Residual Parameter Transfer for Deep Domain Adaptation: Supplementary material

Artem Rozantsev Mathieu Salzmann Pascal Fua
Computer Vision Laboratory, École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
{firstname.lastname}@epfl.ch

Here, we first discuss the matrix representation of the parameters of the layers of each of the streams in our architecture. We then present in more detail our approach to estimating the matrices $\{\mathbf{A}_i^1\}$ and $\{\mathbf{A}_i^2\}$ from the inner part of the computed residual transformation $\{\mathbf{T}_i\}$, which was omitted from the main paper due to space restriction. We also report additional experiments that show that our method is independent from the specific domain discrepancy loss term $\mathcal{L}_{\texttt{disc}}$ we rely on.

1. Matrix form of the parameters

In Section 3.1 of the main paper, we showed that representing the layers' parameters in vector form results in having very large transformation matrices $\mathbf{A}, \mathbf{B}$, which leads to extensive memory usage and significant reduction in training speed. To overcome this problem, we propose to represent the layers' parameters in matrix form. Our strategy for different layer types is as follows:

- Fully connected layer. Such a layer performs a transformation of the form $\mathbf{y} = \sigma(\mathbf{A}\mathbf{x} + \mathbf{b})$, where $\mathbf{A}$ is a matrix and $\mathbf{b}$ a vector whose size is the number of rows of $\mathbf{A}$. In this case, we simply concatenate $\mathbf{A}$ and $\mathbf{b}$ into a single matrix.
- Convolutional layer. Such a layer is parametrized by a tensor $\mathbf{W} \in \mathbb{R}^{N_{out} \times N_{in} \times f_x \times f_y}$, where the convolutional kernel is of size $f_x \times f_y$, and a bias term $\mathbf{b} \in \mathbb{R}^{N_{out}}$. We therefore represent all these parameters as a single matrix by reshaping the kernel weights as an $N_{out} \times N_{in} f_x f_y$ matrix and again concatenating the bias with it.

Following these operations, the parameters of each layer i in the source and target streams are represented by matrices $\mathbf{\Theta}_i^s$ and $\mathbf{\Theta}_i^t$, respectively.

2. Least-Squares Solution

In Section 3.3.1 of the main paper we show how to analytically compute $\{\mathbf{T}_i\}$, that is, the approximation of the solution to Eq. 12. Here we discuss in more detail, how to estimate the transformation matrices $\{\mathbf{A}_i^1\}, \{\mathbf{A}_i^2\}$ from the resulting $\{\mathbf{T}_i\}$. Recall that, for each layer $i \in \Omega$, the inner part of the residual parameter transformation $\mathbf{T}_i$ is defined as

$$\mathbf{T}_i = \left(\mathbf{A}_i^1\right)^\intercal \Theta_i^s \mathbf{A}_i^2 + \mathbf{D}_i \ , \quad \mathbf{T}_i \in \mathbb{R}^{l_i \times r_i} \ . \tag{1}$$

As discussed in Section 3.3.1 of the main paper, we can estimate $\mathbf{A}_i^1 \in \mathbb{R}^{C_i \times l_i}$ and $\mathbf{A}_i^2 \in \mathbb{R}^{N_i \times r_i}$ by fixing $\mathbf{D}_i$, which in turn allows us to rewrite Eq. 1 as

$$\left(\mathbf{A}_i^1\right)^\intercal \mathbf{\Theta}_i^s \mathbf{A}_i^2 = \mathbf{T}_i - \mathbf{D}_i \ , \tag{2}$$

and solve it in the least-squares sense. Eq. 2, however, is under-constrained, which leaves us with a wide range of pairs $\{\{\mathbf{A}_i^1\}, \{\mathbf{A}_i^2\}\}$ that satisfy it. Therefore, in order to make the learning process stable, we suggest finding the optimal $\{\mathbf{A}_i^1\}$ and $\{\mathbf{A}_i^2\}$ that are the closest to the Adam [1] estimates $\{\hat{\mathbf{A}}_i^1\}, \{\hat{\mathbf{A}}_i^2\}$, as discussed in Section 3.3.1 of the main paper. To do so, for every layer $i \in \Omega$, we first find the least-squares solution to

$$\mathbf{A}_i^1 = \operatorname*{argmin}_{\tilde{\mathbf{A}}_i^1} \left\| \tilde{\mathbf{A}}_i^1 - \hat{\mathbf{A}}_i^1 \right\|_2^2 + \left\| \left(\tilde{\mathbf{A}}_i^1\right)^\intercal \mathbf{\Theta}_i^s \hat{\mathbf{A}}_i^2 - \mathbf{T}_i + \mathbf{D}_i \right\|_2^2 \ , \tag{3}$$

and then substitute the resulting $\mathbf{A}_i^1$ into

$$\mathbf{A}_i^2 = \operatorname*{argmin}_{\tilde{\mathbf{A}}_i^2} \left\| \tilde{\mathbf{A}}_i^2 - \hat{\mathbf{A}}_i^2 \right\|_2^2 + \left\| \left(\mathbf{A}_i^1\right)^\intercal \mathbf{\Theta}_i^s \tilde{\mathbf{A}}_i^2 - \mathbf{T}_i + \mathbf{D}_i \right\|_2^2 \ , \tag{4}$$

which we then solve in the least-squares sense. As both of these problems are no longer under-constrained, this procedure results in a solution $\{\mathbf{A}_i^1\}, \{\mathbf{A}_i^2\}$ that will both be close to the Adam estimates $\{\{\hat{\mathbf{A}}_i^1\}, \{\hat{\mathbf{A}}_i^2\}\}$ and approximately satisfy Eq. 1. We can then remove the rows of matrix $\mathbf{A}_i^1$ and columns of $\mathbf{A}_i^2$ that correspond to the rows and columns of $\mathbf{T}_i$ with an L_2 norm less than a small ϵ, as they will make no contribution on the final transformation.

