Model Fusion via Optimal Transport

Sidak Pal Singh[*]
ETH Zurich, Switzerland
contact@sidakpal.com

Martin Jaggi
EPFL, Switzerland
martin.jaggi@epfl.ch

Abstract

Combining different models is a widely used paradigm in machine learning applications. While the most common approach is to form an ensemble of models and average their individual predictions, this approach is often rendered infeasible by given resource constraints in terms of memory and computation, which grow linearly with the number of models. We present a layer-wise model fusion algorithm for neural networks that utilizes optimal transport to (soft-) align neurons across the models before averaging their associated parameters.

We show that this can successfully yield "one-shot" knowledge transfer (i.e, without requiring any retraining) between neural networks trained on heterogeneous non-i.i.d. data. In both i.i.d. and non-i.i.d. settings, we illustrate that our approach significantly outperforms vanilla averaging, as well as how it can serve as an efficient replacement for the ensemble with moderate fine-tuning, for standard convolutional networks (like VGG11), residual networks (like RESNET18), and multi-layer perceptrons on CIFAR10, CIFAR100, and MNIST. Finally, our approach also provides a principled way to combine the parameters of neural networks with different widths, and we explore its application for model compression. The code is available at the following link, https://github.com/sidak/otfusion.

1 Introduction

If two neural networks had a child, what would be its weights? In this work, we study the fusion of two *parent* neural networks—which were trained differently but have the same number of layers—into a single *child* network. We further focus on performing this operation in a *one-shot manner*, based on the network weights only, so as to minimize the need of any retraining.

This fundamental operation of merging several neural networks into one contrasts other widely used techniques for combining machine learning models:

Ensemble methods have a very long history. They combine the outputs of several different models as a way to improve the prediction performance and robustness. However, this requires maintaining the K trained models and running each of them at test time (say, in order to average their outputs). This approach thus quickly becomes infeasible for many applications with limited computational resources, especially in view of the ever-growing size of modern deep learning models.

The simplest way to fuse several parent networks into a single network of the same size is direct *weight averaging*, which we refer to as vanilla averaging; here for simplicity, we assume that all network architectures are identical. Unfortunately, neural networks are typically highly redundant in their parameterizations, so that there is no one-to-one correspondence between the weights of two different neural networks, even if they would describe the same function of the input. In practice, vanilla averaging is known to perform very poorly on trained networks whose weights differ non-trivially.

Finally, a third way to combine two models is *distillation*, where one network is retrained on its training data, while jointly using the output predictions of the other 'teacher' network on those

---

[*]Work done while at EPFL.

samples. Such a scenario is considered infeasible in our setting, as we aim for approaches not requiring the sharing of training data.This requirement is particularly crucial if the training data is to be kept private, like in federated learning applications, or is unavailable due to e.g. legal reasons.

Contributions. We propose a novel layer-wise approach of aligning the neurons and weights of several differently trained models, for fusing them into a single model of the same architecture. Our method relies on optimal transport (OT) [1, 2], to minimize the transportation cost of neurons present in the layers of individual models, measured by the similarity of activations or incoming weights. The resulting layer-wise averaging scheme can be interpreted as computing the Wasserstein barycenter [3, 4] of the probability measures defined at the corresponding layers of the parent models.

We empirically demonstrate that our method succeeds in the one-shot merging of networks of different weights, and in all scenarios significantly outperforms vanilla averaging. More surprisingly, we also show that our method succeeds in merging two networks that were trained for slightly different tasks (such as using a different set of labels). The method is able to "inherit" abilities unique to one of the parent networks, while outperforming the same parent network on the task associated with the other network. Further, we illustrate how it can serve as a data-free and algorithm independent post-processing tool for structured pruning. Finally, we show that OT fusion, with mild fine-tuning, can act as efficient proxy for the ensemble, whereas vanilla averaging fails for more than two models.

Extensions and Applications. The method serves as a new building block for enabling several use-cases: (1) The adaptation of a global model to personal training data. (2) Fusing the parameters of a bigger model into a smaller sized model and vice versa. (3) Federated or decentralized learning applications, where training data can not be shared due to privacy reasons or simply due to its large size. In general, improved model fusion techniques such as ours have strong potential towards encouraging model exchange as opposed to data exchange, to improve privacy & reduce communication costs.

2 Related Work

Ensembling. Ensemble methods [5–7] have long been in use in deep learning and machine learning in general. However, given our goal is to obtain a single model, it is assumed infeasible to maintain and run several trained models as needed here.

Distillation. Another line of work by Hinton et al. [8], Buciluă et al. [9], Schmidhuber [10] proposes distillation techniques. Here the key idea is to employ the knowledge of a pre-trained teacher network (typically larger and expensive to train) and transfer its abilities to a smaller model called the student network. During this transfer process, the goal is to use the relative probabilities of misclassification of the teacher as a more informative training signal.

While distillation also results in a single model, the main drawback is its computational complexity—the distillation process is essentially as expensive as training the student network from scratch, and also involves its own set of hyper-parameter tuning. In addition, distillation still requires sharing the training data with the teacher (as the teacher network can be too large to share), which we avoid here.

In a different line of work, Shen et al. [11] propose an approach where the student network is forced to produce outputs mimicking the teacher networks, by utilizing Generative Adversarial Network [12]. This still does not resolve the problem of high computational costs involved in this kind of knowledge transfer. Further, it does not provide a principled way to aggregate the parameters of different models.

Relation to other network fusion methods. Several studies have investigated a method to merge two trained networks into a single network without the need for retraining [13–15]. Leontev et al. [15] propose Elastic Weight Consolidation, which formulates an assignment problem on top of diagonal approximations to the Hessian matrices of each of the two parent neural networks. Their method however only works when the weights of the parent models are already close, i.e. share a significant part of the training history [13, 14], by relying on SGD with periodic averaging, also called local SGD [16]. Nevertheless, their empirical results [15] do not improve over vanilla averaging.

Alignment-based methods. Alignment of neurons was considered in Li et al. [17] to probe the representations learned by different networks. Recently, Yurochkin et al. [18] independently proposed a Bayesian non-parametric framework that considers matching the neurons of different MLPs in federated learning. In a concurrent work[2], Wang et al. [19] extend [18] to more realistic networks

---

[2]An early version of our paper also appeared at NeurIPS 2019 workshop on OT, arxiv:1910.05653.

including CNNs, also with a specific focus on federated learning. In contrast, we develop our method from the lens of optimal transport (OT), which lends us a simpler approach by utilizing Wasserstein barycenters. The method of aligning neurons employed in both lines of work form instances for the choice of ground metric in OT. Overall, we consider model fusion in general, beyond federated learning. For instance, we show applications of fusing different sized models (e.g., for structured pruning) as well as the compatibility of our method to serve as an initialization for distillation. From a practical side, our approach is # of layer times more efficient and also applies to ResNets.

To conclude, the application of Wasserstein barycenters for averaging the weights of neural networks has—to our knowledge—not been considered in the past.

3 Background on Optimal Transport (OT)

We present a short background on OT in the discrete case, and in this process set up the notation for the rest of the paper. OT gives a way to compare two probability distributions defined over a ground space $\mathcal{S}$, provided an underlying distance or more generally the cost of transporting one point to another in the ground space. Next, we describe the linear program (LP) which lies at the heart of OT.

LP Formulation. First, let us consider two empirical probability measures μ and ν denoted by a weighted sum of Diracs, i.e., $\mu = \sum_{i=1}^{n} \alpha_i \, \delta(\boldsymbol{x}^{(i)})$ and $\nu = \sum_{i=1}^{m} \beta_i \, \delta(\boldsymbol{y}^{(i)})$. Here $\delta(\boldsymbol{x})$ denotes the Dirac (unit mass) distribution at point $\boldsymbol{x} \in \mathcal{S}$ and the set of points $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}) \in \mathcal{S}^n$. The weight $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ lives in the probability simplex (and similarly $\boldsymbol{\beta}$). Further, let $\boldsymbol{C}_{ij}$ denote the ground cost of moving point $\boldsymbol{x}^{(i)}$ to $\boldsymbol{y}^{(j)}$. Then the optimal transport between μ and ν can be formulated as solving the following linear program. $\mathrm{OT}(\mu, \nu; \boldsymbol{C}) := \min \langle \boldsymbol{T}, \boldsymbol{C} \rangle$, with $\boldsymbol{T} \in \mathbb{R}_{+}^{(n \times m)}$ such that $\boldsymbol{T} \mathbf{1}_m = \boldsymbol{\alpha}$, $\boldsymbol{T}^{\top} \mathbf{1}_n = \boldsymbol{\beta}$. Here, $\langle \boldsymbol{T}, \boldsymbol{C} \rangle := \mathrm{tr}\left(\boldsymbol{T}^{\top} \boldsymbol{C}\right) = \sum_{ij} T_{ij} C_{ij}$ is the Frobenius inner product of matrices. The optimal $\boldsymbol{T} \in \mathbb{R}_{+}^{(n \times m)}$ is called as the *transportation matrix* or *transport map*, and T_{ij} represents the optimal amount of mass to be moved from point $\boldsymbol{x}^{(i)}$ to $\boldsymbol{y}^{(j)}$.

Wasserstein Distance. When $\mathcal{S} = \mathbb{R}^d$ and the cost is defined with respect to a metric $D_{\mathcal{S}}$ over $\mathcal{S}$ $\left(\text{i.e., } C_{ij} = D_{\mathcal{S}}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(j)})^p \text{ for any } i, j\right)$, OT establishes a distance between probability distributions. This is called the p-Wasserstein distance and is defined as $\mathcal{W}_p(\mu, \nu) := \mathrm{OT}(\mu, \nu; D_{\mathcal{S}}^p)^{1/p}$.

Wasserstein Barycenters. This represents the notion of averaging in the Wasserstein space. To be precise, the Wasserstein barycenter [3] is a probability measure that minimizes the weighted sum of (p-th power) Wasserstein distances to the given K measures $\{\mu_1, \ldots, \mu_K\}$, with corresponding weights $\boldsymbol{\eta} = \{\eta_1, \ldots, \eta_K\} \in \Sigma_K$. Hence, it can be written as $\mathcal{B}_p(\mu_1, \ldots, \mu_K) = \arg\min_{\mu} \sum_{k=1}^{K} \eta_k \, \mathcal{W}_p(\mu_k, \nu)^p$.

4 Proposed Algorithm

In this section, we discuss our proposed algorithm for model aggregation. First, we consider that we are averaging the parameters of only two neural networks, but later present the extension to the multiple model case. For now, we ignore the bias parameters and we only focus on the weights. This is to make the presentation succinct, and it can be easily extended to take care of these aspects.

Motivation. As alluded to earlier in the introduction, the problem with vanilla averaging of parameters is the lack of one-to-one correspondence between the model parameters. In particular, for a given layer, there is no direct matching between the neurons of the two models. For e.g., this means that the pth neuron of model A might behave very differently (in terms of the feature it detects) from the pth neuron of the other model B, and instead might be quite similar in functionality to the $p + 1^{\text{th}}$ neuron. Imagine, if we knew a perfect matching between the neurons, then we could simply align the neurons of model A with respect to B. Having done this, it would then make more sense to perform vanilla averaging of the neuron parameters. The matching or assignment could be formulated as a permutation matrix, and just multiplying the parameters by this matrix would align the parameters.

But in practice, it is more likely to have soft correspondences between the neurons of the two models for a given layer, especially if their number is not the same across the two models. This is where optimal transport comes in and provides us a soft-alignment matrix in the form of the transport map $\boldsymbol{T}$. In other words, the alignment problem can be rephrased as optimally transporting the neurons in a given layer of model A to the neurons in the same layer of model B.
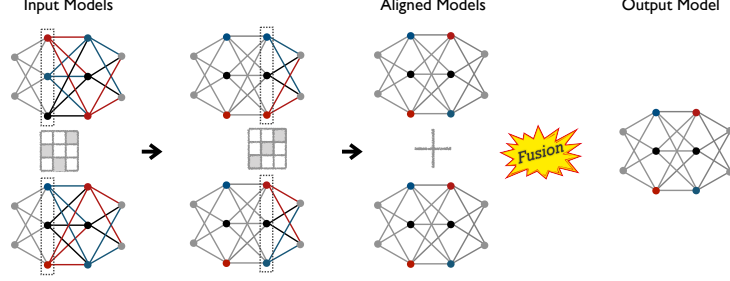
Figure 1: **Model Fusion procedure**: The first two steps illustrate how the model A (top) gets aligned with respect to model B (bottom). The alignment here is reflected by the ordering of the node colors in a layer. Once each layer has been aligned, the model parameters get averaged (shown by the $+$).

General procedure. Let us assume we are at some layer ℓ and that neurons in the previous layers have already been aligned. Then, we define probability measures over neurons in this layer for the two models as, $\mu^{(\ell)} = \big(\boldsymbol{\alpha}^{(\ell)}, \boldsymbol{X}[\ell]\big)$ and $\nu^{(\ell)} = \big(\boldsymbol{\beta}^{(\ell)}, \boldsymbol{Y}[\ell]\big)$, where $\boldsymbol{X}, \boldsymbol{Y}$ are the measure supports.

