SIDE-TUNING: NETWORK ADAPTATION VIA ADDITIVE SIDE NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

When training a neural network for a desired task, one may prefer to adapt a *pre-trained* network rather than start with a randomly initialized one – due to lacking enough training data, performing lifelong learning where the system has to learn a new task while being previously trained for other tasks, or wishing to encode priors in the network via preset weights. The most commonly employed approaches for network adaptation are *fine-tuning* and using the pre-trained network as a fixed *feature extractor*, among others.

In this paper we propose a straightforward alternative: *Side-Tuning*. Side-tuning adapts a pre-trained network by training a lightweight "side" network that is fused with the (unchanged) pre-trained network using a simple additive process. This simple method works as well as or better than existing solutions while it resolves some of the basic issues with fine-tuning, fixed features, and several other common baselines. In particular, side-tuning is less prone to overfitting when little training data is available, yields better results than using a fixed feature extractor, and doesn't suffer from catastrophic forgetting in lifelong learning. We demonstrate the performance of side-tuning under a diverse set of scenarios, including lifelong learning (iCIFAR, Taskonomy), reinforcement learning, imitation learning (visual navigation in Habitat), NLP question-answering (SQuAD v2), and single-task transfer learning (Taskonomy), with consistently promising results.

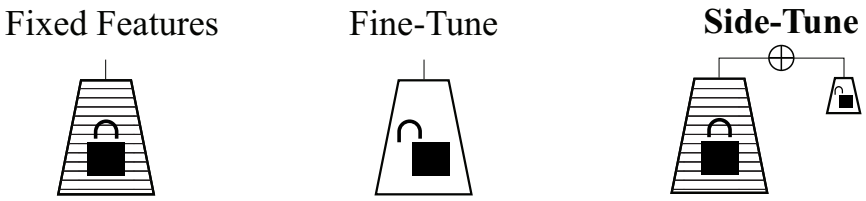


Figure 1: **The *side-tuning* framework vs the common alternatives *fine-tuning* and *fixed features***. Given a pre-trained network that should be adapted to a new task, *fine-tuning* re-trains the pretrained network's weights and *fixed feature extraction* trains a readout function with no re-training of the pre-trained weights. In contrast, *Side-tuning* adapts the pre-trained network by training a lightweight conditioned "side" network that is fused with the (unchanged) pre-trained network using a simple additive process.

1 INTRODUCTION

The goal of *side-tuning* is to capitalize on a pretrained model to better learn one or more novel tasks. By design, *side-tuning* does so without degrading performance of the base model. The framework is straightforward: it assumes access to the frozen base model $B : \mathbb{X} \to \mathbb{Y}$ that maps inputs into some representation space that is shared between the base task and the current (target) task. This representation space is flexible and could either be a latent space (e.g. in $\mathbb{R}^N$) or actual model predictions. *Side-tuning* then *learns* a side model $S : \mathbb{X} \to \mathbb{Y}$, so that the representations for the target task are

$$R(x) \triangleq B(x) \oplus S(x),$$

Method	1 Target Task		$>$ 1 Target Task
	Low Training Data	High Training Data	(Lifelong Learning)
Fixed features	✓	✗ (Information Loss)	✗ (Information Loss)
Fine-tuning	✗ (Overfitting)	✓	✗ (Forgetting)
Side-tuning	✓	✓	✓

Fixed features: No learnable parameters. Too rigid. Suffers from information loss.
Fine-tuning: Large number of learnable parameters. Suffers from overfitting.
***Side-tuning*:** Small number of (strategically placed) learnable parameters.

Figure 2: **Advantages of *side-tuning* vs. alternatives**. Fixed features cannot adapt to new information, while fine-tuning adapts too easily and forgets old information. *Side-tuning* is a simple method to address these limitations.

for some combining operation $\oplus$. We use a learned alpha-blending, $a \oplus b \triangleq \alpha a + (1-\alpha)b$ for this purpose (other options are discussed in Section 3.3).

Certain pre-set curricula of α reduce the side-tuning framework to: fine-tuning, feature extration, and stage-wise training (see Fig. 3, right). Hence those can be viewed as special cases of the general side-tuning framework. Also, other curricula suggest (e.g.) a *maximum a posteriori* estimator that integrates the $B(x)$ prior with the evidence from $S(x)$.

Side-tuning is an example of an *additive learning* approach as it adds (strategically placed) parameters for each new task. Fixed feature extraction would be a simple example of an additive approach with *zero* new parameters. As a result, fixed features are don't adapt the base network over the lifetime of the agent. A number of existing approaches address this by learning new parameters (the number of which scales with the size of the base network) for each new task (e.g. Rusu et al., 2016). Unlike these approaches, *side-tuning* places no constraints on the structure of the side network, allowing the parameters to be strategically allocated. In particular, *side-tuning* can use tiny networks when the base requires only minor updates. By adding fewer parameters per task, *side-tuning* can learn more tasks before the model grows large enough to require parameter consolidation.

These approaches stand in contrast to most existing methods for incremental learning, which do not increase the number of parameters over time and instead gradually fill up the capacity of a large base model. For example, fine-tuning updates all the parameters. A large body of constraint-based methods focus on how to regularize these updates in order to prevent inter-task interference (Cheung et al., 2019). *Side-tuning* does not require such regularization since the *additive* structure means inter-task interference is not possible.

We compare *side-tuning* to alternative approaches on both the iCIFAR and Taskonomy datasets. iCIFAR consists of ten distinct 10-class image classification problems. Taskonomy covers multiple tasks of varied complexity from across computer vision (surface normal and depth estimation, edge detection, image 1000-way classification, etc.). On these datasets, *side-tuning* uses side networks that are much smaller than the base. Consequently, even without consolidation, *side-tuning* uses fewer learnable parameters than the alternative methods.

