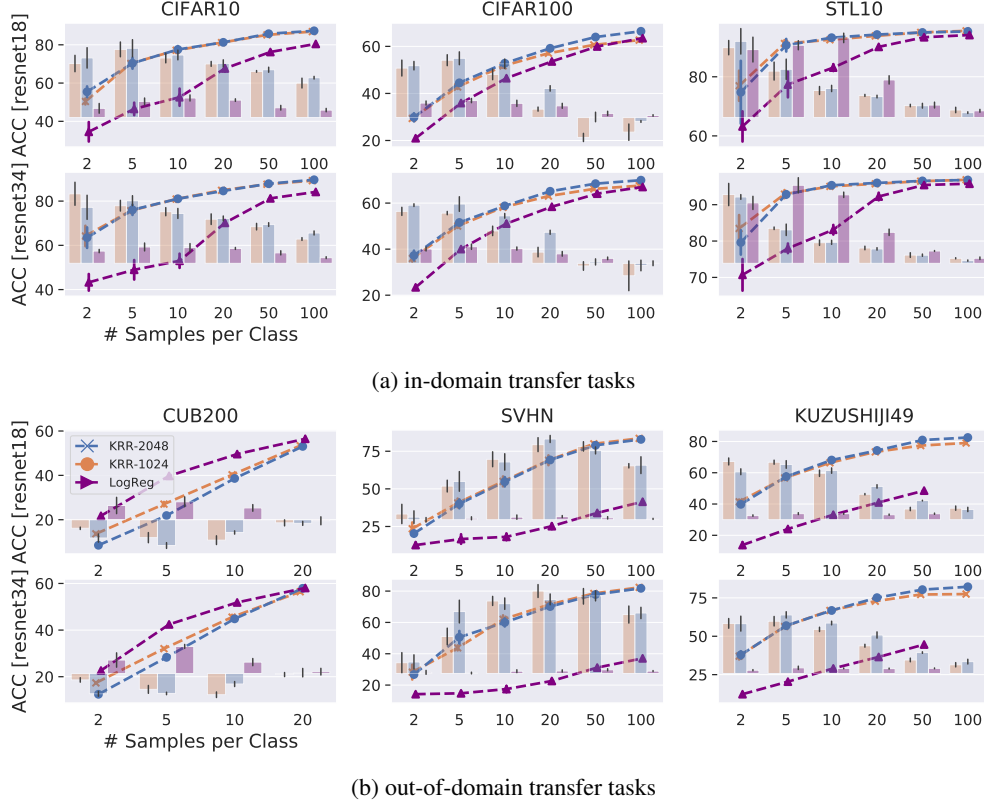


Figure 4: **Accuracy of semi-supervised learning with varying number of labelled samples per class.** The number of labelled examples changes from 1 to 100 per class and the rest are left unlabelled for semi-supervised learning. Left y-axis for line plots refers to the accuracy of semi-supervised learning, and right y-axis for bar plots refers to the relative improvement brought by unlabelled data. **(I)** Our method gives better performance than LogReg overall expect for CUB200. **(II)** Our method is also better at leveraging unlabelled samples for learning as indicated by taller bars for ours than LogReg expect for STL10 and CUB200.

4.3 Accumulated Feature Vectors vs Individual Feature Vectors

As \mathbf{X}_ϕ in our method is a weighted concatenation of feature vectors from layers with non-zero μ_l 's, it is important to conduct a sanity check on the effectiveness of accumulating layers compared to using these layers alone. Therefore, we gradually accumulate layers sorted by their μ_l 's, and plot the performance curve versus the number of accumulated layers. Then these layers are applied individually to make predictions as a comparison. We use the full training dataset in this subsection. The results are shown in Fig. 3.

For in-domain transfer tasks, we see that the performance improves as our method accumulates layers, while the trend is not obvious/significant for out-of-domain transfer tasks. Overall, accumulating a few layers provides better performance than making predictions based on individual layers.

