Side-Tuning: Network Adaptation via Additive Side Networks (Supplementary)

Jeffrey O. Zhang[1*] Alexander Sax[1*] Amir Zamir[1,2]
Leonidas Guibas[2,3] Jitendra Malik[1,3]
[1]University of California, Berkeley [2]Stanford University [3]Facebook AI Research
http://jozhang97.github.io/sidetuning

1. Appendix

1.1. Table of Contents

We provide the following material in the appendices:

1.2. Qualitative Results in Incremental Learning

We show predictions for some fixed set of randomly selected images throughout training.

*Equal Contribution

Side-Tuning

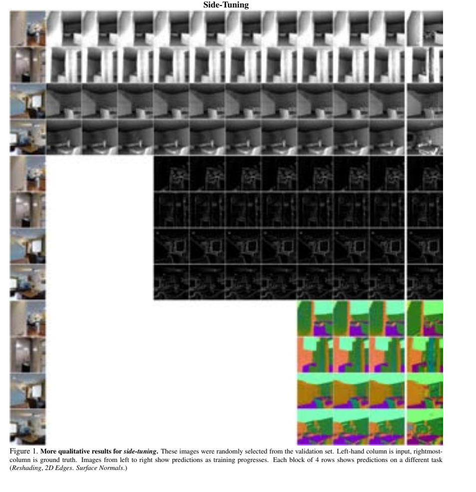


Figure 1. **More qualitative results for *side-tuning*.** These images were randomly selected from the validation set. Left-hand column is input, rightmost-column is ground truth. Images from left to right show predictions as training progresses. Each block of 4 rows shows predictions on a different task (*Reshading*, *2D Edges*. *Surface Normals*.)

Elastic Weight Consolidation (EWC)

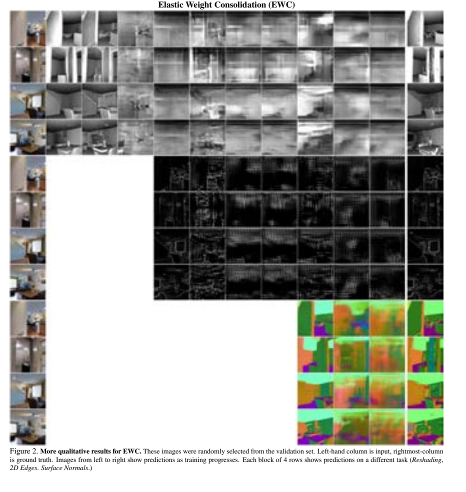


Figure 2. **More qualitative results for EWC.** These images were randomly selected from the validation set. Left-hand column is input, rightmost-column is ground truth. Images from left to right show predictions as training progresses. Each block of 4 rows shows predictions on a different task (*Reshading*, *2D Edges*. *Surface Normals*.)

Parameter Superposition (PSP)

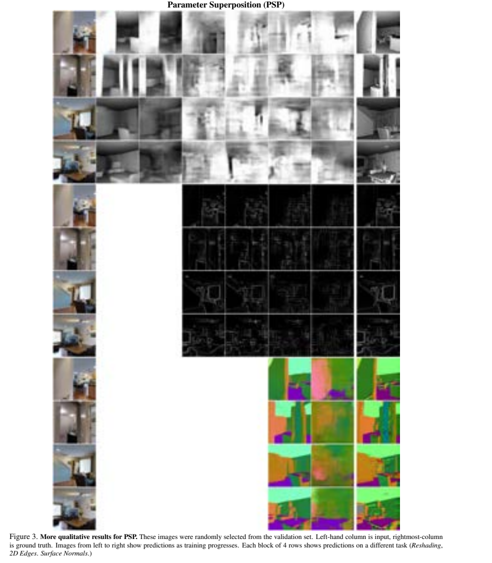


Figure 3. **More qualitative results for PSP.** These images were randomly selected from the validation set. Left-hand column is input, rightmost-column is ground truth. Images from left to right show predictions as training progresses. Each block of 4 rows shows predictions on a different task (*Reshading*, *2D Edges*. *Surface Normals*.)

Independent Fine-Tuned ResNet-50

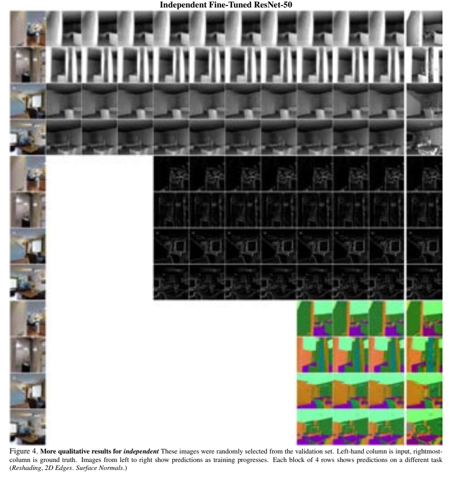


Figure 4. **More qualitative results for *independent*** These images were randomly selected from the validation set. Left-hand column is input, rightmost-column is ground truth. Images from left to right show predictions as training progresses. Each block of 4 rows shows predictions on a different task (*Reshading*, *2D Edges*. *Surface Normals*.)