Next, we use uniform distributions to initialize the histogram (or probability mass values) for each layer. Although we note that it is possible to additionally use other measures of neuron importance [20, 21], but we leave it for a future work. In particular, if the size of layer ℓ of models A and B is denoted by $n^{(\ell)}, m^{(\ell)}$ respectively, we get $\boldsymbol{\alpha}^{(\ell)} \leftarrow \mathbf{1}_{n^{(\ell)}}/n^{(\ell)}$, $\boldsymbol{\beta}^{(\ell)} \leftarrow \mathbf{1}_{m^{(\ell)}}/m^{(\ell)}$. Now, in terms of the alignment procedure, we first align the incoming edge weights for the current layer ℓ. This can be done by post-multiplying with the previous layer transport matrix $\boldsymbol{T}^{(\ell-1)}$, normalized appropriately via the inverse of the corresponding column marginals $\boldsymbol{\beta}^{(\ell-1)}$:

$$\widehat{\boldsymbol{W}}_A^{(\ell,\,\ell-1)} \leftarrow \boldsymbol{W}_A^{(\ell,\,\ell-1)} \boldsymbol{T}^{(\ell-1)} \mathrm{diag}\big(1/\boldsymbol{\beta}^{(\ell-1)}\big). \tag{1}$$

This update can be interpreted as follows: the matrix $\boldsymbol{T}^{(\ell-1)}\mathrm{diag}\big(\boldsymbol{\beta}^{-(\ell-1)}\big)$ has $m^{(\ell-1)}$ columns in the simplex $\Sigma_{n^{(\ell-1)}}$, thus post-multiplying $\boldsymbol{W}_A^{(\ell,\,\ell-1)}$ with it will produce a convex combination of the points in $\boldsymbol{W}_A^{(\ell,\,\ell-1)}$ with weights defined by the optimal transport map $\boldsymbol{T}^{(\ell-1)}$.

Once this has been done, we focus on aligning the neurons in this layer ℓ of the two models. Let us assume, we have a suitable ground metric $D_{\mathcal{S}}$ (which we discuss in the sections ahead). Then we compute the optimal transport map $\boldsymbol{T}^{(\ell)}$ between the measures $\mu^{(\ell)}, \nu^{(\ell)}$ for layer ℓ, i.e., $\boldsymbol{T}^{(\ell)}, \mathcal{W}_2 \leftarrow \mathrm{OT}(\mu^{(\ell)}, \nu^{(\ell)}, D_{\mathcal{S}})$, where $\mathcal{W}_2$ denotes the obtained Wasserstein-distance. Now, we use this transport map $\boldsymbol{T}^{(\ell)}$ to align the neurons (more precisely the weights) of the first model (A) with respect to the second (B),

$$\widetilde{\boldsymbol{W}}_A^{(\ell,\,\ell-1)} \leftarrow \mathrm{diag}\big(1/\boldsymbol{\beta}^{(\ell)}\big){\boldsymbol{T}^{(\ell)}}^{\top} \widehat{\boldsymbol{W}}_A^{(\ell,\,\ell-1)}. \tag{2}$$

We will refer to model A's weights, $\widetilde{\boldsymbol{W}}_A^{(\ell,\,\ell-1)}$, as those aligned with respect to model B. Hence, with this alignment in place, we can average the weights of two layers to obtain the fused weight matrix $\boldsymbol{W}_{\mathcal{F}}^{(\ell,\,\ell-1)}$, as in Eq. (3). We carry out this procedure over all the layers sequentially.

$$\boldsymbol{W}_{\mathcal{F}}^{(\ell,\,\ell-1)} \leftarrow \tfrac{1}{2}\big(\widetilde{\boldsymbol{W}}_A^{(\ell,\,\ell-1)} + \boldsymbol{W}_B^{(\ell,\,\ell-1)}\big). \tag{3}$$

Note that, since the input layer is ordered identically for both models, we start the alignment from second layer onwards. Additionally, the order of neurons for the very last layer, i.e., in the output layer, again is identical. Thus, the (scaled) transport map at the last layer will be equal to the identity.

Extension to multiple models. The key idea is to begin with an estimate $\widehat{M}_{\mathcal{F}}$ of the fused model, then align all the given models with respect to it, and finally return the average of these aligned weights as the final weights for the fused model. For the two model case, this is equivalent to the procedure we discussed above when the fused model is initialized to model B, i.e., $\widehat{M}_{\mathcal{F}} \leftarrow M_B$. Because, aligning model B with this estimate of the fused model will yield a (scaled) transport map equal to the identity. And then, Eq. (3) will amount to returning the average of the aligned weights.

Alignment strategies. The above discussion implies that we need to design a ground metric $D_{\mathcal{S}}$ between the inter-model neurons. So, we branch out into the following two strategies:

(a) Activation-based alignment (ψ = 'acts'): In this variant, we run inference over a set of m samples, $S = \{\mathbf{x}\}_{i=1}^{m}$ and store the activations for all neurons in the model. Thus, we consider the neuron activations, concatenated over the samples into a vector, as the support of the measures, and we denote it as $\boldsymbol{X}_k \leftarrow \text{ACTS}\big(M_k(S)\big)$, $\boldsymbol{Y} \leftarrow \text{ACTS}\big(M_{\mathcal{F}}(S)\big)$. Then the neurons across the two models are considered to be similar if they produce similar activation outputs for the given set of samples. We measure this by computing the Euclidean distance between the resulting vector of activations. This serves as the ground metric for OT computations. In practice, we use the pre-activations.

(b) Weight-based alignment (ψ = 'wts'): Here, we consider that the support of each neuron is given by the weights of the incoming edges (stacked in a vector). Thus, a neuron can be thought as being represented by the row corresponding to it in the weight matrix. So, the support of the measures in such an alignment type is given by, $\boldsymbol{X}_k[\ell] \leftarrow \widehat{\boldsymbol{W}}_k^{(\ell,\,\ell-1)}$, $\boldsymbol{Y}[\ell] \leftarrow \widehat{\boldsymbol{W}}_{\mathcal{F}}^{(\ell,\,\ell-1)}$. The reasoning for such a choice for the support stems from the neuron activation at a particular layer being calculated as the inner product between this weight vector and the previous layer output. The ground metric used for OT is the Euclidean distance, like in the previous alignment strategy. Besides this difference of employing the actual weights in the ground metric (LINE 6, 10), rest of the procedure is identical.

Lastly, the overall procedure is summarized in Algorithm 1 below, where the GETSUPPORT selects between the above strategies based on the value of ψ.

Algorithm 1: Model Fusion (with $\psi = \{\text{'acts', 'wts'}\}-\text{alignment}$)

1: **input:**   Trained models $\{M_k\}_{k=1}^{K}$ and initial estimate of the fused model $\widehat{M}_{\mathcal{F}}$

2: **output:**   Fused model $M_{\mathcal{F}}$ with weights $\boldsymbol{W}_{\mathcal{F}}$

3: **notation:**   For model M_k, size of the layer ℓ is written as $n_k^{(\ell)}$, and the weight matrix between the layer ℓ and $\ell - 1$ is denoted as $\boldsymbol{W}_k^{(\ell,\,\ell-1)}$. Neuron support tensors are given by $\boldsymbol{X}_k, \boldsymbol{Y}$.

4: **initialize:**   The size of input layer $n_k^{(1)} \leftarrow m^{(1)}$ for all $k \in [K]$; so $\boldsymbol{\alpha}_k^{(1)} = \boldsymbol{\beta}^{(1)} \leftarrow \mathbf{1}_{m^{(1)}}/m^{(1)}$ and the transport map is defined as $\boldsymbol{T}_k^{(1)} \leftarrow \text{diag}(\boldsymbol{\beta}^{(1)}) \; \mathcal{I}_{m^{(1)} \times m^{(1)}}$.

5: **for** each layer $\ell = 2, \ldots, L$ **do**

6:      $\boldsymbol{\beta}^{(\ell)}, \boldsymbol{Y}[\ell]$ $\leftarrow \mathbf{1}_{m^{(\ell)}}/m^{(\ell)}$, GETSUPPORT$(\widehat{M}_{\mathcal{F}}, \psi, \ell)$

7:      $\nu^{(\ell)}$ $\leftarrow \big(\boldsymbol{\beta}^{(\ell)}, \boldsymbol{Y}[\ell]\big)$ ▷ Define probability measure for initial fused model $\widehat{M}_{\mathcal{F}}$

8:      **for** each model $k = 1, \ldots, K$ **do**

9:          $\widehat{\boldsymbol{W}}_k^{(\ell,\,\ell-1)}$ $\leftarrow \boldsymbol{W}_k^{(\ell,\,\ell-1)} \boldsymbol{T}_k^{(\ell-1)} \text{diag}\big(\frac{1}{\boldsymbol{\beta}^{(\ell-1)}}\big)$ ▷ Align incoming edges for M_k

10:         $\boldsymbol{\alpha}_k^{(\ell)}, \boldsymbol{X}_k[\ell]$ $\leftarrow \mathbf{1}_{n_k^{(\ell)}}/n_k^{(\ell)}$, GETSUPPORT$(M_k, \psi, \ell)$

11:         $\mu_k^{(\ell)}$ $\leftarrow \big(\boldsymbol{\alpha}_k^{(\ell)}, \boldsymbol{X}_k[\ell]\big)$ ▷ Define probability measure for model M_k

12:         $D_{\mathcal{S}}^{(\ell)}[p, q]$ $\leftarrow \|\boldsymbol{X}_k[\ell][p] - \boldsymbol{Y}[\ell][q]\|_2, \; \forall\, p \in [n_k^{(\ell)}], q \in [m^{(\ell)}]$ ▷ Form ground metric

13:         $\boldsymbol{T}_k^{(\ell)}, \mathcal{W}_2^{(\ell)}$ $\leftarrow \text{OT}\big(\mu_k^{(\ell)}, \nu^{(\ell)}, D_{\mathcal{S}}^{(\ell)}\big)$ ▷ Compute OT map and distance

14:         $\widetilde{\boldsymbol{W}}_k^{(\ell,\,\ell-1)}$ $\leftarrow \text{diag}\big(\frac{1}{\boldsymbol{\beta}^{(\ell)}}\big) \boldsymbol{T}^{(\ell)\top} \widehat{\boldsymbol{W}}_k^{(\ell,\,\ell-1)}$ ▷ Align model M_k neurons

15:      **end for**

16:      $\boldsymbol{W}_{\mathcal{F}}^{(\ell,\,\ell-1)}$ $\leftarrow \frac{1}{K} \sum_{k=1}^{K} \widetilde{\boldsymbol{W}}_k^{(\ell,\,\ell-1)}$ ▷ Average model weights

17: **end for**

4.1 Discussion

Pros and cons of alignment type.   An advantage of the weight-based alignment is that it is independent of the dataset samples, making it useful in privacy-constrained scenarios. On the flip side, the activation-based alignment only needs unlabeled data, and an interesting prospect for a future study would be to utilize synthetic data. But, activation-based alignment may help tailor the fusion to certain desired kinds of classes or domains. Fusion results for both are nevertheless similar.

Combinatorial hardness of the ideal procedure.   In principle, we should actually search over the space of permutation matrices, jointly across all the layers. But this would be computationally

intractable for models such as deep neural networks, and thus we fuse in a layer-wise manner and in a way have a greedy procedure.

# of samples used for activation-based alignment. We typically consider a mini-batch of ~ 100 to 400 samples for these experiments. Table S2 in the Appendix, shows that effect of increasing this mini-batch size on the fusion performance and we find that even as few as 25 samples are enough to outperform vanilla averaging.

Exact OT and runtime efficiency. Our fusion procedure is efficient enough for the deep neural networks considered here (VGG11, RESNET18), so we primarily utilize exact OT solvers. While the runtime of exact OT is roughly cubic in the cardinality of the measure supports, it is not an issue for us as this cardinality (which amounts to the network width) is ≤ 600 for these networks. In general, modern-day neural networks are typically deeper than wide. To give a concrete estimate, the *time taken to fuse six* VGG11 *models is* ≈ 15 *seconds* on 1 Nvidia V100 GPU (c.f. Section S1.4 for more details). It is possible to further improve the runtime by adopting the entropy-regularized OT [22], but this looses slightly in terms of test accuracy compared to exact OT (c.f. Table S4).

5 Experiments

Outline. We first present our results for one-shot fusion when the models are trained on *different data distributions*. Next, in Section 5.2, we consider (one-shot) fusion in the case when model sizes are different (i.e., unequal layer widths to be precise). In fact, this aspect *facilitates a new tool* that can be applied in ways not possible with vanilla averaging. Further on, we focus on the use-case of obtaining an *efficient* replacement for ensembling models in Section 5.3.

Empirical Details. We test our model fusion approach on standard image classification datasets, like CIFAR10 with commonly used convolutional neural networks (CNNs) such as VGG11 [23] and residual networks like ResNet18 [24]; and on MNIST, we use a fully connected network with 3 hidden layers of size $400, 200, 100$, which we refer to as MLPNET. As baselines, we mention the performance of 'prediction' ensembling and 'vanilla' averaging, besides that of individual models. Prediction ensembling refers to keeping all the models and averaging their predictions (output layer scores), and thus reflects in a way the ideal (but unrealistic) performance that we can hope to achieve when fusing into a single model. Vanilla averaging denotes the direct averaging of parameters. All the performance scores are test accuracies. Full experimental details are provided in Appendix S1.1.

5.1 Fusion in the setting of heterogeneous data and tasks

We first consider the setting of merging two models A and B, but assume that model A has some special skill or knowledge (say, recognizing an object) which B does not possess. However, B is overall more powerful across the remaining set of skills in comparison to A. The goal of fusion now is to obtain a single model that can gain from the strength of B on overall skills and also acquire the specialized skill possessed by A. Such a scenario can arise e.g. in reinforcement learning where these models are agents that have had different training episodes so far. Another possible use case lies in federated learning [25], where model A is a client application that has been trained to perform well on certain tasks (like personalized keyword prediction) and model B is the server that typically has a strong skill set for a range of tasks (general language model).