This remarkably simple approach deals with the key challenges of incremental learning. Namely, it does not suffer from either:

- **Catastrophic forgetting**: which is the tendency of a network to abruptly lose previously learned knowledge upon learning new information. We show this in Section 4.2.1.
- **Rigidity**: where networks become increasingly unable to adapt to new problems as they accrue constraints from previous problems. We explore this in Section 4.2.2.

Side-tuning avoids these problems while remaining highly performant, which we demonstrate in Section 4.2.3.

2 Related Work

Broadly speaking, network adaptation methods either overwrite existing knowledge (*substitutive* methods) or save it and add new parameters (*additive* learning). In incremental (lifelong) learning, substitutive methods like fine-tuning are at risk of forgetting early tasks. To prevent forgetting,

existing methods add non-interference constraints that eventually slow down learning or they force tasks to be independent which prevents reusing knowledge. *Side-tuning* is an *additive* approach that performs well and scales well, and, by design, does not suffer from the aforementioned problems. We show this experimentally on various tasks and datasets, including iCIFAR (Rebuffi et al., 2016b), Habitat (Savva et al., 2019), SQuAD v2 (Rajpurkar et al., 2018), and Taskonomy (Zamir et al., 2018). In the remainder of this section we overview *sidetuning's* connection to related fields.

Network Adaptation modifies an existing network to solve a single new task. The most common approach is to update some or all of the network weights (fine-tuning), possibly by adding constraints (Kirkpatrick et al., 2016). Other approaches freeze the weights and modulate the output by learning additional task-specific parameters. An economical approach is to use off-the-shelf-features with one or more readout layers (Razavian et al., 2014). Other approaches use custom connection schema (Rusu et al., 2016). Mallya & Lazebnik (2018) modulate the output by applying learned weight masks. These approaches, like *side-tuning*, are custom examples of *additive learning*.

Incremental learning has the objective of learning a sequence of tasks $\mathcal{T}_1, ..., \mathcal{T}_m$ and, at the end of training, performing well on the entire set. The sequential presentation creates two problems for neural networks. The first is *catastrophic forgetting* and the second is *learning speed*, which should not slow down as more tasks are added. Because of these issues (and scaling), not every network adaptation approach lends itself to incremental learning. Standard incremental learning approaches avoid catastrophic forgetting by imposing constraints how the parameters are updated. Cheung et al. (2019) relegates each task to approximately orthogonal subspaces. Schwarz et al. (2018); Kirkpatrick et al. (2016); Li & Hoiem (2016) add a parameter regularization term per task. Imposing constraints tends to slow down learning on later tasks (*intransigence*, Chaudhry et al. (2018)) while making tasks independent ignores the possibility for useful transfer from relevant previous tasks. *Additive* methods in general, and *side-tuning* in particular, have the advantage that they do not suffer from catastrophic forgetting and are capable of transfer. However, additive methods have not been much explored because it is assumed that they either have poor performance or scale poorly. We show that *side-tuning* has good performance and scaling, and demonstrate the *additive* advantages experimentally; using the iCIFAR and Taskonomy datasets.

Meta-learning seeks to create agents that rapidly adapt to new problems by first training on tasks sampled from a standing distribution of tasks. *Side-tuning* is fundamentally compatible with this formulation and with existing approaches (e.g. Finn et al., 2017) as *side-tuning* can be adopted as a low-level adaptation mechanism. Moreover, recent work suggests that these approaches work primarily by feature adaptation rather than rapid learning (Raghu et al., 2019), and feature adaptation is also the motivation for our method.

Residual Learning exploits the fact that it is sometimes easier to approximate a difference rather than the original function. This has been successfully used in ResNets (He et al., 2016) and robotics, where residual RL (Johannink et al., 2018; Silver et al., 2018) learns a single task by first training a coarse policy (e.g. behavior cloning) and then training a residual network on top (using RL). *side-tuning* is an example of residual learning.

Additive Learning in Other Literature. Concepts similar to additive learning have been studied in a number of fields. For instance, developing infants are hypothesized to learn separate, discontinuous, and context-dependent perception systems during development (Adolph et al., 2011; Kretch & Adolph, 2013). Adults are able to rapidly learn new affordances, but only when those are minor updates to familiar, well-practiced systems (Cole et al., 2013). On a more fine-grained scale, there are areas of functional specificity within the brain (Kanwisher, 2010), including wholly separate pathways where output is mutually conditioned on one another (Schenk & McIntosh, 2010).

3 The Side-tuning Framework

Side-tuning learns a side model $S(X)$ and combines this with a base model $B(x)$ so that the representations for the target task are computed as $R(x) \triangleq B(x) \oplus S(x)$.

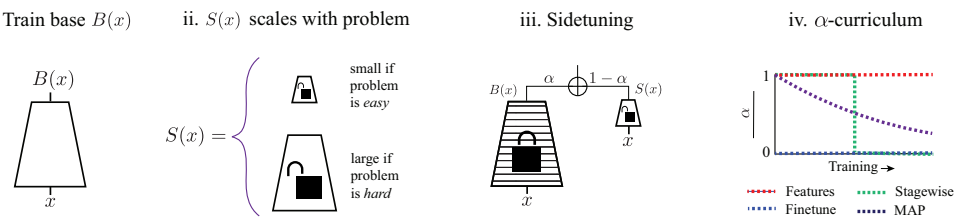


Figure 3: **Mechanics of *side-tuning*.** (i) *Side-tuning* takes some core network (B) and adapts it to a new task by (ii) adapting a side network. (iii) Shows the connectivity structure when using *side-tuning* along with alpha-blending. (iv) Existing adaptation methods turn out to be special cases of *side-tuning*. In particular: fine-tuning, feature extraction, and other approaches are *side-tuning* with a fixed curriculum on the blending parameter α.

