

SUPPLEMENTARY MATERIALS

A MIXTURE WEIGHT REGULARIZATION

Here we discuss two regularizers that can be used to force each mixture weight ξ_ℓ to be a one-hot vector.

Entropy regularization Treating each ξ_ℓ as a probability distribution, we can minimize entropy $H[\xi_\ell]$ to regularize the mixture weights vectors ξ_ℓ :

$$H[\xi] = - \sum_{k=1}^K \xi_k \log \xi_k.$$

However, this regularizer has one significant flaw: if $\xi_i < \xi_j$ for some i and j , the gradient of $H[\xi]$ will push ξ_i and ξ_j to move further apart thus preventing them from switching their order and reaching a state where $\xi_i > \xi_j$.

“Clustering” regularizer To avoid this issue, in all our experiments, we used another regularizer $R[\xi, \theta]$:

$$R[\xi, \theta] = \alpha \sum_{k=1}^K \xi_k g(\|\bar{\theta} - \theta_k\|),$$

where $\bar{\theta} = \sum \xi_k \theta_k$, g is an arbitrary monotonically increasing function with $g(0) = 0$, α is a regularization weight and θ_k are parameters of a block $f_k \in \mathcal{F}$. It is easy to see that when all θ_k are different, $R[\xi, \theta]$ is minimized for one-hot vectors ξ . At the same time, the gradients of R with respect to θ_k push templates towards the current average weight $\bar{\theta}$ with an effective force proportional to ξ_k . Intuitively, these potentially conflicting “clustering” forces getting stronger for templates contributing the most to a particular layer parameters, together with the force making mixture weights more singular, should allow the templates to self-organize.

B MODEL AND TRAINING PARAMETERS

B.1 MODEL DETAILS

Here we provide additional details about the models that were used in our experiments. The approximate number of parameters in individual model components are shown in Table 4.

Model blocks Just like in the original isometric model (Sandler et al., 2019), each residual block used the expansion factor of 6, i.e., “expansion” convolution mapped input tensor with 40ϱ channels into the intermediate 240ϱ -channel tensor, which was later projected back into the tensor with 40ϱ channels. The intermediate depthwise convolution used the kernel size of 3.

Output adapter Output adapter was chosen to be a sequence of two nonlinear kernel-1 convolutions with an average pooling layer in between. Assuming that the first convolution produces a tensor with c_1 channels and the second convolution produces a tensor with c_2 channels, the total number of parameters in this part of the model was approximately $40\varrho c_1 + c_1 c_2 + c_1 + c_2$. For larger networks, we used $c_1 = 960$ and $c_2 = 1280$, while for smaller networks we chose $c_1 = 128$ and $c_2 = 256$.

Logits layer Final fully-connected layer mapped embedding vectors produced by the output adapter into the final predictions. It used about $c_2 C + C$ parameters with C being the total number of labels.

B.2 TRAINING PARAMETERS

In all our experiments, we used RMSPROP optimizer. In smaller scale experiments, the learning rate was chosen to be either 0.02, or 0.04. The dropout keep probability and the weight decay were

80% and 10^{-5} correspondingly. In all of our experiments, we used exponential moving averages of all trained variable for inference.

All datasets that we experimented with are available in `tensorflow datasets`. For the `airplane` dataset, we used an 80% of the total number of labeled samples for the training set and 20% of samples for the test set.

C SINGLE-TASK LEARNING

In Section 4.2, we explored single-task training of networks with mixture modules. Model signatures that we discovered in the process contained multiple repeated occurrences of the same template with the number of repetitions increasing towards the tail of the model. But are these discovered signatures optimal and if so, how much better are they compared to other possible signatures? Here we discuss experiments conducted for several hand-designed model signatures and study how model performance changes if we perturb the sequence of modules and perform inference using a different order of computational blocks compared to the one the models have been trained with.

C.1 MODELS WITH DIFFERENT SIGNATURES

In all experiments discussed here, we train and evaluate classification models on the CIFAR-100 dataset. We explore models with different signatures: **(a)** *sequential signatures* $(f_K^{L/K}) \circ \dots \circ (f_1^{L/K})$ repeating each of K templates L/K times, where L is the number of layers; **(b)** *cyclic signatures* corresponding to $(f_K \circ \dots \circ f_1)^{L/K}$, **(c)** *random signatures*, and **(d)** networks with mixture modules. In all experiments, we chose $\varrho = 0.6$ and a smaller output adapter (see Appendix B.1). Specific model and training parameters chosen for performing these experiments are outlined in Appendix B. Experimental results can be found in Table 5 and Figure 4 shows validation accuracies for different signature choices, different numbers of modules K in \mathcal{F} and different numbers of layers L . While we did not have an opportunity to gather sufficient statistics for all of these experiments, reproducing individual experiments we observed accuracy fluctuations with a standard deviation of approximately 0.5.