The natural constraints in such scenarios are (a) ensuring privacy and (b) minimization communication frequency. This implies that the training examples can not be shared between A and B to respect privacy and a one-shot knowledge transfer is ideally desired, which eliminates e.g., joint training.

At a very abstract level, these scenarios are representative of aggregating models that have been trained on non-i.i.d data distributions. To simulate a heterogeneous data-split, we consider the MNIST digit classification task with MLPNET models, where the unique skill possessed by model A corresponds to recognizing one particular 'personalized' label (say 4), which is unknown to B. Model B contains 90% of the remaining training set (i.e., excluding the label 4), while A has the other 10%. Both are trained on their portions of the data for 10 epochs , and other training settings are identical.

Figure 2 illustrates the results for fusing models A and B (in different proportions), both when they have different parameter initializations or when they share the same initialization. OT fusion [3] significantly outperforms the vanilla averaging of their parameters in terms of the overall test accuracy

---

[3]Only the receiver A's own examples are used for computing the activations, avoiding the sharing of data.

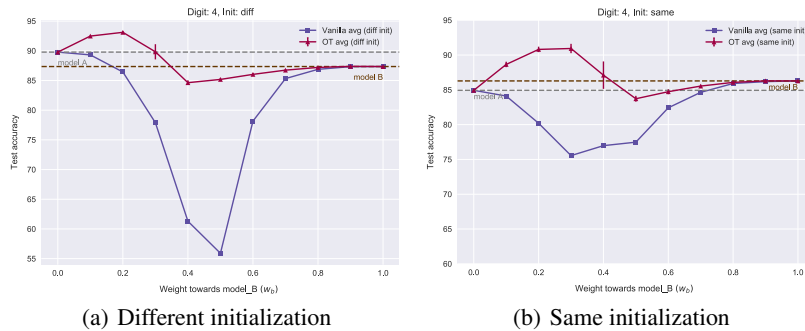(a) Different initialization   (b) Same initialization

Figure 2: **One-shot skill transfer performance** when the specialist model A and the generalist model B are fused in varying proportions (w_B), for different and same initializations. The OT avg. (fusion) curve (in magenta) is obtained by activation-based alignment and we plot the mean performance over 5 seeds along with the error bars for standard deviation. *No retraining is done here.*

in both the cases, and also improves over the individual models. E.g., in Figure 2(a), where the individual models obtain 89.78% and 87.35% accuracy respectively on the overall (global) test set, OT avg. achieves the best overall test set accuracy of 93.11%. Thus, confirming the successful skill transfer from both parent models, without the need for any retraining.

Our obtained results are robust to other scenarios when (i) some other label (say 6) serves as the special skill and (ii) the $\%$ of remaining data split is different. These results are collected in the Appendix S5, where in addition we also present results without the special label as well.

The case of multiple models. In the above example of two models, one might also consider maintaining an ensemble, however the associated costs for ensembling become prohibitive as soon as the numbers of models increases. Take for instance, four models: A, B, C and D, with the same initialization and assume that A again possessing the knowledge of a special digit (say, 4). Consider that the rest of the data is divided as $10\%, 30\%, 50\%, 10\%$. Now training in the similar setting as before, these models end up getting (global) test accuracies of $87.7\%, 86.5\%, 87.0\%, 83.5\%$ respectively. Ensembling the predictions yields 95.0% while vanilla averaging obtains 80.6%. In contrast, OT averaging results in **93.6%** test accuracy ($\approx 6\%$ gain over the best individual model), while being $4\times$ more efficient than ensembling. Further details can be found in the Appendix S7.

5.2 Fusing different sized models

An advantage of our OT-based fusion is that it allows the layer widths to be different for each input model. Here, our procedure first identifies which weights of the bigger model should be mapped to the smaller model (via the transport map), and then averages the aligned models (now both of the size of the smaller one). We can thus combine the parameters of a bigger network into a smaller one, and vice versa, allowing new use-cases in (a) model compression and (b) federated learning.

(a) Post-processing tool for structured pruning. Structured pruning [26–28] is an approach to model compression that aims to remove entire neurons or channels, resulting in an out-of-the-box reduction in inference costs, while affecting the performance minimally. A widely effective method for CNNs is to remove the filters with smallest ℓ_1 norm [26]. *Our key idea here is to fuse the original dense network into the pruned network, instead of just throwing it away.*

Figure 3: **Post-processing for structured pruning**: Fusing the initial dense VGG11 model into the pruned model helps test accuracy of the pruned model on CIFAR10.

Figure 3 shows the gain in test accuracy on CIFAR10 by carrying out OT fusion procedure (with weight-based alignment) when different convolutional layers of VGG11 are pruned to increasing amounts. For all the layers, we consistently obtain a significant improvement in performance, and $\approx 10\%$ or more gain in the high
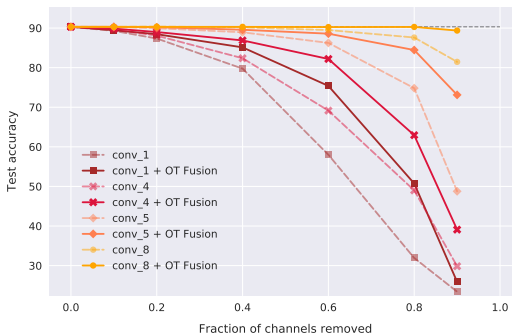
sparsity regime. We also observe similar improvements other layers as well as when multiple (or all) layers are pruned simultaneously (c.f. Appendix S8).

Further, these gains are also significant when measured with respect to the overall sparsity obtained in the model. E.g., structured pruning the CONV_8 to 90% results in a net sparsity of 23% in the model. Here after pruning, the accuracy of the model drops from 90.3% to 81.5%, and on applying OT fusion, the performances recovers to 89.4%. As an another example take CONV_7, where after structured pruning to 80%, OT fusion improves the performance of the pruned model from 87.6% to 90.1% while achieving an overall sparsity of 41% in the network (see S8).

Our goal here is not to propose a method for structured pruning, but rather a post-processing tool that can help regain the drop in performance due to pruning. These results are thus independent of the pruning algorithm used, and e.g., Appendix S8 shows similar gains when the filters are pruned based on ℓ_2 norm (Figure S10) or even randomly (Figure S11). Further, Figure S12 in the appendix also shows the results when applied to VGG11 trained on CIFAR100 (instead of CIFAR10). Overall, OT fusion offers a *completely data-free approach* to improving the performance of the pruned model, which can be handy in the limited data regime or when retraining is prohibitive.

(b) Adapting the size of client and server-side models in federated learning. Given the huge sizes of contemporary neural networks, it is evident that we will not able to fit the same sized model on a client device as would be possible on the server. However, this might come at the cost of reduced performance. Further, the resource constraints might be fairly varied even amongst the clients devices, thus necessitating the flexibility to adapt the model sizes.

We consider a similar formulation, as in the one-shot knowledge transfer setting from Section 5.1, except that now the model B has twice the layer widths as compared to the corresponding layers of model A. Vanilla averaging of parameters, a core component of the widely prevalent FedAvg algorithm [25], gets ruled out in such a setting. Figure 4 shows how OT fusion/average can still lead to a successful knowledge transfer between the given models.

Figure 4: **One-shot skill transfer for different sized models**: Results of fusing the small client model A into the larger server model B, for varying proportions w_B in which they are fused. See Appendix S6 for more details.

5.3 Fusion for efficient ensembling

In this section, our goal is to obtain a single model which can serve as a proxy for an ensemble of models, even if it comes at a slight decrease in performance relative to the ensemble, *for future efficiency*. Specifically, here we investigate how much can be gained by fusing multiple models that differ only in their parameter initializations (i.e., seeds). This means that models are trained on the same data, so unlike in Section 5.1 with a heterogeneous data-split, the gain here might be limited.

We study this in context of deep networks such as VGG11 and RESNET18 which have been trained to convergence on CIFAR10. As a first step, we consider the setting when we are given just two models, the results for which are present in Table 1. We observe that vanilla averaging absolutely fails in this case, and is 3-5× worse than OT averaging, in case of RESNET18 and VGG11 respectively. OT average, however, does not yet improve over the individual models. This can be attributed to the combinatorial hardness of

DATASET + MODEL	M_A	M_B	PREDICTION AVG.	VANILLA AVG.	OT AVG.	FINETUNING VANILLA	OT
CIFAR10 + VGG11	90.31	90.50	91.34	17.02	85.98	90.39	**90.73**
	1 ×		1 ×	2 ×	2 ×	2 ×	**2 ×**
CIFAR10 + RESNET18	93.11	93.20	93.89	18.49	77.00	93.49	**93.78**
	1 ×		1 ×	2 ×	2 ×	2 ×	**2 ×**

Table 1: Results for fusing convolutional & residual networks, along with the effect of finetuning the fused models, on CIFAR10. The number below the test accuracies indicate the factor by which a fusion technique is efficient over maintaining all the given models.

the underlying alignment problem, and the greedy nature of our algorithm as mentioned before. As a simple but effective remedy, we consider finetuning (i.e., retraining) from the fused or averaged models. Retraining helps for both vanilla and OT averaging, but in comparison, the OT averaging
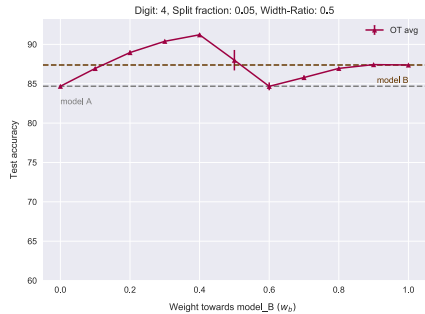
CIFAR100 + VGG11	INDIVIDUAL MODELS	PREDICTION AVG.	FINETUNING	
			VANILLA	OT
Accuracy	[62.70, 62.57, 62.50, 62.92]	66.32	4.02	**64.29**± **0.26**
Efficiency	$1 \times$	$1 \times$	$4 \times$	**4** $\times$
Accuracy	[62.70, 62.57, 62.50, 62.92, 62.53, 62.70]	66.99	0.85	**64.55 ± 0.30**
Efficiency	$1 \times$	$1 \times$	$6 \times$	**6** $\times$
Accuracy	[62.70, 62.57, 62.50, 62.92, 62.53, 62.70, 61.60, 63.20]	67.28	1.00	**65.05**± **0.53**
Efficiency	$1 \times$	$1 \times$	$8 \times$	**8** $\times$

Table 2: Efficient alternative to ensembling via OT fusion on **CIFAR100** for VGG11. Vanilla average fails to retrain. Results shown are mean $\pm$ std. deviation over **5 seeds**.

results in a better score for both the cases as shown in Table 1. E.g., for RESNET18, OT avg. + finetuning gets almost as good as prediction ensembling on test accuracy.

The finetuning scores for vanilla and OT averaging correspond to their best obtained results, when retrained with several finetuning learning rate schedules for a total of 100 and 120 epochs in case of VGG11and RESNET18 respectively. We also considered finetuning the individual models across these various hyperparameter settings (which of course will be infeasible in practice), but the best accuracy mustered via this attempt for RESNET18 was 93.51, in comparison to 93.78 for OT avg. + finetuning. See Appendix S3 and S4 for detailed results and typical retraining curves.

More than 2 models. Now, we discuss the case of more than two models, where the savings in efficiency relative to the ensemble are even higher. As before, we take the case of VGG11 on CIFAR10 and additionally CIFAR100 [4], but now consider $\{4, 6, 8\}-$ such models that have been trained to convergence, each from a different parameter initialization. Table 2 shows the results for this in case of CIFAR100 (results for CIFAR10 are similar and can be found in Table S9).

We find that the performance of vanilla averaging degrades to close-to-random performance, and interestingly even fails to retrain, despite trying numerous settings of optimization hyperparameters (like learning rate and schedules, c.f. Section S3.2). In contrast, OT average performs significantly better even without fine-tuning, and results in a mean test accuracy gain $\sim \{1.4\%, 1.7\%, 2\%\}$ over the best individual models after fine-tuning, in the case of $\{4, 6, 8\}-$ base models respectively. Overall, Tables 1, 2 (also S9) show the importance of aligning the networks via OT before averaging. Further finetuning of the OT fused model, always results in an improvement over the individual models, while being # *models times* more efficient than the ensemble.

Fusion and Distillation. For the sake of completeness, we also compare OT fusion, distillation, and their combination, in context of transferring the knowledge of a large pre-trained teacher network into a smaller pre-trained student network. We find that starting the distillation from the OT fused model yields better performance than initializing randomly or with the student model. Further, when averaged across the considered temperature values $= \{20, 10, 8, 4, 1\}$, we observe that distillation of the teacher into random or student network based initialization performs worse than simple OT avg. + finetuning (which also doesn't require doing such a sweep that would be prohibitive for larger models/datasets). These experiments are discussed in detail in Appendix S12. An interesting direction for future work would be to use intermediate OT distances computed during fusion as a means for regularizing or distilling with hidden layers.

6   Conclusion

We show that averaging the weights of models, by first doing a layer-wise (soft) alignment of the neurons via optimal transport, can serve as a versatile tool for fusing models in various settings. This results in (a) successful one-shot transfer of knowledge between models without sharing training data, (b) data free and algorithm independent post-processing tool for structured pruning, (c) and more generally, combining parameters of different sized models. Lastly, the OT average when further finetuned, allows for just keeping one model rather than a complete ensemble of models at inference. Future avenues include application in distributed optimization and continual learning, besides extending our current toolkit to fuse models with different number of layers, as well as, fusing generative models like GANs [12] (where ensembling does not make as much sense). The promising empirical results of the presented algorithm, thus warrant attention for further use-cases.

---

[4]We simply adapt the VGG11 architecture used for CIFAR10 and train it on CIFAR100 for 300 epochs. Since our focus here was not to obtain best individual models, but rather to investigate the efficacy of fusion.