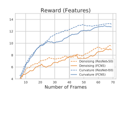
Method	Transfer Learning in Taskonomy: From Curvature (100/4M ims.) Normals (MSE ↓)	Obj. Cls. (Acc. ↑)	Transfer Learning in Taskonomy: From Obj. Class. (100) Normals (MSE ↓)	Navigation (IL): Nav. Rew. (49/4900 epi) (↑) Curvature	Denoise
Standard	0.20/0.010	24.8/63.3	0.25	4.9/9.5	2.3/9.3
Small Base	0.20/0.09	25.3/63.2	0.22	4.3/9.5	2.2/8.4
Large Side	0.21/0.11	25.3/55.6	0.23	X	X
	(a)		(b)	(c)	

Figure 5. **Effect of network size.** Modifying the network size from standard (large basae/small side). Small bases generally have a small impact on performance. For hard tasks (e.g. classification), using a deeper side network can have a large positive effect.

1.3. Additional Experiments

1.3.1 Network Size

We test the effect of base model architecture on performance and find that the small five layer convolutional network does comparable to the ResNet-50 when using features.



1.3.2 Variance in Gradients: Rectified Adam

Rectified Adam is a method introduced to deal with destructive high variance updates at the beginning of training. We tried using this for RL but found no improvements (shown in Figure 6).

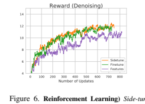


Figure 6. **Reinforcement Learning)** *Side-tuning* matches the performance of the best method when using denoising features as well.

1.3.3 Imitation Learning

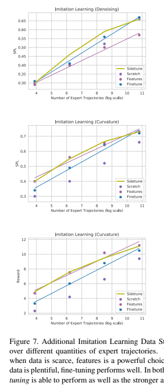


Figure 7. Additional Imitation Learning Data Study. We ablate over different quantities of expert trajectories. We observe that when data is scarce, features is a powerful choice whereas when data is plentiful, fine-tuning performs well. In both scenarios, *side-tuning* is able to perform as well as the stronger approach.

1.3.4 Extremely Few-Shot Learning

In domains with very few examples, we found that *side-tuning* is unable to match the performance of other methods. We evaluated our setup in vision transfer for 5 images from the same building, imitation learning given 5 expert trajectories.

Methods	Nav. Rew. (4 epi) (↑) Curvature	Denoise	Loss (5 ims) (↓) Curvature to Normals
Finetune	-0.1	-1.2	0.35
Features	0.4	1.2	0.36
Scratch	-0.9	-0.9	0.37
Sidetune	-0.3	-1.8	0.42

1.4. Experimental Setup

For full details on our configs, please refer to ./configs in provided code.

1.4.1 Experimental Setup for Incremental Learning

Taskonomy Our data is 4M images on 12 single image tasks. The tasks that we use are the following: curvature, semantic segmentation, reshading, keypoints3d, keypoints2d, texture edges, occlusion edges, distance, depth, surface normals, object classification and autoencoding. The tasks were chosen in no particular special order. Our base model and side model are ResNet-50s. We pretrain on curvatures. Then, we train each task for three epochs before moving on to the next task. We use cross entropy loss for classification tasks (semantic segmentation and object classification), L2 loss for curvature and L1 loss for the other tasks. We use Adam optimizer with an initial learning rate of 1e-4, weight decay coefficient of 2e-6, gradient clipping to 1.0, and batch size of 32. We evaluate our performance on a held out set of images, both immediately after training a specific task, and after training of all the tasks are complete.

iCIFAR We start by pretraining a model on CIFAR 10 (from https://github.com/akamaster/pytorch_resnet_cifar10). Then we partition CIFAR100 into 10 distinct sets of 10 classes. Then, we train for 4 epochs on these tasks using Adam optimizer, learning rate of 1e-3, batch size of 128.

1.4.2 NLP

We train and test on the the question answering dataset SQuAD2.0, a reading comprehension dataset consisting of 100,000 questions with 50,000 unanswerable questions. Both our base encoding and side network is a BERT transformer pretrained on a larger corpus. Finetuning trains a single BERT transfer. We use the training setup found at https://github.com/huggingface/pytorch-transformers (train for 2 epochs at a learning rate of 3e-5) wth one caveat - we use an effective batch size of 3 (vs. their 24) due to the

1.4.3 Experimental Setup for Habitat Experiments

We borrow the experimental setup from work to be published in October 2019:

> We use the Habitat environment with the Gibson dataset. The dataset virtualizes 572 actual buildings, reproducing the intrinsic visual and semantic complexity of real-world scenes.
>
> We train and test our agents in two disjoint sets of buildings. During testing we use buildings that are different and completely unseen during training. We use up to 72 building for training and 14 test buildings for testing. The train and test spaces comprise $15678.4m^2$ (square meters) and $1752.4m^2$, respectively.
>
> The agent must direct itself to a given nonvisual target destination (specified using coordinates), avoiding obstacles and walls as it navigates to the target. The maximum episode length is 500 timesteps, and the target distance is between 1.4 and 15 meters from the start.