4.4 Semi-supervised Transfer Learning via Transductive Regression

There are many ways of incorporating unlabelled data into kernel ridge regression, including manifold regularisation [2] and transductive learning [6]. Since manifold regularisation requires exact computation or an approximation of the Laplacian matrix on labelled and unlabelled samples, which leads to increased learning time, we adopted the transductive learning method for regression problems to leverage unlabelled data when extremely limited labelled training samples $\{\tilde{\mathbf{X}}_\psi, \mathbf{Y}\}$ with large amount of unlabelled samples $\{\tilde{\mathbf{X}}'_\psi\}$ are provided. The solution of transductive ridge regression [6] is

$$\mathbf{W} = (\beta' \tilde{\mathbf{X}}_\psi'^\top \tilde{\mathbf{X}}'_\psi + \beta \tilde{\mathbf{X}}_\psi^\top \tilde{\mathbf{X}}_\psi + \mathbf{I}_{M_s})^{-1} (\beta' \tilde{\mathbf{X}}_\psi'^\top \mathbf{Y}' + \beta \tilde{\mathbf{X}}_\psi^\top \mathbf{Y}) \quad (4)$$

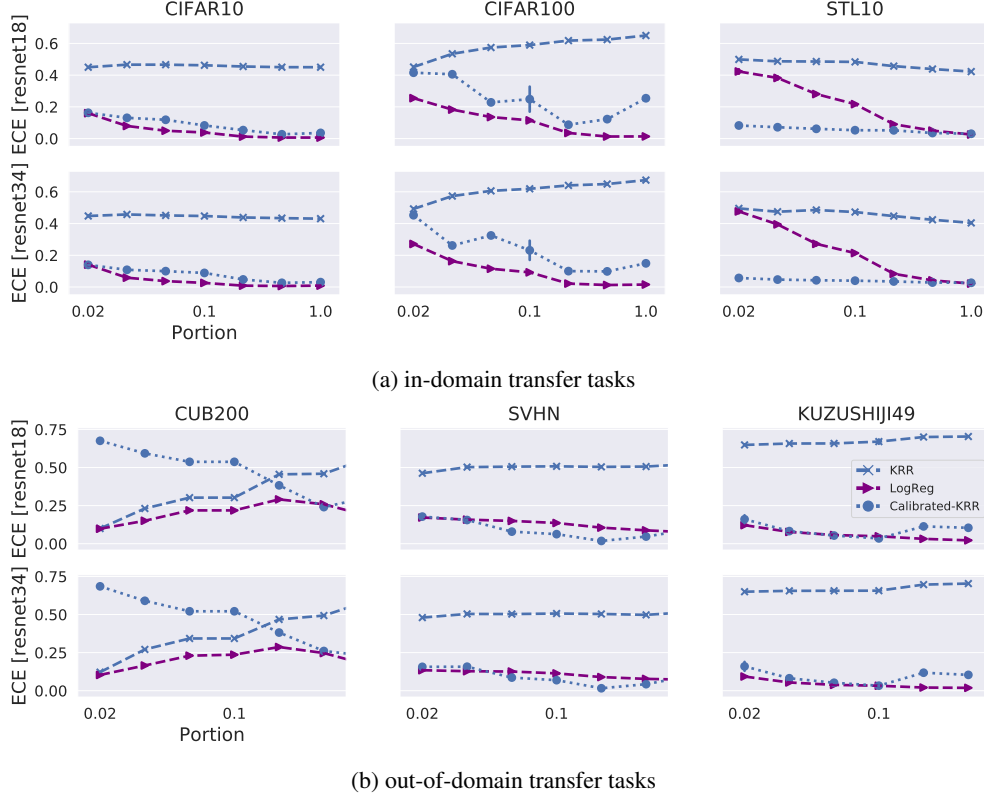


Figure 5: **Expected Calibration Error (lower the better) with varying portions of training data.** LogReg provides better calibrated models than ours does. However, a posthoc adjustment by *Temperature Scaling* helps our method to match the calibration performance with LogReg.

where β' and β are hyperparameters that control the contribution from unlabelled data and labelled data, which can be cross-validated on the labelled data. \mathbf{Y}' comes from the ridge regression model learnt only on labelled data, and is given as $\mathbf{Y}' = g(\tilde{\mathbf{X}}_\psi(\tilde{\mathbf{X}}_\psi^\top \tilde{\mathbf{X}}_\psi + \alpha \mathbf{I}_{M_s})^{-1} \tilde{\mathbf{X}}_\psi^\top \mathbf{Y})$, where α is a hyperparameter, and $g(\cdot)$ sets the maximum value of the vector prediction of a data sample to 1 and the rest to 0. For finetuning the top layer, we use the supervised classifier trained on labelled samples to annotate unlabelled data samples, and incorporate these samples into the training set and retrain the classifier with cross-validation.

We simulate a semi-supervised learning environment by keeping 2, 5, 10, 20, 50, or 100 labeled training samples per class on each dataset, and leave the rest as unlabelled samples. The accuracy of semi-supervised transfer learning on the testset is reported in lineplots in Fig. 4, and the relative improvement against supervised transfer learning is reported in barplots in the same figure.