3.1 Base Model $B(x)$

The base model $B(x)$ provides some core cognition or perception, and we put no restrictions on how $B(x)$ is computed. We never update $B(x)$, and in our approach it has zero learnable parameters. In general $B(x)$ could be nonparametric, and it might not be optimized for any particular task. We consider several choices for $B(x)$ in Section 4.4, but the simplest choice is just a pretrained network.

3.2 Side Model $S(x)$

Unlike the base model, the side network $S(x)$ is updated during training; learning a residual that we apply on top of the base encoding. Iteratively learning residuals for a *single* task is known as *gradient boosting* (see Section 4.4 for a comparison). *Side-tuning* is instead focused on learning *multiple* tasks; it builds off of (and generalizes, see Section 3.3) the *subsumption* architecture (Brooks, 1986), which is a method for learning many tasks by combining many specific behaviors.

One crucial component of the framework is that *the complexity of the side network can scale to the difficulty of the problem at hand.* When the base is *relevant* and requires only a minor update, a very small network can suffice. Section 4.4 explores the effect of network size, how that changes with the choice of base and target tasks.

While the side network can be initialized using a variety of methods, we initialize the side network with a copy of the base network. When the forms of the base and side networks differ, we initialize the side network with weights distilled from the base network using knowledge distillation (Hinton et al., 2015). We test alternatives in Section 4.4.

3.3 Combining Base and Side Representations

The final *side-tuning* representation is a combination, $B(x) \oplus S(x)$. What should $\oplus$ be?

Side-tuning admits several options for this combination operator. Choosing $\max$ yields the subsumption architecture. Concatenation and summation are other viable choices. We observe that alpha blending, $a \oplus b \triangleq \alpha a + (1-\alpha)b$, works well in practice. Alpha blending preserves the dimensions of the inputs and is simpler than concatenation. In fact, concatenation followed by a channel-collapsing operation (e.g. 1x1 convolution) is a strict generalization of alpha-blending.

While simple, alpha blending is expressive enough that it encompasses several common transfer learning approaches. As shown in Figure 3 and when the side network is the same as the base, *side-tuning* is equivalent to feature extraction when $\alpha = 1$. When $\alpha = 0$, *side-tuning* is instead equivalent to fine-tuning. If we allow α to vary during training (which we generally do), then switching α from 1 to 0 is equivalent to the common (stage-wise) training curriculum in RL where a policy is trained on top of some fixed features that are unlocked partway through training.

Another notable curriculum is $\alpha(N) = \frac{k}{k+N}$ for $k > 0$ (hyperbolic decay). In this curriculum, α controls the weighting of the prior ($B(x)$) with the learned estimate ($S(x)$), and the weight of the evidence scales with the amount of data. This curriculum is suggestive of a *maximum a posteriori* estimate and, like the MAP estimate, it converges to the MLE (fine-tuning, $\alpha = 0$).

Finally, α can treated as a learnable free parameter that determines how heavily to weight the base model. In practice, we find that the value of α correlates with task relevance (see Section 4.4).

3.4 SIDETUNING FOR INCREMENTAL LEARNING

Network adaptability is the sole criterion if our only concern is raw performance on a *single* target task. In reality we often care about the performance on both the current and previous tasks. This is the case for *incremental learning*, where we want an agent that can learn a sequence of tasks $\mathcal{T}_1, ..., \mathcal{T}_m$ and, at the end, is capable of reasonable performance across the entire set. Thus, catastrophic forgetting becomes a major issue.

In our experiments we dedicate one new side network to each new task and train it independently of the earlier side networks. In principle, learning of new tasks can benefit from all the side networks learned in previous tasks (i.e. the nth task can use all $n-1$ previous tasks). Since we do not make use of this available information, our results should be considered as a lower bound on *side-tuning* performance. We show that this simple approach provides a strong baseline for incremental learning—outperforming existing approaches in the literature while using fewer parameters on more tasks (in Section 4.2).

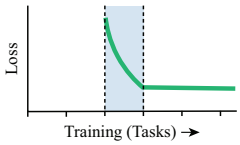


Figure 4: **Theoretical *side-tuning* learning curve.**

Side-tuning takes an additive learning approach to incremental learning, which means that already-learned components are never updated and performance across the whole set can only increase as the agent sees more tasks. This monotonicity is the key property of the *additive* family of algorithms. It is worth repeating that there is *no catastrophic forgetting in additive learning* and a typical learning curve for one of the tasks is shown in Figure 4. Furthermore, because our implementation of *side-tuning* treats problems independently of their order in the sequence (always using the fixed base and one side network), *side-tuning* incurs no rigidity during training. We show this in Section 4.2.2.

3.5 ASYMPTOTIC CONSISTENCY AND BIAS/VARIANCE

When minimizing estimation error there is often a tradeoff between the bias and variance contributions (Geman et al., 1992). Choosing between feature extraction or fine-tuning exemplifies this dilemma. Feature extraction ($\alpha = 0$) locks the weights and corresponds to a point-mass prior that, unless the weights are already optimal, yields a very biased estimator. In fact, the estimator allows no adaptation to new evidence and is *asymptotically inconsistent*. On the other hand, fine-tuning ($\alpha = 1$), is an uninformative prior yielding a low-bias high-variance estimator. With enough data, fine-tuning can produce better estimates, but this usually takes more data than feature extraction. *Side-tuning* addresses both the these problems. It reduces variance by including the fixed features in the representation, and it is *consistent* because it allows updating via the residual side network.

3.6 PERCEPTUAL REGULARIZATION AND CATASTROPHIC FORGETTING

While α provides a way to control the importance of the prior, another natural approach for enforcing a prior is to penalize deviations from the original feature representation. Typically, it is easier to specify meaningful explicit priors on outputs (e.g. L2 for pixels) than on the latent representations, which can be difficult if not impossible to interpret. As long as the decoder $D : \mathbb{Y} \to \mathbb{A}$ is differentiable, any distance measure on the outputs can be *pulled back* through the decoder and into the latent space. This induced distance d_D on the latent representations is called the *pullback metric* in differential geometry, and in deep learning it is called the *perceptual loss* (Johnson et al., 2016). This may be a useful method for knowledge transfer when (i) the previous task is *relevant* to the new task and (ii) there is limited training data. A recent successful application of this approach would be the auxiliary losses in *GPT* (Radford et al., 2018), though we did not find it effective.