Broader Impact

Model fusion is a fundamental building block in machine learning, as a way of direct knowledge transfer between trained neural networks. Beyond theoretical interest it can serve a wide range of concrete applications. For instance, collaborative learning schemes such as federated learning are of increasing importance for enabling privacy-preserving training of ML models, as well as a better alignment of each individual's data ownership with the resulting utility from jointly trained machine learning models, especially in applications where data is user-provided and privacy sensitive [29]. Here fusion of several models is a key building block to allow several agents to participate in joint training and knowledge exchange. We propose that a reliable fusion technique can serve as a step towards more broadly enabling privacy-preserving and efficient collaborative learning.

References

[1] Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781. 2, 29

[2] Leonid V Kantorovich. On the translocation of masses. In *Dokl. Akad. Nauk. USSR (NS)*, volume 37, pages 199–201, 1942. 2, 29

[3] Martial Agueh and Guillaume Carlier. Barycenters in the wasserstein space. *SIAM Journal on Mathematical Analysis*, 43(2):904–924, 2011. 2, 3

[4] Marco Cuturi and Arnaud Doucet. Fast computation of wasserstein barycenters. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 685–693, Bejing, China, 22–24 Jun 2014. PMLR. 2, 29

[5] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug 1996. ISSN 1573-0565. doi: 10.1023/A:1018054314350. URL https://doi.org/10.1023/A:1018054314350. 2

[6] David H. Wolpert. Original contribution: Stacked generalization. *Neural Netw.*, 5(2):241–259, February 1992. ISSN 0893-6080. doi: 10.1016/S0893-6080(05)80023-1. URL http://dx.doi.org/10.1016/S0893-6080(05)80023-1.

[7] Robert E. Schapire. A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'99, pages 1401–1406, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1624312.1624417. 2

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 2, 32

[9] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5. doi: 10.1145/1150402.1150464. URL http://doi.acm.org/10.1145/1150402.1150464. 2

[10] Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992. 2

[11] Zhiqiang Shen, Zhankui He, and Xiangyang Xue. Meal: Multi-model ensemble via adversarial learning, 2018. 2

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 2, 9

[13] Joshua Smith and Michael Gashler. An investigation of how neural networks learn from the experiences of peers through periodic weight averaging. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 731–736. IEEE, 2017. 2

[14] Joachim Utans. Weight averaging for neural networks and local resampling schemes. In *Proc. AAAI-96 Workshop on Integrating Multiple Learned Models. AAAI Press*, pages 133–138, 1996. 2

[15] Mikhail Iu Leontev, Viktoriia Islenteva, and Sergey V Sukhov. Non-iterative knowledge fusion in deep convolutional neural networks. *arXiv preprint arXiv:1809.09399*, 2018. 2

[16] Sebastian Urban Stich. Local sgd converges fast and communicates little. In *ICLR 2019 - International Conference on Learning Representations*, 2019. 2

[17] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent learning: Do different neural networks learn the same representations?, 2016. 2

[18] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks, 2019. 2

[19] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkluqlSFDS. 2

[20] Kedar Dhamdhere, Mukund Sundararajan, and Qiqi Yan. How important is a neuron. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SylKoo0cKm. 4

[21] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks, 2017. 4

[22] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013. 6

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. 6

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr. 2016.90. URL http://dx.doi.org/10.1109/CVPR.2016.90. 6

[25] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2016. 6, 8

[26] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets, 2016. 7

[27] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[28] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *J. Emerg. Technol. Comput. Syst.*, 13(3), February 2017. ISSN 1550-4832. doi: 10.1145/3005348. URL https://doi.org/10.1145/3005348. 7

[29] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019. 10

[30] Ari S. Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation, 2018. 16

[31] Sira Ferradans, Nicolas Papadakis, Julien Rabin, Gabriel Peyré, and Jean-François Aujol. Regularized discrete optimal transport. *Scale Space and Variational Methods in Computer Vision*, page 428–439, 2013. ISSN 1611-3349. doi: 10.1007/978-3-642-38267-3_36. URL http://dx.doi.org/10.1007/978-3-642-38267-3_36. 29

[32] L. Ambrosio, Nicola Gigli, and Giuseppe Savare. Gradient flows: in metric spaces and in the space of probability measures. 2006. URL https://www.springer.com/gp/book/9783764387211. 29

[33] Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Mean-field theory of two-layers neural networks: dimension-free bounds and kernel limit, 2019. 30

Appendix

Contents

S1 Technical specifications

S1.1 Experimental Details

VGG11 training details. It is trained by SGD for 300 epochs with an initial learning rate of 0.05, which gets decayed by a factor of 2 after every 30 epochs. Momentum = 0.9 and weight decay = 0.0005. The batch size used is 128. Checkpointing is done after every epoch and the best performing checkpoint in terms of test accuracy is used as the individual model. The block diagram of VGG11 architecture is shown below for reference.
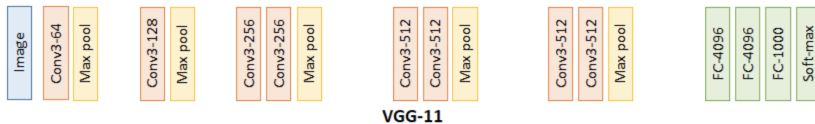
Figure S1: Block diagram of the VGG11 architecture. Adapted from https://bit.ly/2ksX5Eq.

MLPNET training details. This is also trained by SGD at a constant learning rate of 0.01 and momentum = 0.5. The batch size used is 64.

RESNET18 training details. Again, we use SGD as the optimizer, with an initial learning rate of 0.1, which gets decayed by a factor of 10 at epochs $\{150, 250\}$. In total, we train for 300 epochs and similar to the VGG11 setting we use the best performing checkpoint as the individual model. Other than that, momentum = 0.9, weight decay = 0.0001, and batch size = 256. We skip the batch normalization for the current experiments, however, it can possibly be handled by simply multiplying the batch normalization parameters in a layer by the obtained transport map while aligning the neurons.

Other details. *Pre-activations.* The results for the activation-based alignment experiments are based on pre-activation values, which were generally found to perform slightly better than post-activation values.

Regularization. The regularization constant used for the activation-based alignment results in Table S2 is 0.05.

Common details. The bias of a neuron is set to zero in all of the experiments. It is possible to handle it as a regular weight by keeping the corresponding input as 1, but we leave that for future work.

S1.2 Combining weights and activations for alignment

The output activation of a neuron over input examples gives a good signal about the presence of features in which the neuron gets activated. Hence, one way to combine this information in the above variant with weight-based alignment is to use them in the probability mass values.

In particular, we can take a mini-batch of samples and store the activations of all the neurons. Then we can use the mean activation as a measure of a neuron's significance. But it might be that some neurons produce very high activations (in absolute terms) irrespective of the kind of input examples. Hence, it might make sense to also look at the standard deviation of activations. Thus, one can combine both these factors into an importance weight for the neuron as follows:

$$\text{importance}_k[2, \cdots, L] = \overline{M_k}([x_1, \cdots, x_d]) \odot \sigma(M_k([x_1, \cdots, x_d])) \tag{4}$$

Here, M_k denotes the kth model into which we pass the inputs $[x_1, \cdots, x_d]$, $\overline{M}$ denotes the mean, $\sigma(.)$ denotes the standard deviation and $\odot$ denotes the elementwise product. Thus, we can now set the probability mass values $b_k^{(l)} \propto \text{importance}_k[l]$, and the rest of the algorithm remains the same.

S1.3   Optimal Transport

We make use of the Python Optimal Transport (POT)[S1] for performing the computation of Wasserstein distances and barycenters on CPU. These can also be implemented on the GPU to further boost the efficiency, although it suffices to run on CPU for now, as evident from the timings below.

S1.4   Timing information

The following timing benchmarks are done on 1 Nvidia V100 GPU. The time taken to average two MLPNET models for MNIST is ≈ 3 seconds. For averaging VGG11 models on CIFAR10, it takes about ≈ 5 seconds. While in case of RESNET18 on CIFAR10, it takes ≈ 7 seconds. These numbers are for the activation-based alignment, and also include the time taken to compute the activations over the mini-batch of examples.

The weight-based alignment can be faster as it does not need to compute the activations. For instance, when weight-based alignment is employed to average two VGG11 models on CIFAR10, it takes ≈ 2.5 seconds.

S2   Ablation studies

S2.1   Aggregation performance as training progresses

We compare the performance of averaged models at various points during the course of training the individual models (for the setting of MLPNet on MNIST). We notice that in the early stages of training, vanilla averaging performs even worse, which is not the case for OT averaging. The corresponding Figure S2 and Table S1 can be found in Section S2.1 of the Appendix. Overall, we see OT averaging outperforms vanilla averaging by a large margin, thus pointing towards the benefit of aligning the neurons via optimal transport.
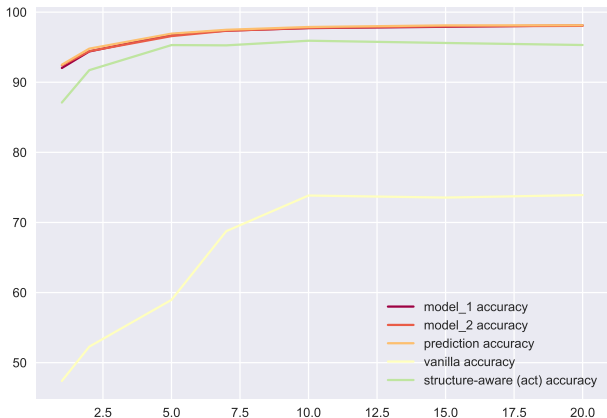
Figure S2: Illustrates the performance of various aggregation methods as training proceeds, for (MNIST, MLPNET). The plots correspond to the results reported in Table S1. The activation-based alignment of the OT average (labelled as structure-aware accuracy in the figure) is used based on $m = 200$ samples.

S2.2   Transport map for the output layer.

Since our algorithm runs until the output layer, we inspect the alignment computed for the last output layer. We find that the ratio of the trace to the sum for this last transport map is ≈ 1, indicating accurate alignment as the ordering of output units is the same across models.

[S1]http://pot.readthedocs.io/en/stable/

14

Epoch	Model A	Model B	Prediction avg.	Vanilla avg.	OT avg.
01	92.03	92.40	92.50	47.39	87.10
02	94.39	94.43	94.79	52.28	91.72
05	96.83	96.58	96.93	58.96	95.30
07	97.36	97.34	97.48	68.76	95.26
10	97.72	97.75	97.88	73.84	95.92
15	97.91	97.97	98.11	73.55	95.60
20	98.11	98.04	98.13	73.91	95.31

Table S1: **Activation-based alignment (MNIST, MLPNet):** Comparison of performance when ensembled after different training epochs. The # samples used for activation-based alignment, $m = 50$. The corresponding plot for this table is illustrated in Figure S2.

S2.3 Effect of mini-batch size needed for activation-based mode

Here, the individual models used are MLPNET's which have been trained for 10 epochs on MNIST. They differ only in their seeds and thus in the initialization of the parameters alone. We ensemble the final checkpoint of these models via OT averaging and the baseline methods.

M_A	M_B	Prediction avg.	Vanilla avg.	m	OT avg. (Sinkhorn) Accuracy (mean $\pm$ stdev)	M_A aligned
(a) Activation-based Alignment						
97.72	97.75	97.88	73.84	2	24.80 ± 6.93	20.08 ± 2.42
				10	75.04 ± 11.35	88.18 ± 8.45
				25	90.95 ± 3.98	95.36 ± 0.96
				50	93.47 ± 1.69	96.04 ± 0.59
				100	95.40 ± 0.52	$\mathbf{97.05 \pm 0.17}$
				200	$\mathbf{95.78 \pm 0.52}$	97.01 ± 0.16
(b) Weight-based Alignment						
97.72	97.75	97.88	73.84	—	95.66	96.32

Table S2: One-shot averaging for (MNIST, MLPNet) **with Sinkhorn and regularization** $= 0.05$: Results showing the performance (i.e., test classification accuracy (in %)) of the OT averaging in contrast to the baseline methods. The last column refers to the aligned model A which gets (vanilla) averaged with model B, giving rise to our OT averaged model. m is the size of mini-batch over which activations are computed.

S2.4 Effect of regularization

The results for activation-based alignment presented in the Table S2 above use the regularization constant $\lambda = 0.05$. Below, we also show the results with a higher regularization constant $\lambda = 0.1$. As expected, we find that using a lower value of regularization constant leads to better results in general, since it better approximates OT.

M_A	M_B	Prediction	Vanilla	m	OT avg. Accuracy (mean $\pm$ stdev)	M_A aligned
97.72	97.75	97.88	73.84	2	25.05 ± 7.22	19.42 ± 2.28
				10	72.86 ± 11.93	74.35 ± 14.40
				25	89.49 ± 5.21	90.88 ± 4.91
				50	92.88 ± 2.03	94.54 ± 1.36
				100	95.14 ± 0.49	96.42 ± 0.39
				200	$\mathbf{95.70 \pm 0.54}$	$\mathbf{96.63 \pm 0.23}$

Table S3: Activation-based alignment (MNIST, MLPNet) **with Sinkhorn and regularization** $= 0.1$: Results showing the performance (i.e., test classification accuracy) of the averaged and aligned models of OT based averaging in contrast to vanilla averaging of weights as well as the prediction based ensembling. m denotes the number of samples over which activations are computed, i.e., the mini-batch size.

S2.5 Exact vs regularized variant

In Table S4, we contrast the results obtained when no regularization is used and exact optimal transport is considered. Since using the exact optimal transport is fast enough, we default to using it hereafter.