Our method outperforms finetuning the last layer on five out of six transfer tasks indicated by the lineplots. Our method also gives significant relative improvement when unlabelled samples are incorporated through transductive regression on these five tasks as well, while unlabelled samples don't improve the generalisation ability for the top layer finetuning method. Negative results are concentrated on CUB200, where unlabelled samples become detrimental to our method while helpful for finetuning the top layer.

5 Discussion

5.1 Temperature Scaling for Calibration

Although our method provides both speed up and accuracy improvement in transfer learning, we are also interested in how well-calibrated our learnt classifier is compared to finetuning the top layer. It is

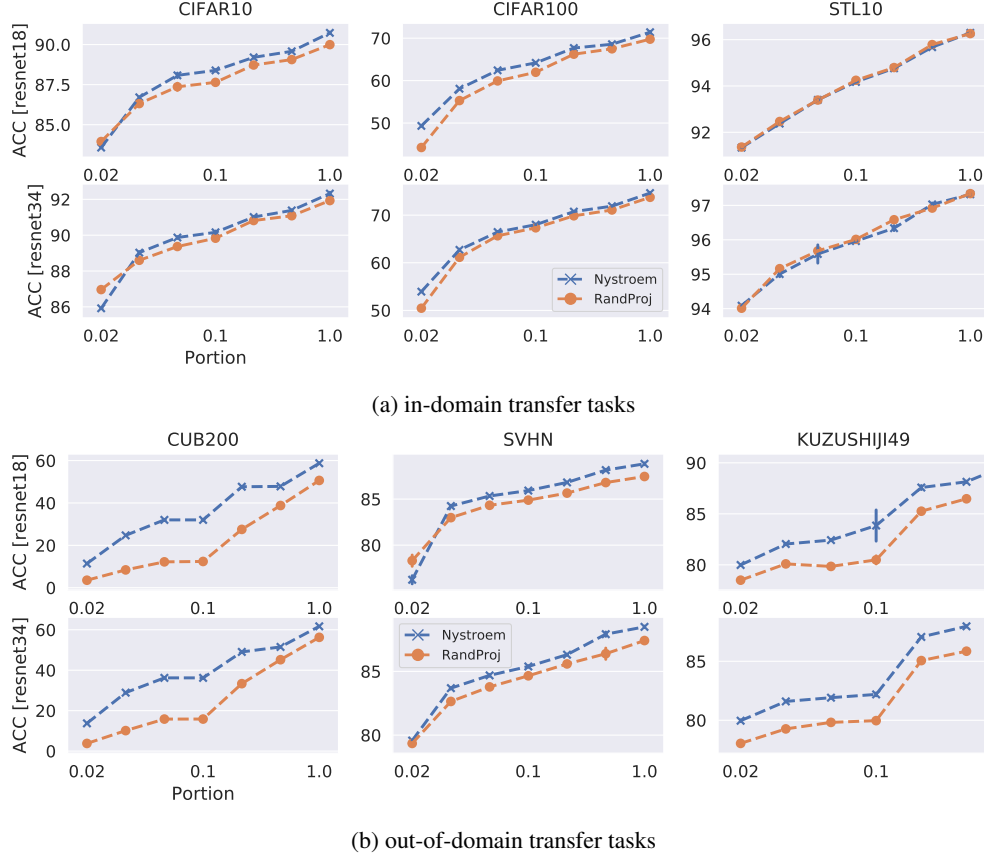


Figure 6: **Nyström vs. Random Projection in the first step of our method.** The performance improvement of Nyström over random projection is relatively larger on out-of-domain transfer tasks than it is on in-domain ones, and the observation is consistent across varying portions of training data.

expected that KRR, SVM and tree-based boosted classifiers are not well-calibrated as the predicted outputs can not be directly interpreted as classifiers’ confidence [26]. We calculate the Expected Calibration Error (ECE) [15] for our method, and baseline models - finetuning the top layer with logistic regression. The formula of ECE is given as $ECE = \sum_{c=1}^C \frac{B_c}{N} |\text{acc}(B_c) - \text{conf}(B_c)|$, where N is the number of samples, and C is the number of bins in the estimation.

The results shown in Fig. 5 validate our expectation that our method gives worse calibration on test set compared to logistic regression. A simple cure is Temperature Scaling [15], which optimises a parameter t to rescale the output from the classifier in order to reduce the ECE on training data, and doesn’t change the predicted labels. In our case, we can simply cross-validate t very efficiently once the output from KRR is produced.