Perceptual regularization is often used to dampen *catastrophic forgetting*. For example, Elastic Weight Consolidation uses a diagonalized second-order Taylor expansion of the expectation of the pullback metric. *Learning Without Forgetting* uses a decoder-based approach that can be interpreted as jointly updating both the base network and the pullback metric. We show that such regularization does not fully address the problem of catastrophic forgetting (Section 4.2.1). *Side-tuning* avoids catastrophic forgetting by design (as the base network is never updated).

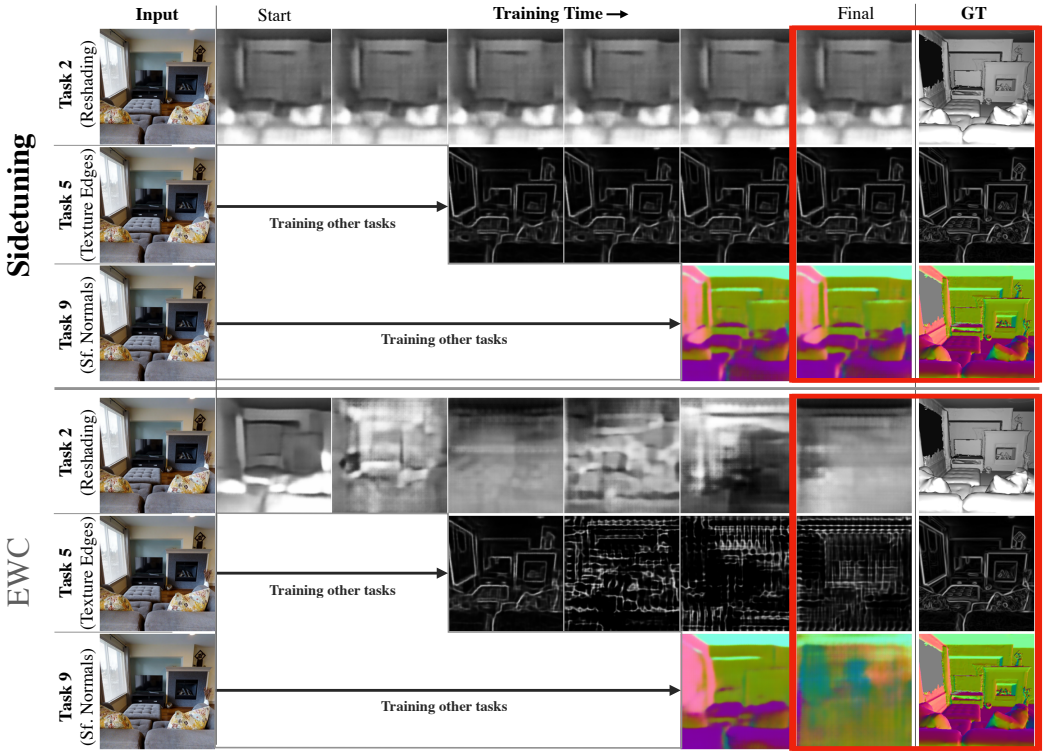


Figure 5: ***Side-tuning* does not forget in incremental learning.** Qualitative results for incremental learning on Taskonomy with additive learning (*side-tuning*, top 3 rows) and constraint-based learning (EWC, bottom 3 rows). Each row contains results for one task and columns show how predictions change over the course of training. Predictions from EWC quickly degrade over time, showing that EWC still catastrophically forgets. Predictions from *side-tuning* do not degrade, and the initial quality is better in later tasks (e.g. compare the table in surface normals). We provide additional comparisons (including for PSP) in the supplementary.

4 Experiments

In the first section we show that *side-tuning* compares favorably to existing incremental learning approaches on both iCIFAR and the more challenging Taskonomy dataset. We then extend to multiple domains (computer vision, RL, imitation learning, NLP) in the more common (transfer learning) scenario for $N = 2$ tasks. Finally, we share insights on *side-tuning* in a series of analysis experiments.

4.1 Baselines

We provide comparisons of *side-tuning* against the following methods:

Scratch: The network is given a random initialization and then trained normally.

Feature extraction (features): The base network is used as-is and is not updated during training. Instead, a readout function is trained.

Fine-tuning: An umbrella term that encompasses a variety of techniques, we consider a more narrow definition where pretrained weights are used as initialization and then training proceeds as in *scratch*.

Elastic Weight Consolidation (EWC). A constraint-based incremental learning approach from Kirkpatrick et al. (2016). We use the formulation from which scales better-, giving an advantage to EWC since otherwise we could use a larger *side-tune* network and maintain parameter parity.

Parameter Superposition (PSP): A parameter-masking approach from Cheung et al. (2019).

Independent: Each task uses a network trained independently for the target task. This method uses far more learnable parameters than all the alternatives (e.g. saving a separate ResNet-50 for each task) and achieves very strong performance. Due to the scaling, it is generally not considered an incremental learning method.