M_A	M_B	PREDICTION AVG.	VANILLA AVG.	ALIGNMENT TYPE	OT AVG.	M_A ALIGNED Accuracy (mean)
Regularized OT (via Sinkhorn)				Activation	95.78	97.01
97.72	97.75	97.78	73.84	Weight	95.66	96.32
Exact OT				Activation	96.21	97.72
97.72	97.75	97.78	73.84	Weight	96.63	97.72

Table S4: **Exact vs Regularized OT:** Results showing the performance gain with exact OT for activation/weight based alignment. Here, regularization $\lambda = 0.05$.

S2.6 Layer-wise Optimal Transport distances
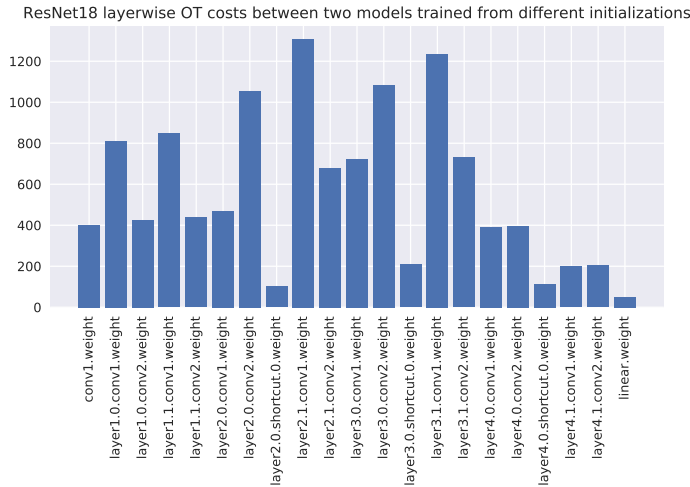
Figure S3: Illustrates the layerwise Optimal Transport costs between the corresponding layers of two ResNet18 models trained from different initializations, when using activation-based alignment with mini-batch size $m = 200$.

A possible application of our model fusion approach can be for inspecting the similarity of representations at various layers across different neural networks. Thus, it could provide an alternative perspective for this problem of understanding the similarity of representations, besides the canonical correlation analysis (CCA) based methods used in the past [30]. Figure S3 gives an example of this for two ResNet18 models trained from different initializations. Here, we used activation-based alignment with mini-batch size $m = 200$. An extensive study, however, remains beyond the scope of this paper.

S3 Detailed finetuning results

In Tables S5, S7, and S8, we report the results of finetuning (i.e. retraining) the averaged models for (MNIST, MLPNET) and (CIFAR10, VGG11). For comparison, we also show the performance of individual models when further finetuned in this setting. Although in general, **the individual model finetuning is not realistic**, since it is not known which one will lead to an improvement and this incurs # models $\times$ the finetuning cost.

S3.1 Two model scenario

S3.1.1 For MNIST + MLPNET

The finetuning is carried out for 60 epochs at the following set of constant learning rates $\{0.01, 0.002, 0.001, 0.00067, 0.0005\}$. Note that the original models were trained for 10 epochs at a learning rate of 0.01. For OT average, we use the activation-based alignment with mini-batch size $m = 200$.

Table S5 shows the results for each method at their best respective finetuning runs.

FINETUNING LR	MODEL A	MODEL B	VANILLA AVG.	OT AVG. (EXACT)
Baseline Results				
—	97.72	97.75	73.84	96.54
*Results for the **best** finetuning run* (reported at the best checkpoint)				
0.01	98.21	98.13	98.23	**98.35**
0.002	98.13	98.03	98.13	**98.21**
0.001	98.09	98.03	97.98	**98.14**
0.00067	**98.11**	98.00	97.83	98.07
0.0005	**98.09**	98.01	97.70	98.05

Table S5: **Effect of finetuning the individual and averaged models for (MNIST, MLPNet):** Best finetuning runs have been reported for each method. Cells in orange highlight the best scores in each regime.

We also show in Table S6 the results when averaged across 5 finetuning runs for each of the finetuning LR, as the cost of finetuning here is not as prohibitive in comparison to finetuning VGG11 and ResNet18 models. We see that performance trend remains in accordance with the previous Table S5.

FINETUNING LR	MODEL A	MODEL B	VANILLA AVG.	OT AVG. (EXACT)
Baseline Results				
—	97.72	97.75	73.84	96.21 ± 0.36
***Averaged** results across the finetuning runs* (reported at the best checkpoint)				
0.01	98.19 ± 0.02	98.11 ± 0.02	98.22 ± 0.02	$\mathbf{98.28 \pm 0.05}$
0.002	98.13 ± 0.01	98.03 ± 0.01	98.13 ± 0.01	$\mathbf{98.15 \pm 0.07}$
0.001	$\mathbf{98.11 \pm 0.02}$	98.01 ± 0.01	97.99 ± 0.01	98.08 ± 0.05
0.00067	$\mathbf{98.11 \pm 0.02}$	98.00 ± 0.01	97.83 ± 0.02	98.05 ± 0.04
0.0005	$\mathbf{98.09 \pm 0.01}$	98.01 ± 0.00	97.68 ± 0.01	98.03 ± 0.03

Table S6: **Effect of finetuning the individual and averaged models for (MNIST, MLPNet):** Average of the results across 5 finetuning runs as well as their standard deviation are reported for each method. Cells in orange highlight the best scores in each regime.

S3.1.2 For CIFAR10 + VGG11

As a recall, the original models were trained for 300 epochs at an initial learning rate of 0.05, which was decayed by a factor of 2 after every 30 epochs. The finetuning is carried out for 100 epochs at the following set of initial learning rates $\{0.01, 0.05, 0.0033, 0.0025\}$. Also, similar to training, the learning rate is decayed in the finetuning process. Note that, here finetuning at the initial learning rate of 0.01 causes model B to diverge and hence we skip the results for this setting.

For OT average, we use the weight-based alignment. Table S7 shows the best results for each method during their finetuning run.

FINETUNING LR	MODEL A	MODEL B	VANILLA AVG.	OT AVG. (EXACT)
Baseline Results				
—	90.31	90.50	17.02	85.98
Results after finetuning (reported scores are at best checkpoint)				
0.01	90.29	90.53	90.39	**90.73**
0.005	90.36	90.47	90.16	**90.64**
0.0033	90.28	90.39	90.13	**90.39**
0.0025	90.45	**90.50**	89.88	90.30

Table S7: **Effect of finetuning the individual and averaged models for (CIFAR10, VGG11):** Model A & Model B baseline accuracies correspond to best checkpoints when originally trained for 300 epochs. Cells in orange highlight the best scores in each regime.

S3.1.3 For CIFAR10 + RESNET18

As a recall, the original models were trained for 300 epochs at an initial learning rate of 0.1, which was decayed by a factor of 10 at the epochs $\{150, 250\}$. The finetuning is carried out for 120 epochs at the following set of initial learning rates $\{0.1, 0.04, 0.02\}$. For OT average, we use the activation-based alignment, with mini-batch size $m = 200$.

FINETUNING LR	MODEL A	MODEL B	VANILLA AVG.	OT AVG. (EXACT)
Baseline Results				
—	93.11	93.20	18.49	67.46
Results after finetuning (reported at the best checkpoint)				
(a) *LR decay epochs* $= [20, 40, 60, 80, 100]$				
0.1	93.51	93.43	93.29	**93.78**
0.04	93.35	93.34	93.28	**93.35**
0.02	93.28	**93.28**	93.09	92.97
(b) *LR decay epochs* $= [40, 80]$				
0.1	93.49	93.32	93.34	**93.59**
0.04	93.27	93.34	**93.49**	93.38
0.02	93.21	**93.33**	93.17	93.15

Table S8: **Effect of finetuning the individual and averaged models for (CIFAR10, RESNET18):** Model A and Model B baseline accuracies correspond to best checkpoints when originally trained for 300 epochs. Cells in orange highlight the best scores in each regime.

Table S8 shows the best results for each method during their finetuning run. The learning rate is decayed by a factor of 2 in the finetuning process as per two schedules: (a) after every 20 epochs, and (b) after every 40 epochs. These are indicated in the respective sections of the Table S8.

S3.2 Multiple model scenario: CIFAR10

Now, we discuss in detail, the experiments performed for the multiple model setting on CIFAR10. Namely, when we have 4 and 6 VGG11 models, that have different initializations, but are trained identically on the entire data, as mentioned in Table S9.

CIFAR10+ VGG11	INDIVIDUAL MODELS	PREDICTION AVG.	VANILLA AVG.	OT AVG.	FINETUNING VANILLA	FINETUNING OT
Accuracy	[90.31, 90.50, 90.43, 90.51]	91.77	10.00	73.31	12.40	90.91
Efficiency	$1\times$	$1\times$	$4\times$	$4\times$	$4\times$	$4\times$
Accuracy	[90.31, 90.50, 90.43, 90.51, 90.49, 90.40]	91.85	10.00	72.16	11.01	91.06
Efficiency	$1\times$	$1\times$	$6\times$	$6\times$	$6\times$	$6\times$

Table S9: Results of our OT average + finetuning based efficient alternative for ensembling in contrast to vanilla average + finetuning, for more than two input models (VGG11) with different initializations trained on CIFAR10.

We consider finetuning the averaged models, with many different optimization hyperparameters, however vanilla average fails to finetune or retrain. In particular, we finetune for 150 epochs with learning rate obtained by dividing the original learning rate (with which models were trained) by factors of $\{1, 2, 4, 8, 16\}$ (called 'initial decay'). Further, similar to learning rate schedule followed in the training, we try decaying the learning rate by a factor of $\{1.1, 1.5, 2.0\}$ after every 20 epochs. We also tried adjusting the interval after which the learning rate was decayed (like 40 epochs), but this was again to no avail in being able to finetune the vanilla average. So for simplicity, in the rest of discussion, we consider that the interval after which the learning rate gets decayed is 20 epochs.

Across all the settings OT average is able to successfully retrain, except when the learning rate is set to the original learning rate of 0.05, with which models were trained (i.e., initial decay of 1). This is to be expected as the OT average without retraining itself already performs fairly well, and setting such a high learning rate is bound to cause this. In contrast, vanilla average fails to retrain at all, with the best accuracy of 12.40 and 11.01 for the case of 4 and 6 models, when the initial decay is 1, and the learning rate decay is 1.1.

Finetuning from OT average results, in a significant improvement for numerous settings of the above hyperparameters, and below, we show the top 5 such settings in Table S10 for both 4 and 6 models. (For OT average, we use the activation-based alignment.)

INITIAL DECAY FACTOR	SCHEDULED LR DECAY FACTOR	DECAY INTERVAL	FINETUNING VANILLA AVG.	FINETUNING OT AVG.
(i) Number of models = 4				
2	2.0	20	10.34	90.91
4	2.0	20	10.32	90.80
2	2.0	40	10.34	90.74
2	1.5	20	10.34	90.67
4	2.0	40	10.32	90.66
(ii) Number of models = 6				
2	2.0	20	10.00	91.06
2	1.5	20	10.00	90.97
4	2.0	20	10.00	90.88
4	2.0	40	10.00	90.81
8	2.0	40	10.00	90.69

Table S10: Different finetuning settings which show how OT fusion can improve over the individual models after finetuning, while the vanilla average fails to do so. As a result, we obtain one single improved model that can be used as an efficient replacement for the ensemble.

S4   Finetuning curves
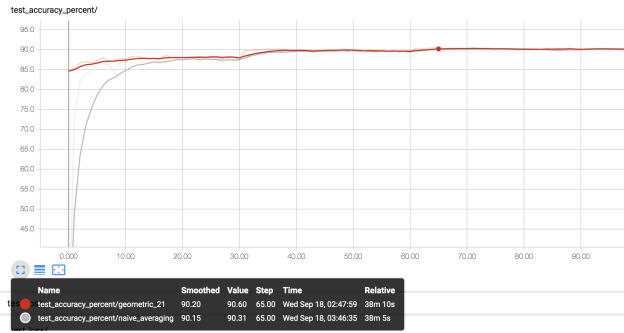
Figure S4: Illustrates the performance of OT averaging (referred to as geometric in the figure legend) and vanilla averaging during the process of retraining for CIFAR10 with VGG11.
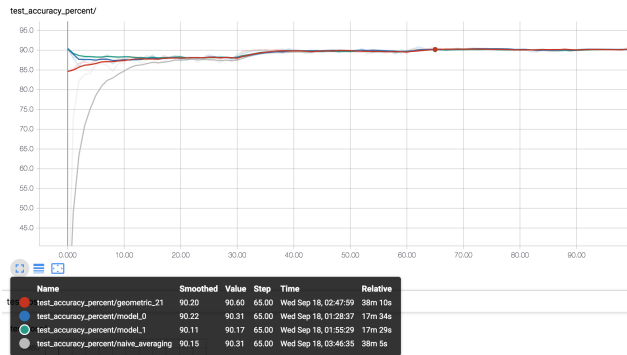
Figure S5: Retraining with reference plots of individual models. Other than that same as above.