5.2 Random Projections

Our method adopted the Nyström method for low-rank approximation of feature vectors at individual layers, which involves hashing N data samples into M buckets first, then solving a linear system, and the hash functions can be applied across all layers. A more direct approach is to hash individual features in each feature vector into M buckets as in $X_i S_i \in \mathbb{R}^{N \times M}$. This approach eliminates the step of solving a linear system, which reduces the time complexity to $\mathcal{O}(sM d_i)$ for each layer. The comparison between Nyström and random projection is presented in Fig. 6.

Overall, Nyström provides better accuracy across all six tasks than random projection. However, it is noticeable that the difference between the two is smaller on in-domain transfer tasks. The observation also serves as a piece of supporting evidence that our method is relatively consistent when different low-rank approximation schemes are applied in the first step.

5.3 Task-dependent Distillation

Our method in previous sections still requires the pretrained ImageNet model during testing time. Now we present task-dependent distillation to leverage the predictions constructed in our method as a regularisation in training smaller networks for individual downstream tasks.

Table 3: Predictions from our method as a regularisation for training small models on individual tasks. “w/o” refers to training models with cross-entropy loss only and “w” refers to training with MSE loss as a regulariser. The median performance of five random initialisations is reported. Overall, predictions from our method helps smaller models to generalise better.

	ResNet 8 / ResNet 10		
	CIFAR10	CIFAR100	STL10
w/o	91.50 / 92.22	69.24 / 69.41	67.31 / 69.21
w	92.23 / 93.22	69.03 / 70.87	69.91 / 71.64

Once predictions on the training set of a task are made by KRR in our method, we store the predictions and remove the pretrained model. This step only requires memory complexity of $\mathcal{O}(NC)$, where C is the number of classes.

We train a ResNet with 8 layers and one with 10 layers on individual tasks with the cross-entropy loss, and use Mean Squared Error loss (MSE) to regress the output of neural networks to the predictions made by KRR from our method. The results are presented in Tab. 3. Overall, regularising small models with predictions of our methods from ImageNet models helps them to obtain better generalisation.

6 Conclusion

We provided a promising four-step Ridge Regression based transfer learning scheme for deep learning models. It doesn’t require finetuning, which simplifies the transfer learning problem to simple regressions, and it is capable of identifying a few layers to accumulate for making better predictions.

We evaluated our method on supervised transfer with varying portions of training data, and handle semi-supervised transfer learning problems via transductive regression. Both show significant improvement compared to finetuning only the last layer.

Discussions addressed the issue of calibration by Temperature Scaling, and demonstrate the superiority of Nyström over plain random projections. Lastly, we showed that the predictions from our method can be used to improve shallow/small models when training directly on transfer tasks.

Acknowledgements

Shuai Tang and Virginia R. de Sa are supported by NSF IIS-1817226, and Shuai’s research is partly funded by Adobe’s gift funding. We gratefully thank Charlie Dickens and Wesley J. Maddox for fruitful discussions, and appreciate Mengting Wan and Shi Feng for comments on our draft.

References

- [1] J. Ba and R. Caruana. Do deep nets really need to be deep? *ArXiv*, abs/1312.6184, 2014.
- [2] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.
- [3] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. Deep learning for classical japanese literature. *ArXiv*, abs/1812.01718, 2018.
- [4] K. L. Clarkson and D. P. Woodruff. Low rank approximation and regression in input sparsity time. In *STOC ’13*, 2013.
- [5] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *AISTATS*, 2011.
- [6] C. Cortes and M. Mohri. On transductive regression. In *Advances in Neural Information Processing Systems*, 2006.
- [7] C. Cortes and M. Mohri. Domain adaptation in regression. In *Algorithmic Learning Theory*, 2011.

- [8] C. Cortes, M. Mohri, and A. Rostamizadeh. Algorithms for learning kernels based on centered alignment. *Journal of Machine Learning Research*, 13:795–828, 2012.
- [9] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu. Boosting for transfer learning. In *International Conference on Machine Learning*, 2007.
- [10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.
- [11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.
- [12] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.
- [13] N. Frosst and G. E. Hinton. Distilling a neural network into a soft decision tree. *ArXiv*, abs/1711.09784, 2017.
- [14] A. Gittens and M. W. Mahoney. Revisiting the nyström method for improved large-scale machine learning. *The Journal of Machine Learning Research*, 17(1):3977–4041, 2016.
- [15] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, 2017.
- [16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.
- [17] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *ArXiv*, abs/1603.05027, 2016.
- [18] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.
- [19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.
- [20] K. Hornik, M. B. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [21] M. Jagadeesan. Understanding sparse jl for feature hashing. In *Advances in Neural Information Processing Systems*, 2019.
- [22] A. Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Master’s thesis, University of Toronto, Toronto, 2009.
- [23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 1998.
- [24] K. Muandet, D. Balduzzi, and B. Schölkopf. Domain generalization via invariant feature representation. In *International Conference on Machine Learning*, 2013.
- [25] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.
- [26] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *International Conference on Machine Learning*, 2005.
- [27] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.
- [28] N. Pham and R. Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *SIGKDD Conference on Knowledge Discovery and Data Mining*, 2013.
- [29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F.-F. Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2015.
- [30] M. Udell and A. Townsend. Why are big data matrices approximately low rank? *SIAM Journal on Mathematics of Data Science*, 1(1):144–160, 2019.
- [31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.
- [32] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, 2010.
- [33] D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10:1–157, 2014.
- [34] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, 2014.