As one would expect, we see that the model accuracy grows monotonically both with K and L . Sequential signatures are seen to outperform models with cyclic and random signatures. At the same time, models with mixture weights appear to have an accuracy close to that of sequential models. Examples of discovered mixture weights are shown in Figure 3. After studying multiple such signatures, we observed that they tend to repeat the same modules in contiguous uninterrupted sequences and the number of repetitions grows towards the end of the model. In other words, early modules use fewer repetitions. Interestingly, even in the absence of regularization, the mixture weights Ξ frequently approach a collection of one-hot vectors, i.e., $\max_i(\xi_\ell)_i \approx 1$ for most layers ℓ .

We also conducted experiments with modules containing blocks that accept both the output of the previous layer and the original input fed into the first block (output of the input adapter). Here the difference between different signatures shown in Table 6 is even more dramatic.

Table 4: Approximate number of parameters used in each component of the model: (a) 2000ϱ parameters in *input adapter*, (b) $40\varrho c_1 + c_1 c_2 + c_1 + c_2$ in the *output adapter* and (c) $c_2 C + C$ in the *logits layer*. Here ϱ is the model depth multiplier (affecting the number of channels), C is the number of classes and (c_1, c_2) are the numbers of output channels of two convolutions used in the output adapter.

	Input Adapter	Block with SE	Block without SE	Output Adapter	Logits
Smaller ($\varrho = 0.6$)	1200	20000	9000	37000	$256C$
Larger ($\varrho = 1$)	2000	52000	22000	$1.27 \cdot 10^6$	$1280C$

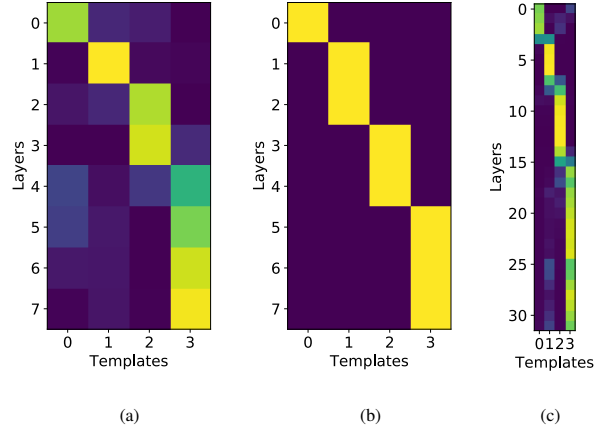


Figure 3: Examples of mixture-weight model signatures obtained with 4 modules and 8 layers: (a) mixture weights mid-training without any extra regularizers; (b) mixture weights obtained with the “clustering” regularizer; (c) mixture weight for a 16-layer model with 4 templates.

Table 5: Comparison of validation accuracies for networks with fixed and trainable signatures on the CIFAR-100 dataset. We compare different types of signatures with a particular number of layers L and number of templates K : (a) cyclic signatures $[(1, 2, 1, 2)]$, (b) sequential signatures $[(1, 1, 2, 2)]$, (c) random signatures (obtained by randomly shuffling a sequential or repeated signature) and (d) mixture-weight models. Top results (and those smaller by at most 0.5) are highlighted.

K	L	Signature			
		Cyclic	Sequential	Random	Mixture
2	4	60.6	61.0	60.5	61.1
	8	61.6	63.2	61.7	62.5
	16	62.4	64.1	64.1	63.4
	32	63.7	65.2	64.2	64.9
4	8	64.0	65.1	65.5	64.1
	16	63.3	66.7	65.1	66.4
	32	64.9	65.5	63.7	66.2
8	16	65.3	67.3	66.5	67.0
	32	66.2	67.4	66.6	66.4

Table 6: Comparison of validation accuracies for the CIFAR-100 dataset in models where each block receives the output of the previous layer and the original input (output of the input adapter). We compare different types of signatures with a particular number of layers L (12 or 16) and 4 templates. We consider: (a) cyclic signatures $[(1, 2, 1, 2)]$, (b) sequential signatures $[(1, 1, 2, 2)]$, and (c) mixture-weight models.