$\mathbf{f}_j$	$\in \mathbb{R}^{N_j}$	layer outputs
$\mathbf{p}_j$	$\in \mathbb{R}^{D}$	layer output projections
$\mathbf{R}_j$	$\in \mathbb{R}^{N_j \times D}$	random projection matrices
$\mathbf{f}$	$\in \mathbb{R}^{D}$	resulting feature representation

N_j	number of output neurons of layer $j \in \Lambda$
Λ	predefined set of layers
D	predefined projection dimensionality

Table 1: Notation

3. Additional Experiments

To show that our method does not depend on the specific form of the domain discrepancy loss term $\mathcal{L}_{\texttt{disc}}$, we have replaced the domain classifier from DANN [2], which we used in the main paper, with the one from RMAN [3] that was recently introduced and showed state-of-the-art performance on many Domain Adaptation tasks. In short, this approach builds upon the methods of [2] and [4] by combining the outputs from multiple layers of the feature extracting architecture using random multilinear fusion.

More formally, in RMAN [3], the outputs $\{\mathbf{f}_j\}$ of a predefined set of layers Λ are projected into D-dimensional vectors $\{\mathbf{p}_j\}$ via a set of random projection matrices $\{\mathbf{R}_j\}$. Table 1 describes the notation in more detail. The resulting feature representation $\mathbf{f}$ is then formed as

$$\mathbf{f} = \frac{1}{\sqrt{D}}\left(\odot_j^{|\Lambda|}\mathbf{p}_j\right) \;, \quad j \in \Lambda \tag{5}$$

where $\odot$ is the element-wise (Hadamard) product. Finally, $\mathbf{f}$ is passed to the domain classifier ϕ, which tries to predict from which domain the sample comes, in the same way as done in [2]. We then construct $\mathcal{L}_{\texttt{disc}}$ in the same manner as in Section 3.2 of the main paper. In short, the major difference between DANN [2] and RMAN [3] is the input to the domain classifier, which allows for an easy integration of this method with our approach.

As the code for RMAN is not currently available, we reimplemented it and report the results on the SVHN → MNIST domain adaptation task. To this end, we used the SvhNet [5] architecture that was discussed in more detail in Section 4.2.1 of the main paper. To fuse the output of the final fully-connected layer and the raw classifier output into the 128-dimensional representation $\mathbf{f}$, we used projection matrices $\{\mathbf{R}_j\}$, the elements of which were sampled from the a Gaussian distribution $\mathcal{N}(0,1)$. The domain classifier that operates on $\mathbf{f}$ has exactly the same architecture as the one used in DANN [2] for the SVHN→MNIST domain adaptation task. In Table 2 we compare the results of our approach that uses the RMAN-based domain discrepancy term with the original RMAN method, which can be seen as a version of ours with all the parameters in the corresponding layers shared between the source and target streams. Note that our approach, which allows to transform the source weights to target ones, significantly outperforms RMAN. Fig. 1 illustrates the reduction in the complexity of the parameter transfer for every layer of the SvhNet architecture. As in our experiments in the main paper, the ranks of the transformation matrices are significantly reduced during the optimization process.

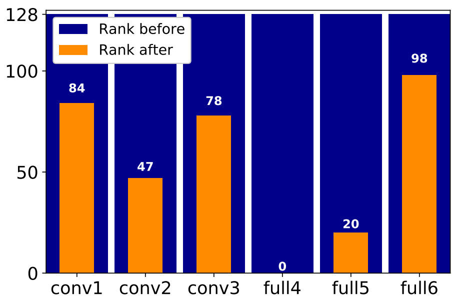

Figure 1: Automated complexity selection. Reduction of the transformation ranks in each SvhNet layer. The layers are shown on the x-axis and the corresponding ranks before and after optimization on the y-axis.

	Accuracy: mean [std]
RMAN [3]	81.0 [0.77]
Ours	**84.6 [1.26]**

Table 2: Comparison of our method that uses the RMAN-based domain discrepancy term with the original RMAN algorithm on the SVHN→MNIST domain adaptation task.

References

[1] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimisation," in *International Conference for Learning Representations*, 2015. 1

[2] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. S. Lempitsky, "Domain-Adversarial Training of Neural Networks," *Journal of Machine Learning Research*, vol. 17, pp. 591–5935, 2016. 2

[3] M. Long, Z. Cao, J. Wang, and M. I. Jordan, "Domain Adaptation with Randomized Multilinear Adversarial Networks," *arXiv Preprint*, vol. abs/1705.10667, 2017. 2

[4] M. Long, J. Wang, and M. I. Jordan, "Deep Transfer Learning with Joint Adaptation Networks," in *International Conference on Machine Learning*, 2017. 2

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. 2