A Dataset Descriptions

CIFAR10 [22] consists of 60k images, each of size 32×32 . The train/test split is made available, and the training set contains 50k images and the test set contains the rest. Each image has an object at the center, and the total 10 object categories are similar to ones in ImageNet dataset.

CIFAR100 [22] has 60k images as well, each of size 32×32 . The ratio of training images and test ones is the same as in CIFAR10. Each image has an object at the center, and in total, there are 100 object categories, which makes the task harder than CIFAR10.

STL10 [5] has 500 images for training and 800 images for testing per class, each of size 96×96 and in total, there are 10 classes. Since images of this dataset come from labelled samples from ImageNet but with lower resolution, models pretrained on ImageNet are expected to generalise well.

SVHN [25] consists of real-world images obtained from house numbers in Google Street View images, therefore, there are 10 categories. The training set contains 73,257 images, and the test set contains 26,032. The dataset also provides a set of 531,131 unlabelled images, and we didn't make use of it in our study.

CUB200 [32] is an image dataset with photos of 200 bird species (mostly North American). The total number of training images is 6,033, therefore, each class has around 30 training examples. The task itself is considered to difficult as it requires the model to pay attention to details of the bird presented in each image, which makes it a fine-grained classification problem.

Kuzushiji49 [3] is a Japanese character recognition task, which contains 48 Hiragana characters and one Hiragana iteration mark. The dataset itself is much larger than aforementioned ones, and contains only gray-scale images. The training set contains 232,365 images, and in our study, we only used half of the whole set. The test set contains 38,547 images, which is used to evaluate the effectiveness of our method and other methods.

B Results: RBF Baselines

RBF kernels as universal kernels are widely used in many research domains. Since we used features produced by neural network models learnt on the ImageNet dataset as inputs to an RBF kernel, it is reasonable to compare to the method that takes an ensemble of RBF kernels with various bandwidths and directly takes the vectorised images as inputs. Nyström approximation is applied to reduce the memory complexity.

Individual RBF kernels are selected as follows, and learning kernel alignment [8] is also applied to find the optimal combination of RBF kernels with different bandwidths.

$$k_{ij,p} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / (2^p \gamma)), \quad (5)$$

where $\gamma = \text{median}_{i,j \in \{1,2,\dots,N\}} \|\mathbf{x}_i - \mathbf{x}_j\|^2$
and $p \in \{-2, -1, \dots, 10\}$

The results are presented in Tab. 4. Since RBF kernels are directly operating on pixels of images without neural networks, the performance is worse than ours or finetuning the top layer (LogReg). It serves as an supporting evidence that inductive biases (prior knowledge) introduced by convolutional layers are important in image recognition tasks.

Table 4: Results of LogReg, our method and RBF kernels.

Methods	CIFAR10 [In]	CIFAR100 [In]	STL10 [In]
LogReg	87.45 / 89.94	69.08 / 72.76	95.08 / 96.55
RBF	51.40	21.76	43.58
Ours	90.77 / 92.31	71.31 / 74.63	96.30 / 97.31

C Results: Transferring within In-domain tasks

We have three in-domain transfer tasks, and train a model for each task then evaluate its performance on other tasks using our method. The results are presented in Tab. 5. Overall, our method provides reasonable performance across tasks, and it doesn't involve finetuning the models. Specifically, for STL10, as the dataset itself has very few images in the training set, models trained on CIFAR10 and CIFAR100 give better generalisation on STL10 than those trained on STL10 itself.

Table 5: Transferring within In-domain tasks.

Tasks for Pretraining	Transfer Tasks		
	CIFAR10	CIFAR100	STL10
CIFAR10	94.61	57.16	82.81
CIFAR100	87.85	76.21	80.63
STL10	73.72	43.65	67.85