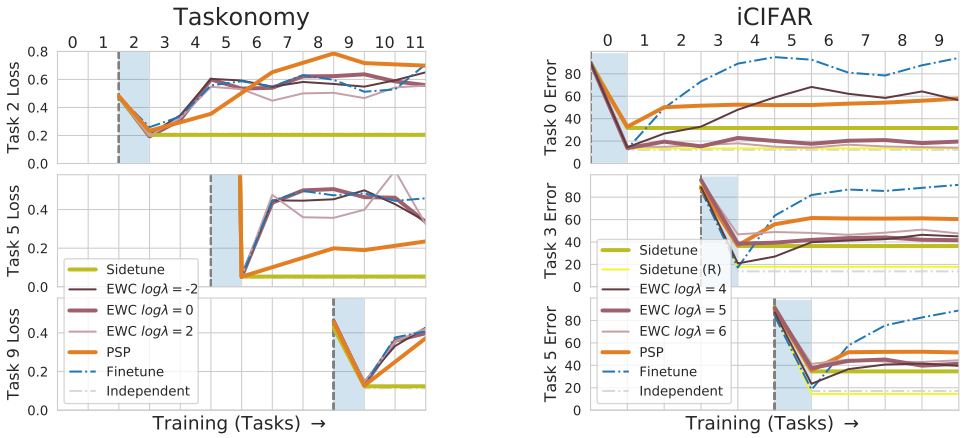


Figure 6: **Incremental Learning on Taskonomy and iCIFAR**. The above curves show loss and error on incremental learning experiments for three tasks on Taskonomy (left) and iCIFAR dataset (right). The fact that *side-tuning* losses are flat after training (as we go right) shows that it does not forget previously learned tasks. Similarly, the performance remains consistent even on later tasks (as we go down), showing that *side-tuning* does not become rigid. Alternative methods clearly forget (e.g. PSP) and/or become rigid (e.g. EWC).

4.2 Incremental Learning: No Catastrophic Forgetting in Additive Learning

On both the Taskonomy dataset (Zamir et al., 2018) and incremental CIFAR (iCIFAR, Rebuffi et al., 2016a), *side-tuning* outperforms existing incremental learning approaches while using fewer parameters[1]. Moreover, the performance gap is larger on more challenging datasets.

Taskonomy includes labels for multiple computer vision tasks including 2D (e.g. edge detection), 3D (e.g. surface normal estimation), and semantic (e.g. object classification) tasks. We first selected the twelve tasks that make predictions from a single RGB image, and then created an incremental learning setup by selecting a random order in which to learn these tasks (starting with *curvature*). As images are 256x256 we use a ResNet-50 for the base network and a 5-layer convolutional network for the *side-tuning* side network. The number of learnable network parameters used across all tasks is 24.6M for EWC and PSP, and 11.0M for *side-tuning*[2].

iCIFAR. First, we pretrain the base network (ResNet-44) on CIFAR-10. Then the 10 subsequent tasks are formed by partitioning CIFAR-100 classes into 10 disjoint sets of 10-classes each. We train on each subtask for 20k steps before moving to the next one. The state-of-the-art substitutive baselines (EWC and PSP) update the base network for each task (683K parameters), while *side-tuning* updates a four layer convolutional network per task (259K parameters after 10 tasks).

4.2.1 Catastrophic Forgetting

As expected, there is no catastrophic forgetting in *side-tuning*. Figure 6 shows that the error for *side-tuning* does not increase after training (blue shaded region), while it increases sharply for the other methods on both Taskonomy and iCIFAR.

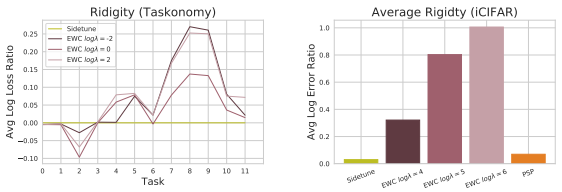
The difference is meaningful, and Figure 5 shows sample predictions from *side-tuning* vs. EWC for a few tasks during and after training. As is evident from the bottom rows, EWC exhibits catastrophic forgetting on all tasks (worse image quality as we move right). In contrast, *side-tuning* (top) shows no forgetting and the final predictions are significantly closer to the ground-truth (boxed red).

4.2.2 Rigidity

Side-tuning learns later tasks as easily as the first, while constraint-based methods such as EWC stagnate. The predictions for later tasks such as *surface normals* (in Figure 5) are significantly better using *side-tuning*—even immediately after training and before any forgetting can occur.

[1]Full experimental details, including learning rate and architecture, are provided in the supplementary.

[2]All parameters not counting readout parameters, which are common between all methods.



Method	Avg. Rank (↓) Taskonomy	iCIFAR
Fine-tune	3.50	3.70
EWC ($\lambda = 10^0, 10^5$)	3.00	1.90
PSP	2.33	3.10
Side-tune	**1.12**	**1.30**

Figure 7: **Rigidity and average rank on Taskonomy and iCiFAR**. From left: *Side-tuning* always learns new tasks easily; EWC becomes increasingly unable to learn new tasks as training progresses. Center: The same trend holds on iCIFAR, and the average rigidity is zero for *side-tuning* (and almost zero for PSP). Right: *Side-tuning* outperforms alternatives on both datasets, achieving a significantly better average rank on all tasks.

Figure 7 quantifies this slowdown. We measure rigidity as the log-ratio of the *observed loss* on the ith task over the loss when that task is *instead trained first* in the sequence. As expected, *side-tuning* experiences zero slowdown on both datasets. For EWC, the increasing constraints make learning new tasks increasingly difficult—and the log-ratio increases with the number of tasks (Taskonomy, left). EWC is observed to be too rigid (log-ratio > 0) even in iCIFAR, where the later tasks are similar to earlier ones.

4.2.3 Final Performance

Overall, *side-tuning* significantly outperforms the other methods while using fewer than half the number of trainable parameters of the other methods. When the other methods use smaller networks, their performance decreases further. On both iCIFAR and Taskonomy, *side-tuning* achieves the best average rank (1.12 of 4 on Taskonomy, while the next best is 2.33 (PSP)).