S5 Skill Transfer: Additional Results
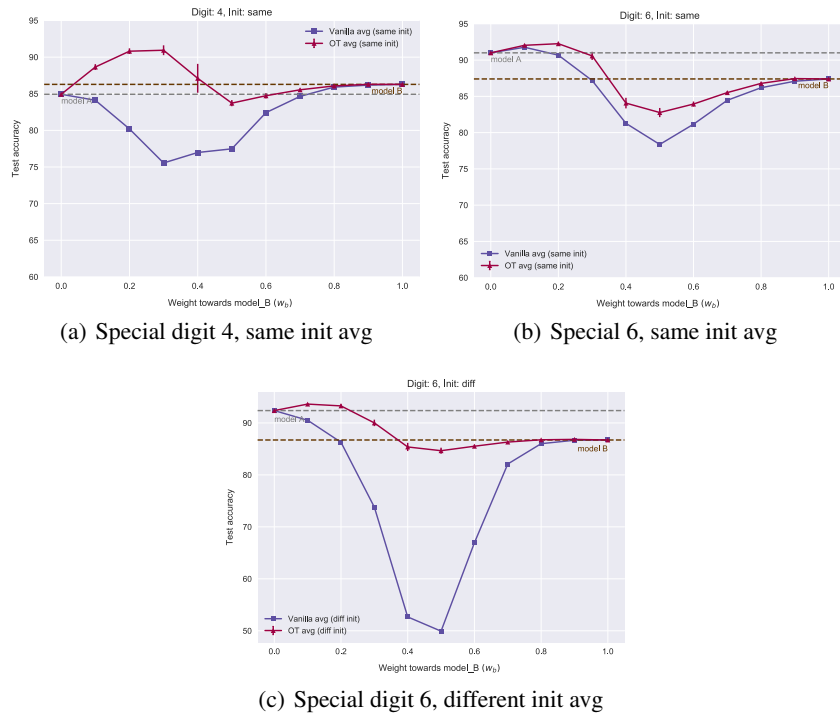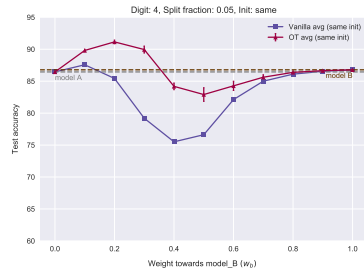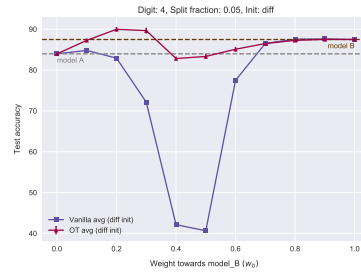
S5.1 Remaining Data Split: 10%

(a) Special digit 4, same init avg

(b) Special 6, same init avg

(c) Special digit 6, different init avg

Figure S6: **Skill Transfer performance**: Comparison results of OT based model fusion (OT avg) with vanilla averaging for different w_B. Each point for OT avg. curve (magenta colored) is obtained by activation-based alignment with a batch size $m = 400$, and we plot the mean performance over 5 seeds along with the error bars, which show the corresponding standard deviation. Here the remaining data besides the special digit, is split as 90% for model B and the other 10% for model A.

S5.2 Remaining Data Split: 5%

(a) Special digit 4, same init avg

(b) Special digit 4, different init avg
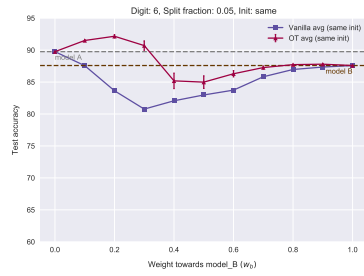
(c) Special digit 6, same init avg
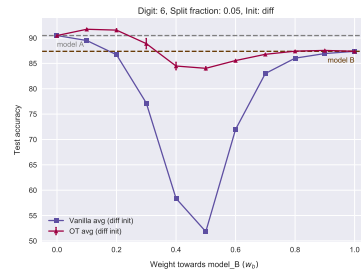
(d) Special digit 6, different init avg

Figure S7: **Skill Transfer performance**: Comparison results of OT based model fusion (OT avg) with vanilla averaging for different w_B. Each point for OT avg. curve (magenta colored) is obtained by activation-based alignment with a batch size $m = 400$, and we plot the mean performance over 5 seeds along with the error bars, which show the corresponding standard deviation. Here the remaining data besides the special digit, is split as 95% for model B and the other 5% for model A.

S5.3 Scenarios without specialized labels

Even if we don't exclude a digit and just alter the fraction of data between A and B, results are similar. E.g., take MLPNETS A and B with *same* initialization (*to help vanilla averaging*), but A has 30% and B has 70% of the data. This results in (global) test accuracy $\%$ of 94.2 and 95.0 for A and B resp. OT fusion is better than vanilla averaging when combining A and B for all proportions, with best results as, OT: mean **95.3** (stdev=0.1), vanilla avg: 95.1 at proportions $0.1, 0.9$ respectively. Ensembling is better than both (95.5), but requires 2x more memory and inference time.

Likewise, for other data splits (such as 10% vs 90%, 50% vs 50%, etc), OT fusion outperforms the individual models as well as vanilla averaging. For, further settings, also see Section S7.

S6 Results for one-shot skill-transfer under size constraints

Here, we present results for one-shot skill-transfer when the two models are of unequal sizes. More concretely, as an example, we consider that the hidden layers of the generalist model B are twice as wide as that of the specialist model A. Figure S8 illustrates the results for OT-based model fusion (OT average) in such a setting. Note that, here vanilla averaging can not be applied as the models are of different sizes. To the best of our knowledge, we are unaware of any other method that can allows for such one-shot skill transfer (i.e., fuse the given different size models into a single model in one-shot).

(a) Special digit 4, data split% = 10

(b) Special digit 4, data split% = 5

(c) Special digit 6, data split% = 10

(d) Special digit 6, data split% = 5
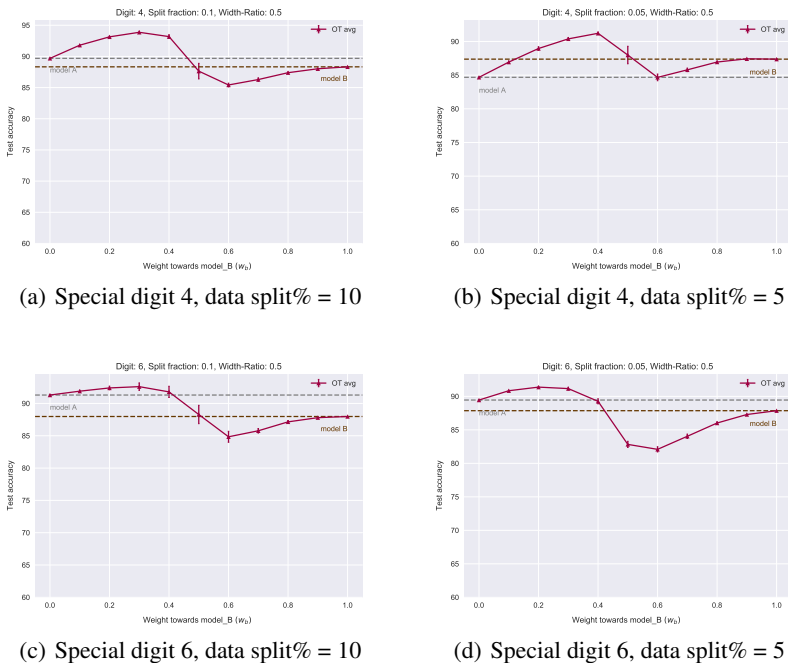
Figure S8: **Skill Transfer performance for different sized models**: Results of OT-based model fusion (OT avg) for different w_B. Unlike the results in the previous section, vanilla averaging is not possible here as the models are of unequal sizes. 'Width-Ratio 0.5' in the figure title denotes the ratio of the hidden layers sizes of model A and B. Each point for OT avg. curve (magenta colored) is obtained by activation-based alignment with a batch size $m = 400$, and we plot the mean performance over 5 seeds along with the error bars, which show the corresponding standard deviation. The data split % indicates the amount of remaining data besides the special digit which is present with model A. Model B contains 100 - data split% of this remaining data.

Rest of the technical details are identical as in the setup of Sections 5.1 in the main text and S5 in the supplementary.

S7   Multi-model one-shot skill transfer

To recap, here we take four MLPNET models: A, B, C and D, with the same initialization and assume that A again possessing the knowledge of a special digit (say, 4). Consider that the rest of the data is divided as $10\%, 30\%, 50\%, 10\%$.

Now they are trained in a similar setting for 10 epochs, by the end of which these models obtain (global) test accuracies of $87.7\%, 86.5\%, 87.0\%, 83.5\%$ respectively. Since A is the only model which has seen the special digit '4', we assign it a larger proportion in the final fused model. In particular, we consider fusing the models in proportions of $0.7, 0.1, 0.2, 0.1$ respectively *(later normalized to sum to 1)*. Then, ensembling the predictions yields 95.0% while vanilla averaging obtains 80.6%. In contrast, OT averaging results in **93.6%** test accuracy ($\approx 6\%$ gain over the best individual model), while being $4\times$ more efficient than ensembling.

This is also robust to many other proportions in which the models are combined. For example, decreasing the weight of model A so that the proportions are $0.6, 0.1, 0.2, 0.1$, gives: Prediction ensembling 95.03%, vanilla average 78.44%, OT average **92.72%**. Or increasing the proportion of B and D, i.e., let the proportions be instead $0.7, 0.15, 0.2, 0.15$. The results for such a case are as follows, Prediction ensembling 94.91%, vanilla average 76.14%, OT average 91.67%. Take another example, say we increase the proportion of model C now, so as to have the proportions $0.7, 0.1, 0.3, 0.1$. In this case, we get Prediction ensembling 95.15%, vanilla average 77.93%, OT average **92.21**%. We can go on for many other examples, but the results remain similar.

Overall, we find that OT average leads to a significant across all these examples, and outperforms vanilla average by a large margin. In comparison to prediction ensembling, it is slightly worse in terms of accuracy, but it enjoys $4\times$ efficiency, with respect to future usage and maintenance.

S8   Post-processing for structured pruning

CIFAR10.   In this section, we present the detailed results for using OT fusion as a post-processing tool for structured pruning. We show the benefit gained by OT fusion when separately pruning all layers of VGG11, as well as pruning them all together. This is illustrated for the three cases: (a) when filters with smallest ℓ_1 norms are removed, (b) when filters with smallest ℓ_2 norms are removed, and (c) when filters are removed randomly, in Figures S9, S10, and S11 respectively.

CIFAR100.   Also in Figures S12, we show the results of a similar experiment when pruning a VGG11 model trained on CIFAR100. Here as well, OT fusion leads to a performance boost when used as for post-processing. For simplicity, we only include the results with ℓ_1-pruner.
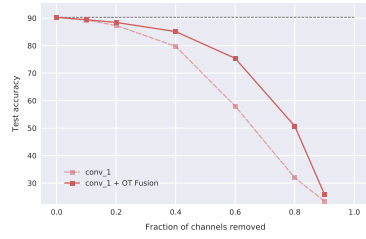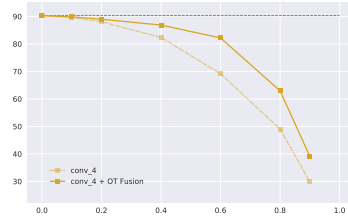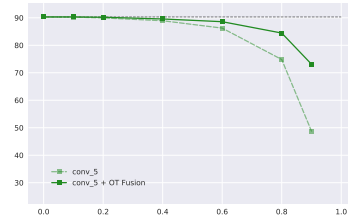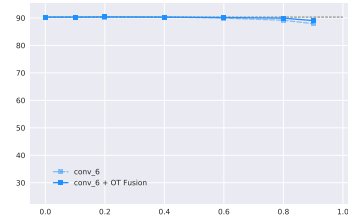
(a) conv_1
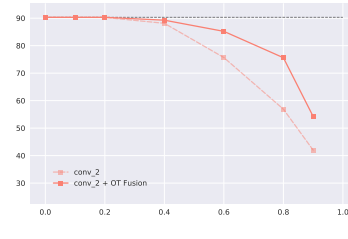
(b) conv_2
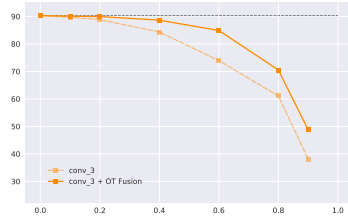
(c) conv_3
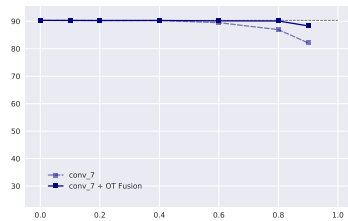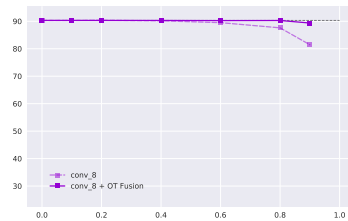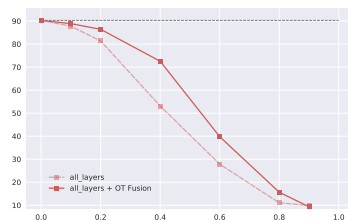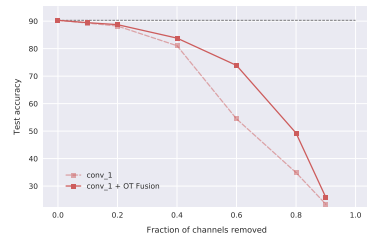

(d) conv_4

(e) conv_5

(f) conv_6

(g) conv_7

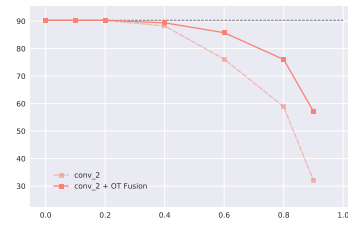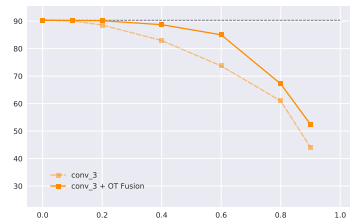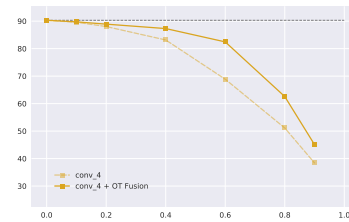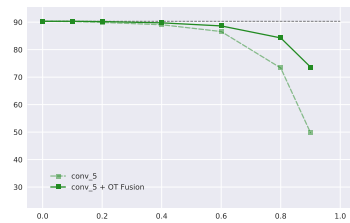(h) conv_8