K	L	Signature		
		Cyclic	Sequential	Mixture
4	12	65.7	67.0	67.3
	16	64.9	67.6	67.6

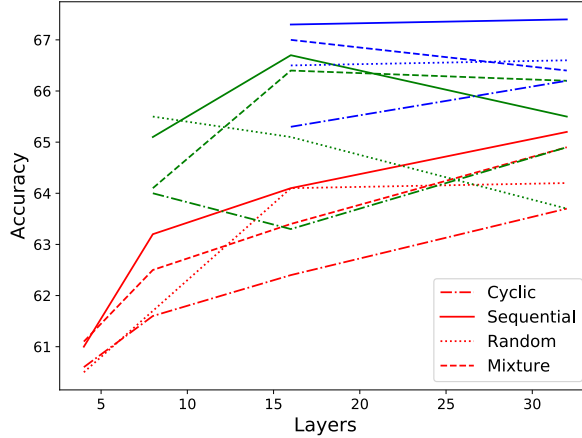


Figure 4: Visualization of results with fixed and trainable signatures on CIFAR-100 presented in Table 5: validation accuracies for signatures with 2 templates (red), 4 templates (green) and 8 templates (blue).

D MULTI-TASK LEARNING

Here, we summarize multi-task training results obtained for a set 5 supervised tasks trained on: IMAGENET, CIFAR-100, PLACES365, SUN397 and FOOD101 datasets. In our experiments, we compared the following multi-task training approaches:

1. **Simple baseline:** A more straightforward approach, in which all model blocks are conventional model layers that do not use any parameter sharing (across layers). Here input adapter, model body and the output adapter were shared by all models solving different tasks and only the logits layers were trained independently for every head. Since model activations may differ for different tasks, we also accumulated per-task batch normalization moving means (adding thousands of parameters for each task on top of parameters for the final logits).
2. **Baseline with linear patches:** A similar approach inspired by Mudrakarta et al. (2019), where batch normalization β and γ variables were not shared and trained for each task independently.
3. **Model with mixture weights:** Model with mixture weights, where all mixture weights ξ_i were initialized randomly and all batch normalization β and γ were also linearly mixed using these mixture weights⁷.
4. **Model with mixture weights and linear patches:** Same as the previous model, except β and γ variables were not shared here, but were trained for each layer and each task independently.

The final comparison results are presented in Table 7.

While it appears that additional templates result in overall performance improvement, there are reasons to believe that the model does not reach a true optimum. Notice that the model with linear patches can be viewed as a special case of a network with mixture modules and linear patches, where the total number of templates K is the same as the number of layers L and all mixture weights for all tasks satisfy $(\xi_\ell^T)_k = \delta_{\ell,k}$, where ξ_ℓ^T are the mixture weights for layer ℓ for task T . In our experiments where we chose $K = L = 16$ (see Table 7), the final training loss turned out to be higher than that of the conventional model with linear patches, which suggests that the training procedure fails to identify a solution in which every layer uses a unique template. As a sanity check, we verified that by initializing mixture weights ξ_ℓ at $(\xi_\ell)_k \approx \delta_{\ell,k}$, the resulting model matches baseline model performance (with linear patches) much better and the final cross-entropy losses are essentially identical. Another indication that our model does not currently reach the optimum is

⁷moving means and variances were not shared

the fact that the models with mixture weights and linear patches appear to perform worse than the model with mixture weights alone. Improving modular network training will be one of priorities of our future work.

Table 7: Comparison of models co-trained on 5 datasets. Models marked with “diag” used mixture weight initialization with $(\xi_\ell)_i \approx \delta_{i,\ell}$. Reported values correspond to validation accuracies with the training accuracy also specified for IMAGENET and PLACES datasets. Training accuracy for other datasets reached 99.9% in all experiments. Because of this, the final comparison of model performance may need to rely on IMAGENET accuracy and the final aggregated cross-entropy loss L_{CE} . Here $K = |\mathcal{F}|$ is the number of templates in \mathcal{F} and models with “linear patches” (Mudrakarta et al., 2019) are models that maintain distinct per-task batch normalization β and γ variables.