This is a direct result of the fact (shown above) that *side-tuning* does not suffer from catastrophic forgetting or rigidity. It is not due to the fact that the sidetuning structure is specially designed for these types of image tasks; it is not (we show in Sec. 4.3 that it performs well on other domains). In fact, the much larger networks used in EWC and PSP *should* achieve better performance on any single task. For example, EWC produces sharper images early on in training, before it has had a chance to accumulate too many constraints (e.g. *reshading* in Fig. 5). But this factor was outweighed by *side-tuning's* immunity from the effects of catastrophic forgetting and creeping rigidity.

4.3 Universality of the Results: Side-Tuning in Other Domains

In order to assess if the *side-tuning* results are somehow domain/task-specific, we provide results showing that it is well-behaved in other settings. As the concern with *additive learning* is mainly that it is too inflexible to learn new tasks, we compare with fine-tuning (which outperforms other lifelong learning tasks when forgetting is not an issue). For extremely limited amounts of data, feature extraction can outperform fine-tuning. We show that *side-tuning* generally performs as well as features or fine-tuning–whichever is better.

	QA on SQuAD		**Navigation (IL)**		**Navigation (RL)**		**Transfer Learning in Taskonomy**	
Method	Match (↑) Exact	F1	Nav. Rew. (↑) Curvature	Denoise	Nav. Rew. (↑) Curvature	Denoise	From Curvature (100/4M ims.) Normals (MSE ↓)	Obj. Cls. (Acc. ↑)
Fine-tune	**79.0**	**82.2**	10.5	9.2	10.7	**10.0**	**0.200 / 0.094**	**24.6 / 62.8**
Features	49.4	49.5	**11.2**	8.2	**11.9**	8.3	**0.204** / 0.117	24.4 / 45.4
Scratch	0.98	4.65	9.4	**9.4**	7.5	7.5	0.323 / **0.095**	19.1 / 62.3
Side-tune	**79.6**	**82.7**	**11.1**	**9.5**	**11.8**	**10.4**	**0.199 / 0.095**	**24.8 / 63.3**
	(a)		**(a)**		**(c)**		**(d)**	

Figure 8: **Side-tuning comparisons in other domains.** Sidetuning matched the adaptability of fine-tuning on large datasets, while performing as well or better than the best competing method in each domain: **(a)** In SQuAD v2 question-answering, using BERT instead of a convolutional architecture. **(b)** In Habitat, learning to navigate by imitating expert navigation policies, using inputs based on either *Curvature* or *Denoising*. Finetuning does not perform as well in this domain. **(c)** Using RL (PPO) and direct interaction instead of supervised learning. **(d)** In Taskonomy, performing either *Normal Estimation* or *Object Classification* using a base trained for *Curvatures* and either 100 or 4M images for transfer. Results using *Obj. Cls.* base are similar and provided in the appendix.

Question-Answering in SQuAD v2: We also evaluated *side-tuning* on a question-answering task (SQuAD v2 (Rajpurkar et al., 2018)) using a non-convolutional architecture. We use a pretrained BERT (Devlin et al., 2018) model for our base, and a second for the side network[3]. Unlike in the previous experiments, BERT uses attention and no convolutions. Still, *side-tuning* adapts to the new task just as well as fine-tuning, outperforming features and scratch (Figure 8)a.

Imitation Learning for Navigation in Habitat: We trained an agent to navigate to a target coordinate in the Habitat environment, the standard task in Habitat Savva et al. (2019). The results are provided in Fig. 8b. The agent is provided with both RGB input image and also an occupancy map of previous locations. The map does not contain any information about the environment—just previous locations. In this section we use Behavior Cloning to train an agent to imitate experts following the shortest path on 49k trajectories in 72 buildings. The agents are evaluated in 14 held-out validation buildings. Depending on the what the base network was trained on, the source task might be useful (*Curvature*) or harmful (*Denoising*) for imitating the expert and this determines whether features or learning from scratch performs best. Figure 8b shows that regardless of the which approach worked best, *side-tuning* consistently matched or beat it.

Reinforcement Learning for Navigation in Habitat Using a different learning algorithm (PPO) and using direct interaction instead of expert trajectories, we observe trends identical to imitatino learning. We trained agents directly in Habitat (74 buildings)[3]. Figure 8c shows performance in 16 held-out buildings after 10M frames of training. *Side-tuning* performs comparably to the $\max$ of competing approaches.

Transfer learning in Taskonomy: We trained networks to perform one of three target tasks (object classification, surface normal estimation, and curvature estimation) on the Taskonomy dataset (Zamir et al., 2018) and varied the size of the training set $N \in \{100, 4 \times 10^6\}$. In each scenario, the base network was trained (from scratch) to predict one of the non-target tasks. The side network was a copy of the original base network. We experimented with a version of fine-tuning that updated both the base and side networks; the results were similar to standard fine-tuning [3]. In all scenarios, *side-tuning* successfully matched the adaptiveness of fine-tuning, and significantly outperformed learning from scratch, as shown in Figure 8d. The additional structure of the frozen base did not constrain performance with large amounts of data (4M images), and *side-tuning* performed as well as (and sometimes slightly better than) fine-tuning.

4.4 Learning Mechanics in Side-Tuning

Initialization. A good side network initialization can yield a minor boost in performance. We found that initializing from the base network slightly outperforms a low-energy initialization[4], which slightly outperforms Xavier initialization. However, we found that these differences were not statistically significant across tasks (H_0 : *pretrained* = *xavier*; $p = 0.07$, Wilcoxon signed-rank test). We suspect that initialization might be more important on harder problems. We test this by repeating the analysis without the simple texture-based tasks (2D keypoint + edge detection and autoencoding) and find the difference in initialization is now significant ($p = 0.01$).