(i) all

Figure S9: Post-processing for structured pruning **with ℓ_1 norm**, all figures: Fusing the initial dense VGG11 model into the pruned model helps test accuracy of the pruned model on **CIFAR10**.
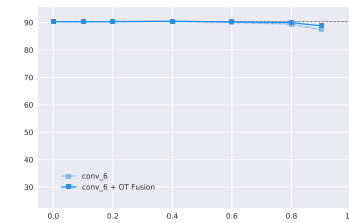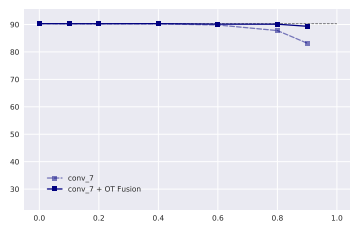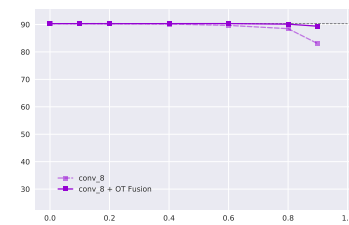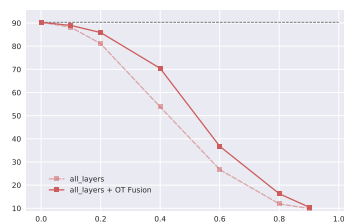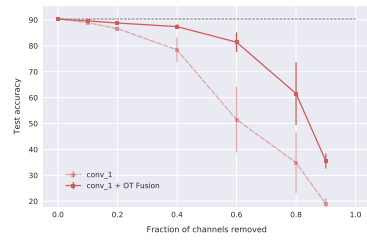
(a) conv_1

(b) conv_2

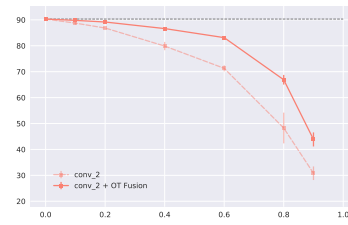
(c) conv_3

(d) conv_4

(e) conv_5

(f) conv_6

(g) conv_7

(h) conv_8

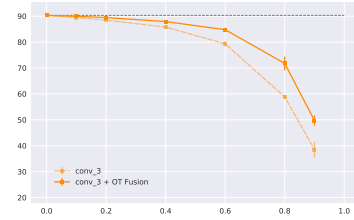

(i) all

Figure S10: Post-processing for structured pruning **with ℓ_2 norm**, all figures: Fusing the initial dense VGG11 model into the pruned model helps test accuracy of the pruned model on **CIFAR10**.
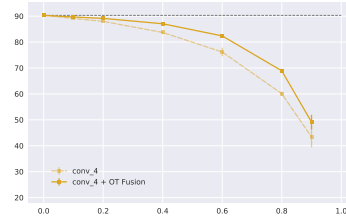
(a) conv_1

(b) conv_2

(c) conv_3

(d) conv_4

(e) conv_5

(f) conv_6

(g) conv_7

(h) conv_8

(i) all

Figure S11: Post-processing for structured pruning **with random**, all figures: Fusing the initial dense VGG11 model into the pruned model helps test accuracy of the pruned model on **CIFAR10**. Results are averaged over 3 seeds.
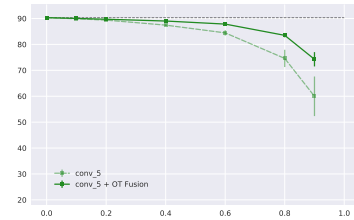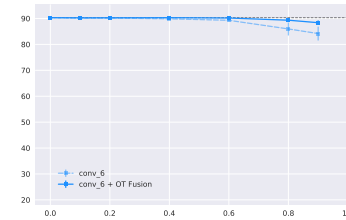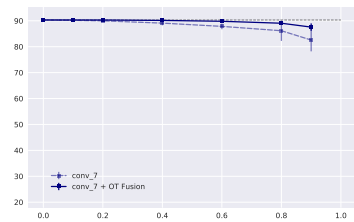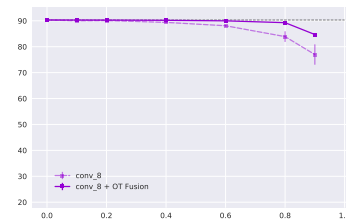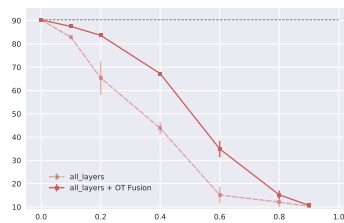
(a) conv_1

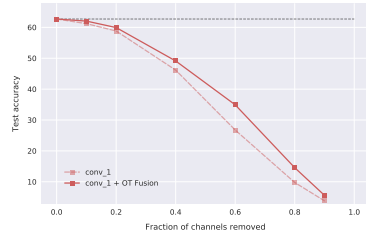(b) conv_2

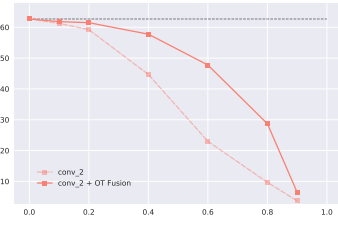
(c) conv_3
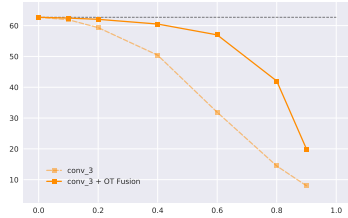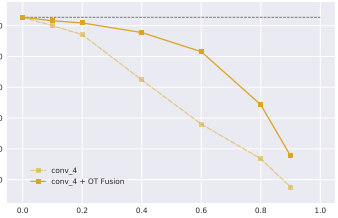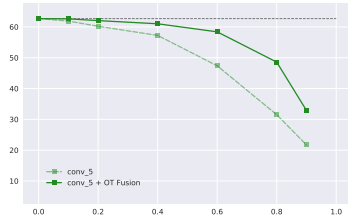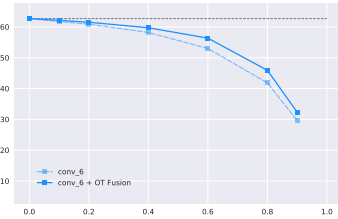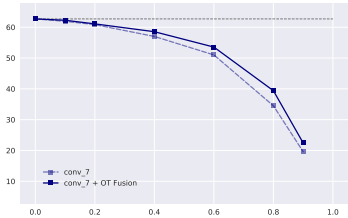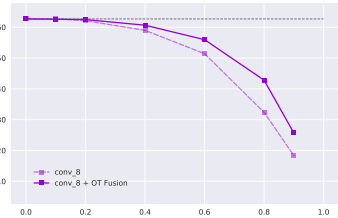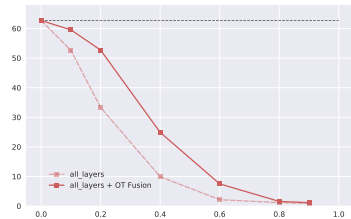

(d) conv_4

(e) conv_5

(f) conv_6

(g) conv_7

(h) conv_8

(i) all

Figure S12: Post-processing for structured pruning **with ℓ_1 norm**, all figures: Fusing the initial dense VGG11 model into the pruned model helps test accuracy of the pruned model on **CIFAR100**.

S9 Additional discussion on the update rule in the algorithm

S9.1 Barycentric projection

The original formulation by [1] considers finding a mapping $f : \boldsymbol{S}_A \mapsto \boldsymbol{S}_B$, which maps points in the support of μ_A to points in the support of μ_B. However, under this formulation, the optimal transport problem is not always feasible and Kantorovich [2] relaxed this by instead considering the optimization over the set of coupling matrices $\boldsymbol{T}$ (i.e., doubly stochastic matrices). Hence, a simple way to obtain the optimal map f from this coupling/transportation matrix $\boldsymbol{T}$ is $f(\boldsymbol{S}_A{}^i) = \boldsymbol{Z}_i$, where $\boldsymbol{Z} = \boldsymbol{S}_B \boldsymbol{T}^\top \operatorname{diag}\left(\alpha^{-1}\right)$, c.f. Ferradans et al. [31]. Essentially, this maps each point in support of A to a weighted average of the points in support of B. This is referred to as the barycentric projection.

For our algorithm in the weight-based alignment, we basically need the opposite map $f' : \boldsymbol{S}_B \mapsto \boldsymbol{S}_A$ to get the weights for the model A aligned with respect to B. This can be done by simply exchanging the above supports and transposing the matrix $\boldsymbol{T}$. Thus we have, $f'(\boldsymbol{S}_B^i) = \boldsymbol{Z}_i'$, where $\boldsymbol{Z}' = \boldsymbol{S}_A \boldsymbol{T} \operatorname{diag}\left(\beta^{-1}\right)$. In other words, we represent each point in the support of B with a weighted average of points in support of A.

Lastly, the supports in this weight-based alignment are defined by the corresponding weight matrices of the layers. Thus, implying the update in Eq. (1). Note that, when the underlying ground cost used is the squared Euclidean distance, this barycentric mapping is known to be optimal [32].

S9.2 Free-support barycenters

Next, we discuss the setup and a part of the derivation of free-support barycenters proposed in [4].

Problem formulation. Assume that the ground metric $D_{\mathcal{S}}$ is the Euclidean distance and $p = 2$. Consider the supports $\boldsymbol{S}_A$ and $\boldsymbol{S}_B$ as family of n_A and n_B points respectively in $\mathbb{R}^d$. Therefore represent them by a matrix in $\mathbb{R}^{d \times n_A}$ and $\mathbb{R}^{d \times n_B}$ respectively. If we use the notation, $\mathbf{s_A} \stackrel{\text{def}}{=} \operatorname{diag}\left(\boldsymbol{S}_A^\top \boldsymbol{S}_A\right)$ and $\mathbf{s_B} \stackrel{\text{def}}{=} \operatorname{diag}\left(\boldsymbol{S}_B^\top \boldsymbol{S}_B\right)$, then we can write the pairwise squared-Euclidean distances as follows:

$$\boldsymbol{C}_{AB} = \mathbf{s_A} \mathbf{1}_{n_B}^\top + \mathbf{1}_{n_A} \mathbf{s_B}^\top - 2 \boldsymbol{S}_A^\top \boldsymbol{S}_B \in \mathbb{R}^{n_A \times n_B}. \tag{5}$$

Now the optimal transport objective mentioned in Section 3 can be written in a more compact form in the equation (6) by using the matrix inner product notation. As a recall, $\langle \boldsymbol{U}, \boldsymbol{V} \rangle = \operatorname{tr}(\boldsymbol{U}^\top \boldsymbol{V})$, so we have:

$$\begin{aligned} \langle \boldsymbol{T}, \boldsymbol{C}_{AB} \rangle &= \left\langle \boldsymbol{T}, \mathbf{s_A} \mathbf{1}_d^\top + \mathbf{1}_d^\top \mathbf{s_B} - 2 \boldsymbol{S}_A^\top \boldsymbol{S}_B \right\rangle \\ &= \operatorname{tr}(\boldsymbol{T}^\top \mathbf{s_A} \mathbf{1}_d^\top) + \operatorname{tr}(\boldsymbol{T}^\top \mathbf{1}_d^\top \mathbf{s_B}) - 2 \left\langle \boldsymbol{T}, \boldsymbol{S}_A^\top \boldsymbol{S}_B \right\rangle \\ &= \mathbf{s_A}^\top \alpha + \mathbf{s_B}^\top \beta - 2 \left\langle T, \boldsymbol{S}_A^\top \boldsymbol{S}_B \right\rangle. \end{aligned} \tag{6}$$

Suppose we are given the transport map $\boldsymbol{T}^\star$ which is optimal for the above Eq (6), but we do not know the support $\boldsymbol{S}_B$. One way to compute it is by minimizing the above with respect to $\boldsymbol{S}_B$. Hence, let's discard the constant terms in $\mathbf{s_A}$ and α. Recall $\mu_A = (\alpha, \boldsymbol{S}_A)$ and $\mu_B = (\beta, \boldsymbol{S}_B)$, so we have that minimizing $\operatorname{OT}\left(\mu_A, \mu_B, \boldsymbol{C}_{AB}\right)$ with respect to the locations $\boldsymbol{S}_B$ is same as solving

$$\min_{\boldsymbol{S}_B \in \mathbb{R}^{d \times n_B}} \mathbf{s_B}^\top \beta - \left\langle \boldsymbol{T}^\star, \boldsymbol{S}_A^\top \boldsymbol{S}_B \right\rangle \tag{7}$$

Quadratic Approximation. Cuturi and Doucet [4] show that the above minimization problem in Eq. (7) is non-convex in the locations $\boldsymbol{S}_B$, the proof of which can be found in their work. Therefore,they resort to a local quadratic approximation mentioned in equation (8), minimizing which yields the Newton update in equation (9).

$$\begin{aligned} \mathbf{s_B}^\top \beta - \left\langle \boldsymbol{T}^\star, \boldsymbol{S}_A^\top \boldsymbol{S}_B \right\rangle = {}& \left\| \boldsymbol{S}_B \operatorname{diag}\left(\beta^{1/2}\right) - \boldsymbol{S}_A \boldsymbol{T}^\star \operatorname{diag}\left(\beta^{-1/2}\right) \right\|^2 \\ & - \left\| \boldsymbol{S}_A \boldsymbol{T}^\star \operatorname{diag}\left(\beta^{-1/2}\right) \right\|^2 \end{aligned} \tag{8}$$

$$\boldsymbol{S}_B \leftarrow \boldsymbol{S}_A \boldsymbol{T}^\star \operatorname{diag}\left(\beta^{-1}\right) \tag{9}$$

This can be interpreted as follows as follows: the matrix $\boldsymbol{T}^\star \operatorname{diag}\left(\beta^{-1}\right)$ has n_B columns in the simplex Σ_{n_A} and thus post-multiplying $\boldsymbol{S}_A$ with this matrix means that we are performing convex combinations of the points in $\boldsymbol{S}_A$ with weights defined by the optimal transport map $\boldsymbol{T}^\star$.