Model	Dataset					L_{CE}
	CIFAR	FOOD	IMAGENET	PLACES	SUN	
Simple baseline	77.0	70.4	61.1 (70.4)	48.6 (56.5)	61.3	4.59
Baseline with patches	76.8	71.3	61.8 (71.8)	49.2 (56.4)	60.0	4.48
Mixture weights (MW) ($K = 32$)	76.6	71.9	63.1 (73.5)	48.7 (60.0)	58.8	4.29
MW with patches ($K = 16$)	74.9	70.3	60.9 (71.0)	48.3 (56.2)	58.8	4.6
MW with patches ($K = 32$)	74.9	70.3	62.7 (72.7)	48.8 (57.8)	58.0	4.4
MW with patches (diag. $K = 16$)	77.3	71.1	62.1 (71.6)	48.9 (57.1)	59.9	4.48
MW with patches (diag. $K = 32$)	77.7	71.7	62.3 (72.6)	49.3 (57.9)	60.0	4.4

E UNSUPERVISED DOMAIN ADAPTATION

E.1 RELATED WORK

Unsupervised domain adaptation (UDA) methods aim to learn a transformation of unlabeled target domain such that its distribution is close to the source domain distribution according to some metric of similarity. There has been proposed a significant number of UDA methods. Some methods Tzeng et al. (2017); Long et al. (2017); Tzeng et al. (2014); Long et al. (2015) minimize the discrepancy between the source and target domain distributions by introducing an additional domain classifier; the target domain features are transformed so that the domain classifier cannot distinguish the domains accurately. Other works have explored feature moments matching Zellinger et al. (2017); Peng et al. (2019) and second-order correlation Sun et al. (2016); Peng & Saenko (2018) between source and target domain. Recent domain adaptation papers Zhu et al. (2017); Hoffman et al. (2018); Liu et al. (2017) use GANs to minimize the pixel-level domain shift. Some recent papers such as Chang et al. (2019); Wang et al. (2020) show the power of batch normalization for unsupervised domain adaptation. In fact, the concurrent work by Wang et al. (2020) shows that fine-tuning of domain-specific batch normalization parameters alone gives promising results. We argue that our domain adaptation methods based on a compositional model described in Section 4.4 can be viewed as an extension of the domain-specific batch normalization since mixture weights affect the learnable parameters β and γ of batch normalization.

E.2 SETUP

Digits For the experiment on digits datasets, we pretrained a classifier jointly on MNIST, Corrupted MNIST (shear), Corrupted MNIST (scale) and SVHN, and used USPS dataset as the target domain. The model contained 4 templates and 16 layers, used dropout with factor 0.3, with a linearly decaying learning rate from 2×10^{-3} to 2×10^{-5} ; convolutional kernel weights regularization was set to 10^{-5} . The model was trained for 2 epochs, at which point the accuracy over 97% was reached for all source datasets except SVHN (over 92%). For the compositional models, we then copied the values of mixture weights of Corrupted MNIST (shear) to the target mixture weights since it gave the best source-only accuracy among source domains. We fine-tuned our model by minimizing the DA loss for 10 epochs. The total number of parameters in the classifier is ≈ 135000 .

DomainNet For the experiment with DomainNet dataset, we selected 100 classes that occur most frequently in the dataset, and reshaped the images to 128×128 . We used infograph, clipart, painting,

real and quickdraw as source domains, and sketch as target domain. For the baseline methods, we use an isometric model with 12 blocks that output tensors of shape $16 \times 16 \times 40$, and 12 templates and 16 blocks for the compositional models. We used the same learning rate, dropout and kernel regularization parameters as in the Digits experiment. We pretrained the classifier on the source domains for 20 epochs and fine-tuned them for another 20 epochs. For the compositional models, we copied the mixture weights from the painting domain to the target domain as it resulted in the best source-only accuracy. The classifier contained ≈ 340000 parameters in total.

E.3 ADVERSARIAL DISCRIMINATIVE DOMAIN ADAPTATION

After pretraining the model on source domain and copying the mixture weights, we started training the domain discriminator and target mixture weights simultaneously in an adversarial fashion. The discriminator was trained with Adam optimizer and a polynomially decaying learning rate from 2×10^{-4} to 10^{-5} . We compared the results of the compositional ADDA with those of the conventional ADDA model without weight sharing. The conventional baseline used a standard isometric model with the number of layers equal to number of templates in the compositional case and contained roughly same number of learnable parameters as our model. Our discriminator is a 3-layer CNN with a binary cross-entropy loss.

E.4 MOMENT MATCHING DOMAIN ADAPTATION

In this setup, we used an isometric model with 12 templates for the baseline method and a compositional model with 12 templates and 16 layers for compositional moment matching method. We pretrained our models for 20 epochs and finetuned them on target domain for 15 epochs. For the compositional model, we copied the mixture weights from the painting domain for before finetuning. For the moment matching, we used Adam optimizer with learning rate decaying from 10^{-4} to 10^{-5} . Both models used dropout with drop probability 0.3 and kernel regularization with weight 10^{-5} .