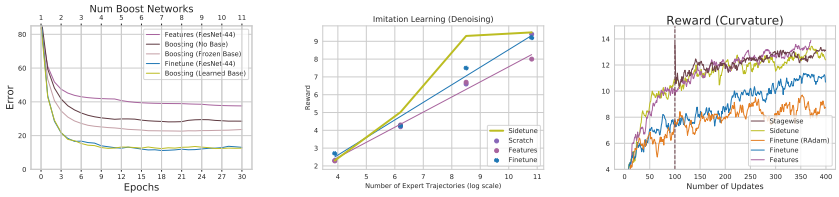
Task relevance predicts alpha α. In our experiments, we treat α as a learnable parameter and find that the relative values of α are predictive of emprical performance. In imitation learning (Fig. 4.3), *curvature* ($\alpha = 0.557$) outperformed *denoising* ($\alpha = 0.252$). In Taskonomy, the α values from training on just 100 images predicted the actual transfer performance to normals in Zamir et al. (2018), (e.g. *curvature* ($\alpha = 0.56$) outperformed *object classification* ($\alpha = 0.50$)). For small datasets, usually $\alpha \approx 0.5$ and the relative order, rather than the actual value is important.

Not Boosting. Since the side network learns a residual on top of the base network, we ask: what benefits we could glean by extending *side-tuning* to do boosting? Although network boosting this does improve performance on iCIFAR (Figure 9a), if catastrophic forgetting is not a concern then the parameters would've been better used in a deeper network rather than many shallow networks.

Benefits for intermediate amounts of data We showed in the previous section that *side-tuning* performs like the best of $\{\text{features}, \text{fine-tuning}, \text{scratch}\}$ in domains with abundant or scant data. In

[3]We defer remaining experimental details (learning rate, full architecture, etc.) to the the appendix. See provided code for full details.

[4]Where the side network is trained so that it does not impact the output. Full details in the supplementary.



(a) **(b)** **(c)**

Figure 9: **Analysis of learning mechanics.** **(a)** Boosting: deeper network $>$ many shallow learners. **(b)** *Side-tuning* outperformed alternatives on intermediate amounts of data. **(c)** Features/*Side-tuning* do more than reduce gradient variance.

order to test whether *side-tuning* could profitably synthesize the features with *intermediate* amounts of data, we evaluated each approach's ability to learn to navigate using 49, 490, 4900, or 49k expert trajectories and pretrained *denoising* features. *Side-tuning* was always the best-performing approach and, on intermediate amounts of data (e.g. 4.9k trajectories), outperformed the other techniques (*side-tune* 9.3 vs. fine-tune: 7.5, features: 6.7, scratch: 6.6), Figure 9b).

Network size. Does network size matter? We find (i) If the target problem benefits from a large network (e.g. classification tasks), then performance is sensitive to side network size but not size of the base. (ii) The base network can usually be distilled to a smaller network and *sidetuning* will still offer advantages over alternatives. In the supplementary material we provide supporting experiments from Taskonomy using both high- and low-data settings (curvature $\rightarrow$ $\{$obj. class, normals$\}$, obj. class $\rightarrow$ normals), and in Habitat (RL using $\{$curvature, denoise$\}$ $\rightarrow$ navigation).

More than just stable updates. In RL, fine-tuning often fails to improve performance. One common rationalization is that the early updates in RL are 'high variance'. The usual solution is to first train using fixed features and then unfreeze the weights at some point in training (via a hyperparameter to be set). We found that this stage-wise approach performs as well (but no better than) keeping the features fixed–and *side-tuning* performed as well as both while being simpler than stage-wise (Fig. 9c). We tested the 'high-variance update' theory by fine-tuning with both gradient clipping and an optimizer designed to prevent such high-variance updates by adaptively warming up the learning rate (RAdam, Liu et al., 2019). This provided no benefits over vanilla fine-tuning, suggesting that the benefits of side-tuning are not solely due to gradient stabilization early in training.

5 Conclusion

We have introduced the *side-tuning* framework: a simple yet effective approach for *additive learning*. Since, by design, it does not suffer from catastrophic forgetting or rigidity, it is naturally suited to incremental learning. The theoretical advantages are reflected in empirical results, and *side-tuning* outperforms existing approaches in challenging contexts and with state-of-the-art neural networks. We demonstrated that the approach is effective in multiple domains and with various network types.

6 Future Work

The naïve approach to incremental learning used in this paper made a number of design decisions. These decisions could be analyzed and subsequently relaxed. In particular:

Flexible parameterizations for side networks: Our incremental learning experiments used the same side network architecture for all subtasks. A method for automatically adapting the networks to the subtask at hand could make more efficient use of the computation and supervision.

Better forward transfer: Our experiments used only a single base and single side network. Leveraging the already previously trained side networks could yield better performance on later tasks.

Learning when to deploy side networks: Like most incremental learning setups, we assume that the tasks are presented in a sequence and that task identities are known. Using several active side networks in tandem would provide a natural way to detecting distribution shift.

Using side-tuning to measure task relevance: We noted that α tracked task relevance, but a more rigorous treatment of the interaction between the base network, side network, α and final performance could yield insight into how tasks relate to one another.

REFERENCES

Karen E Adolph, Sarah E Berger, and Andrew J Leo. Developmental continuity? crawling, cruising, and walking. *Developmental science*, 14(2):306–318, 2011.

Rodney Brooks. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. *CoRR*, abs/1812.00420, 2018. URL http://arxiv.org/abs/1812.00420.

Brian Cheung, Alex Terekhov, Yubei Chen, Pulkit Agrawal, and Bruno A. Olshausen. Superposition of many models into one. *CoRR*, abs/1902.05522, 2019. URL http://arxiv.org/abs/1902.05522.

Whitney G Cole, Gladys LY Chan, Beatrix Vereijken, and Karen E Adolph. Perceiving affordances for different motor skills. *Experimental brain research*, 225(3):309–319, 2013.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL http://arxiv.org/abs/1703.03400.

Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. *CoRR*, abs/1812.03201, 2018. URL http://arxiv.org/abs/1812.03201.

Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016. URL http://arxiv.org/abs/1603.08155.