This setup is shared between imitation learning and RL, which differ in the data, architecture and optimization process.

Imitation Learning We collect 49,325 shortest path expert trajectories in Habitat, 2,813,750 state action pairs. We learn a neural network mapping from states to actions. Our base encoding is a ResNet-50 and the side network is a five layer convolutional network. The representation output is then fed into a neural network policy. We train the model for 10 epochs using cross entropy loss and Adam at an initial learning rate of 2e-4 and weight decay coefficient of 3.8e-7. We initialize alpha to 0.5. Finetuning uses the same model architecture but updates all the weights. Feature extraction only uses the ResNet-50 to collect features.

RL Similarly, we borrow the RL setup from the same work.

> In all experiments we use the common Proximal Policy Optimization (PPO) algorithm with Generalized Advan- tage Estimation. Due to the computational load of ren- dering perceptually realistic images in Gibson we are only able to use a single rollout worker and we therefore decorre- late our batches using experience replay and off-policy vari- ant of PPO. The formulation is similar to Actor-Critic with Experience Replay (ACER) in that full trajectories are sampled from the replay buffer and reweighted using the first-order approximation for importance sampling.

During training, the agent receives a large one-time reward for reaching the goal, a positive reward proportional to Euclidean distance toward the goal and a small negative reward each timestep. The maximum episode length is 500 timesteps, and the target distance is between 1.4 and 15 meters from the start.

Due to this paradigms' compute and memory constraints, it would be difficult for us to use large architectures for this setting. Thus, our base encoding is a five layer convolutional network distilled from the trained ResNet-50. Our side network is also a five layer convoultional network. Finetuning is handled the same way - update all the weights in this setup. Feature extraction uses the five layer network to collect features.

1.4.4 Experimental Setup for Learning Mechanics

Low energy initialization In classical teacher student distillation, the student is trained to minimize the distance between its output and the teacher's output. In this setting, we minimize the distance between the teacher's output and the summation of the student's output and the teacher's output). The output space may have a different geometry than that of the input space and this would allow us to work with

1.5. Additional Analysis

1.5.1 Additional Evaluations on Lifelong Learning

We provide alternative perspectives and additional insights for our lifelong learning tasks.

iCIFAR In the main paper, we see that the average rank of side-tuning higher than that of PNN. We find that side-tuning can bridge this gap with a multilayer perceptron (adapter) to merge the base and side networks. This is a common practice in PNN. In Fig. 8, we see with the adapter network, the two methods are very similar when measuring classification error.

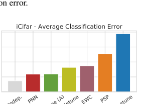


Figure 8. **Average Accuracy in iCIFAR for All Methods.** Note that the performance of Side-tune (A) is comparable to that of PNN. Side-tuning (A) using multilayer perceptron (adapter) similar to what PNN uses.

Taskonomy In the main paper, we found that the ranking of our method is better than all other methods, including PNN. By altering the connections in the PNN, we found an alternate (PNN3) that has comparable performance to side-tuning. In 9, we show all the losses normalized by single task loss (independent) as presented in [1]. The quantitative performance of our method outperforms all other methods and matches closely with that of PNN. We qualitatively show in Figures 10, 11, 12, that these methods are comparable in performance.

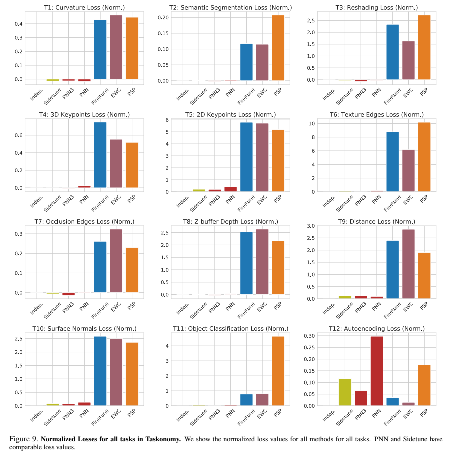


Figure 9. **Normalized Losses for all tasks in Taskonomy.** We show the normalized loss values for all methods. PNN and Sidetune have comparable loss values.

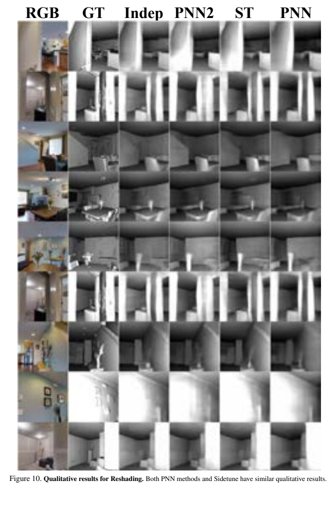


Figure 10. **Qualitative results for Reshading.** Both PNN methods and Sidetune have similar qualitative results.