Relation to Model Fusion. Let's come back to our algorithm where we use the weight-based alignment. Here, the locations (or the supports) are defined by the corresponding weight matrices of the layers. This bears resemblance to the update in Eq. (1) in Section 4, where the equivalent of the unknown $\boldsymbol{S}_B$ are the weights of A aligned with respect to B.

S10 Connection to the mean-field limit

HIDDEN LAYERS	% TEST ACCURACY					% RELATIVE GAP	
	MODEL A	MODEL B	PREDICTION AVG.	VANILLA AVG.	OT AVG.	VANILLA AVG.	OT AVG.
$[40, 20, 10]$	96.69	96.91	97.50	34.82	82.91	64.07	**14.44**
$[200, 100, 50]$	98.00	97.97	98.16	47.30	93.93	51.73	**4.16**
$[400, 200, 100]$	98.13	98.09	98.21	73.51	96.70	25.09	**1.45**
$[1000, 500, 250]$	98.08	98.21	98.20	78.21	97.35	20.36	**0.87**
$[2000, 1000, 500]$	98.26	98.16	98.21	85.71	97.41	12.77	**0.86**

Table S11: Relative gap of OT avg. wrt the best individual model as the width of the hidden layers increases.

Effect of layer width: Table S11 illustrates that as the width of networks increases, the gap in performance of one-shot OT averaging compared to the best individual network decreases. This also suggests a very interesting potential connection with the mean-field limit for neural networks [33].

Here, the authors show that as the size of the hidden layer goes to infinity, doing gradient descent on the network weights is equivalent to considering a probability density over the neurons in a layer, which evolves with Wasserstein gradient flow. Then given two neural networks evolving under the dynamics of Wasserstein gradient flow, fusing them into one network by Wasserstein barycenter would be a natural consideration.

We empirically show that this limit is roughly achieved in practice when the width $\approx$1000. In particular, Table S11 illustrates that as the width of networks increases, the gap in performance of one-shot averaging (with respect to the best individual network) on MNIST decreases. As a consequence, this further implies that in the setting of finite hidden layer sizes, it would help to choose the α and β in a better way than just uniform. We aim to study this aspect in a future work.

S11 Teacher-Student Fusion

We present the results for a setting where we have trained teacher and student networks, and we would like to combine the knowledge of large teacher network into the smaller student network. This is essentially reverse of the client-server setting described in Section 5.2. We consider that all the hidden layers of the teacher model M_A, are a constant $\rho\times$ wider than all the hidden layers of student model M_B. We experiment with two instances of this (a) on MNIST + MLPNET, with $\rho \in \{2, 10\}$ and (b) on CIFAR10 + VGG11, with $\rho \in \{2, 8\}$, and the results are presented in the Table S12. This leads to the mentioned model sizes (# of parameters) for both these models. Our OT average uses activation-based alignment for both the settings described in the Table S12.

Vanilla averaging can not be used due to different sizes of the networks. So, as a first baseline, we consider the performance of finetuning the model M_B. We observe that across all the settings, OT avg. + finetuning outperforms this baseline as well as the original model M_B, resulting in the desired knowledge transfer from the teacher network.

DATASET + MODEL	# params (M_A, M_B)	M_A	M_B	OT AVG.	FINETUNING M_B	OT AVG.
MNIST +	(414 K, 182 K)	98.11	97.84	95.67	98.06	**98.22**
MLPNET	(414 K, 32 K)	98.11	97.08	96.50	97.31	**97.42**
CIFAR10 +	(118 M, 32 M)	91.22	90.66	86.73	90.67	**90.89**
VGG11	(118 M, 3 M)	91.22	89.38	88.40	89.64	**89.85**

Table S12: *Compressing M_A to smaller models.* The finetuning results of each method are at their best scores across different finetuning hyperparameters (like, learning rate schedules). OT avg. has the same number of parameters as M_B.

Dataset + Model	# Student params	Finetune			LR schedule hyper-params		
		type	LR	Epochs	Enabled	LR Decay Factor	LR Decay Epochs
MNIST + MLPNET	182 K	M_B	0.01			—	—
	32 K		0.002	60	✗		
	182 K	OT Avg.	0.01				
	32 K		0.001				
CIFAR10 + VGG11	32 M	M_B	0.01			2.0	$[20, 40, 60, 80, 100]$
	3 M			120	✓	1.5	$[10, 30, 50, 70, 90, 110]$
	32 M	OT Avg.	0.01			2.0	$[20, 40, 60, 80, 100]$
	3 M						

Table S13: Hyper-parameters corresponding to the results for model compression presented in Tables S12. LR denotes the learning rate. For the MNIST + MLPNET setting a constant learning rate was employed, similar to its training procedure. While for the CIFAR10 + VGG11 setting, the learning rate schedule was also tuned, keeping in accordance with its training procedure as well. The # params column indicates the size of the resultant compressed (smaller) model.

We show that even if the smaller model M_B were to be finetuned with many different hyper-parameters, it does not outperform the performance gained by finetuning the OT average.

For MNIST + MLPNET, we did a sweep for the following set of finetuning learning rates $\{0.01, 0.002, 0.001, 0.00067, 0.0005\}$. These correspond to scaling the training learning rate by a factor $\{1, 5, 10, 15, 20\}$ respectively. Both OT average and the model M_B were finetuned for 60 epochs using these choices as a constant learning rate.

Next, for CIFAR10 + VGG11, we additionally sweep for the hyper-parameters associated with the learning rate (LR) schedule during finetuning. Since unlike the MNIST case, the original models here used a decaying learning rate schedule. The sweep was carried out for the following set of values: LR decay factor = $\{1.2, 1.5, 2\}$, LR decay epochs =

$\{[20, 40, 60, 80, 100], [10, 30, 50, 70, 90, 110], [30, 60, 90]\}$. The learning rate (LR) itself was picked from $\{0.01, 0.005, 0.0033, 0.0025\}$ corresponding to scaling the training learning rate by a factor of $\{5, 10, 15, 20\}$ respectively, as done in Section S3.1.2 before. Table S13 thus indicates the hyper-parameter choice corresponding to the best results presented in Table S12.

From sweeping on the width-ratio $\rho = 8$ for CIFAR10+ VGG11 (i.e., when the smaller model has 3M params), we found that the learning rates $\{0.01, 0.005\}$ produced the best results for both the finetuning type/methods (namely, OT average and model M_B). Thus, while sweeping on the width-ratio $\rho = 2$ for CIFAR10+ VGG11 (i.e., when the smaller model has 32M params), we reduce the hyper-parameter space by restricting the learning rate from this set $\{0.01, 0.005\}$, while still sweeping the other sets of hyper-param values, in order to save on resource costs.

Besides these best results for each method, we find that even under the same hyper-parameter configuration, finetuning OT average leads to a better performance than finetuning the smaller model M_B across multiple runs, for majority of the hyper-parameter settings. Overall, we conclude that the OT average and then finetuning successfully allows to transfer the performance from a bigger model into a smaller one.

S12   Results for distillation

In this section, we present the results in relation to distilling the knowledge of a bigger model M_A into a smaller model M_B. Here, we consider that one already has a trained version of both the models and we are interested in boosting the performance of the smaller model.

We discuss two ways of approaching this problem. One is to consider the OT average of the individual models and then finetune. The other option is to use distillation [8], where we augment the loss during finetuning with a term that essentially encourages the student's (smaller model) logit distribution (smoothed) to be close that of the teacher's (bigger model) logit distribution. The smoothing is done by raising the temperature (T) in the final softmax. This loss term from distillation gets is weighted by a factor of γ and a factor of $1 - \gamma$ is given to the usual finetuning loss.

The main drawback of distillation is that it requires searching for the optimal values of these hyper-parameters: temperature T and loss-weighting factor γ. As evident from the Tables S16 and S17, even for MNIST+ MLPNET, depending on the size of the smaller model, the optimal hyper-parameter values can be quite different. This can be prohibitive when dealing with larger models or datasets.

Nevertheless, we compare the performance of both these approaches in Tables S14 and S15, for the setting of MNIST+ MLPNET with width-ratio $\rho = 2, 10$ (the ratio of hidden layers sizes of the larger to the smaller model) respectively. For distillation, we consider three possible initializations for the smaller (student) model: random, model M_B, and OT average of models M_A, M_B.

M_A	M_B	PREDICTION AVG.	OT AVG.	FINETUNING M_B	OT AVG.	DISTILLATION RANDOM	M_B	OT AVG.
98.11	97.84	98.10	95.49	98.04	98.19	98.18	98.22	**98.30**
		Mean across distillation temperatures				98.13	98.17	**98.26**

Table S14: *Compressing the bigger teacher model M_A to half its size ($\rho = 2$). The distillation scores for each student network initialization are taken for its best hyperparameter values. Both finetuning and distillation were run for 60 epochs using SGD with the same hyperparameters. Each entry has been averaged across 4 seeds.*

M_A	M_B	PREDICTION AVG.	OT AVG.	FINETUNING M_B	OT AVG.	DISTILLATION RANDOM	M_B	OT AVG.
98.11	97.08	98.13	96.50	97.19	97.35	97.39	97.67	**97.68**
		Mean across distillation temperatures				97.21	97.55	**97.59**

Table S15: *Compressing the bigger teacher model M_A to one-tenth of its size ($\rho = 10$). The student model for distillation is initialized in 3 possible ways: random, OT avg., and model M_B. In the first row, the distillation scores are taken at its best hyperparameter values. Both finetuning and distillation were run for 60 epochs. Each entry in the table has been averaged across four seeds.*

The choice of distillation hyper-parameters tried was: temperature $T = \{20, 10, 8, 4, 1\}$ and loss-weighting factor $\gamma = \{0.05, 0.1, 0.5, 0.7, 0.95, 0.99\}$. Tables S14 and S15 report the best scores across these hyper-parameter choices. Also, each each of the reported scores in the tables have been averaged across 4 seeds. The optimization parameters are same for both finetuning and distillation to ensure fair comparison. Namely, the learning rate $= 0.01$ and momentum $= 0.5$, and both were optimized with SGD for 60 epochs.

In terms of the results, we observe that initializing with OT average does the best in comparison to random and model M_B-based initializations. Further, we see that when the results for distillation with the other initializations are averaged across the distillation temperatures, the gain in performance pales in comparison to simply OT average and finetuning (c.f. Table S14).

TEMPERATURE	DISTILLATION INITIALIZATIONS		
T	RANDOM (γ)	M_B (γ)	OT AVG. (γ)
20	98.13 (0.05)	98.20 (0.10)	**98.26** (0.05)
10	98.15 (0.05)	98.19 (0.05)	**98.28** (0.05)
8	98.18 (0.05)	98.22 (0.05)	**98.28** (0.05)
4	98.11 (0.10)	98.21 (0.10)	**98.30** (0.05)
1	98.06 (0.05)	98.04 (0.05)	**98.17** (0.05)
Mean	98.13	98.17	**98.26**

Table S16: *Distillation results for the setting of $\rho = 2$:* Best results for various distillation initializations are shown for all the tried temperatures (T) values. The corresponding choice of the loss-weighing factor (γ) for these best scores is indicated next to them in brackets. Each of the scores has been averaged over four seeds. The cell in orange indicates the top result across all hyper-parameter settings and methods.

TEMPERATURE	DISTILLATION INITIALIZATIONS		
T	RANDOM (γ)	M_B (γ)	OT AVG. (γ)
20	97.25 (0.50)	97.61 (0.70)	**97.68** (0.70)
10	97.32 (0.70)	**97.67** (0.70)	97.65 (0.70)
8	97.38 (0.50)	**97.67** (0.70)	97.65 (0.70)
4	97.39 (0.70)	97.53 (0.70)	**97.57** (0.99)
1	96.73 (0.05)	97.28 (0.95)	**97.40** (0.95)
Mean	97.21	97.55	**97.59**

Table S17: *Distillation results for the setting of $\rho = 10$:* Best results for various distillation initializations are shown for all the tried temperatures (T) values. The corresponding choice of the loss-weighing factor (γ) for these best scores is indicated next to them in brackets. Each of the scores has been averaged over four seeds. The cell in orange indicates the top result across all hyper-parameter settings and methods.

Lastly, in Tables S16 and S17, we show the detailed results for each of the distillation initializations for each of the temperature values tried. These correspond respectively to the summarized results presented in Tables S14 and S15. We observe that distillation from OT average performs the best for most of the hyper-parameter settings, as well as in terms of the overall top performance for both the width-ratio ρ settings.

To conclude, when distillation is out of the question due to resource constraints, OT average + finetuning can go a long way. While in cases where distillation is permissible, it can be advantageous to initialize with OT average when fusing a big model into a smaller one.