Nancy Kanwisher. Functional specificity in the human brain: a window into the functional architecture of the mind. *Proceedings of the National Academy of Sciences*, 107(25):11163–11170, 2010.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. URL http://arxiv.org/abs/1612.00796.

Kari S Kretch and Karen E Adolph. Cliff or step? posture-specific learning at the edge of a drop-off. *Child development*, 84(1):226–240, 2013.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016. URL http://arxiv.org/abs/1606.09282.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

Arun Mallya and Svetlana Lazebnik. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *CoRR*, abs/1801.06519, 2018. URL http://arxiv.org/abs/1801.06519.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. *arXiv e-prints*, art. arXiv:1909.09157, Sep 2019.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. URL http://arxiv.org/abs/1806.03822.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014. URL http://arxiv.org/abs/1403.6382.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016a. URL http://arxiv.org/abs/1611.07725.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016b. URL http://arxiv.org/abs/1611.07725.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. URL http://arxiv.org/abs/1606.04671.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. *arXiv preprint arXiv:1904.01201*, 2019.

Thomas Schenk and Robert D McIntosh. Do we have independent visual streams for perception and action? *Cognitive Neuroscience*, 1(1):52–62, 2010.

Jonathan Schwarz, Jelena Luketina, Wojciech M Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *arXiv preprint arXiv:1805.06370*, 2018.

Tom Silver, Kelsey R. Allen, Josh Tenenbaum, and Leslie Pack Kaelbling. Residual policy learning. *CoRR*, abs/1812.06298, 2018. URL http://arxiv.org/abs/1812.06298.

Amir R. Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.

A APPENDIX

A.1 TABLE OF CONTENTS

We provide the following material in the appendices:

A.2... Qualitative results for incremental learning
A.3... Additional experiments
A.4... Experimental details

A.2 Qualitative Results in Incremental Learning

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

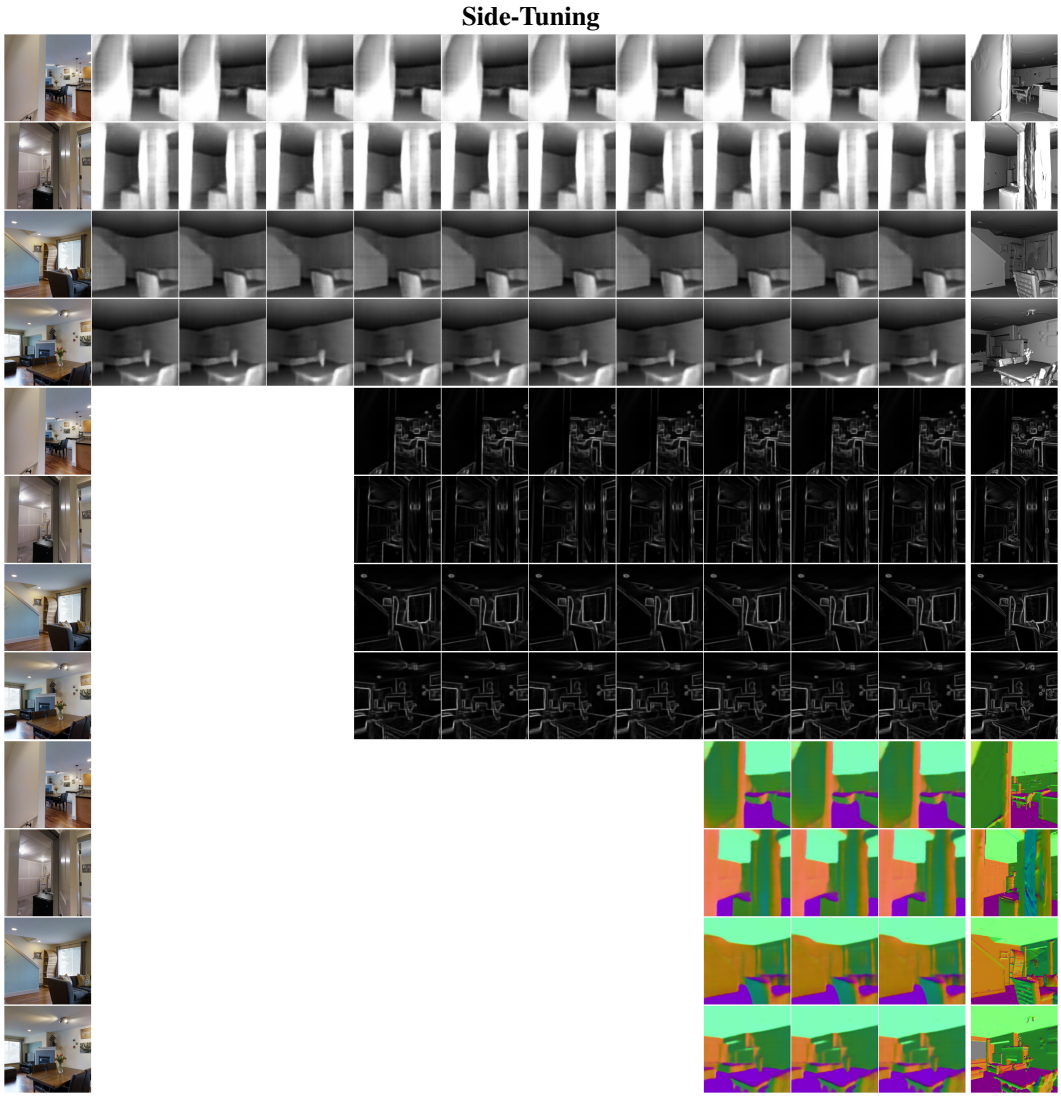


Figure 10: **More qualitative results for *side-tuning*.** These images were randomly selected from the validation set. Left-hand column is input, rightmost-column is ground truth. Images from left to right show predictions as training progresses. Each block of 4 rows shows predictions on a different task (*Reshading*, *2D Edges*. *Surface Normals*.)

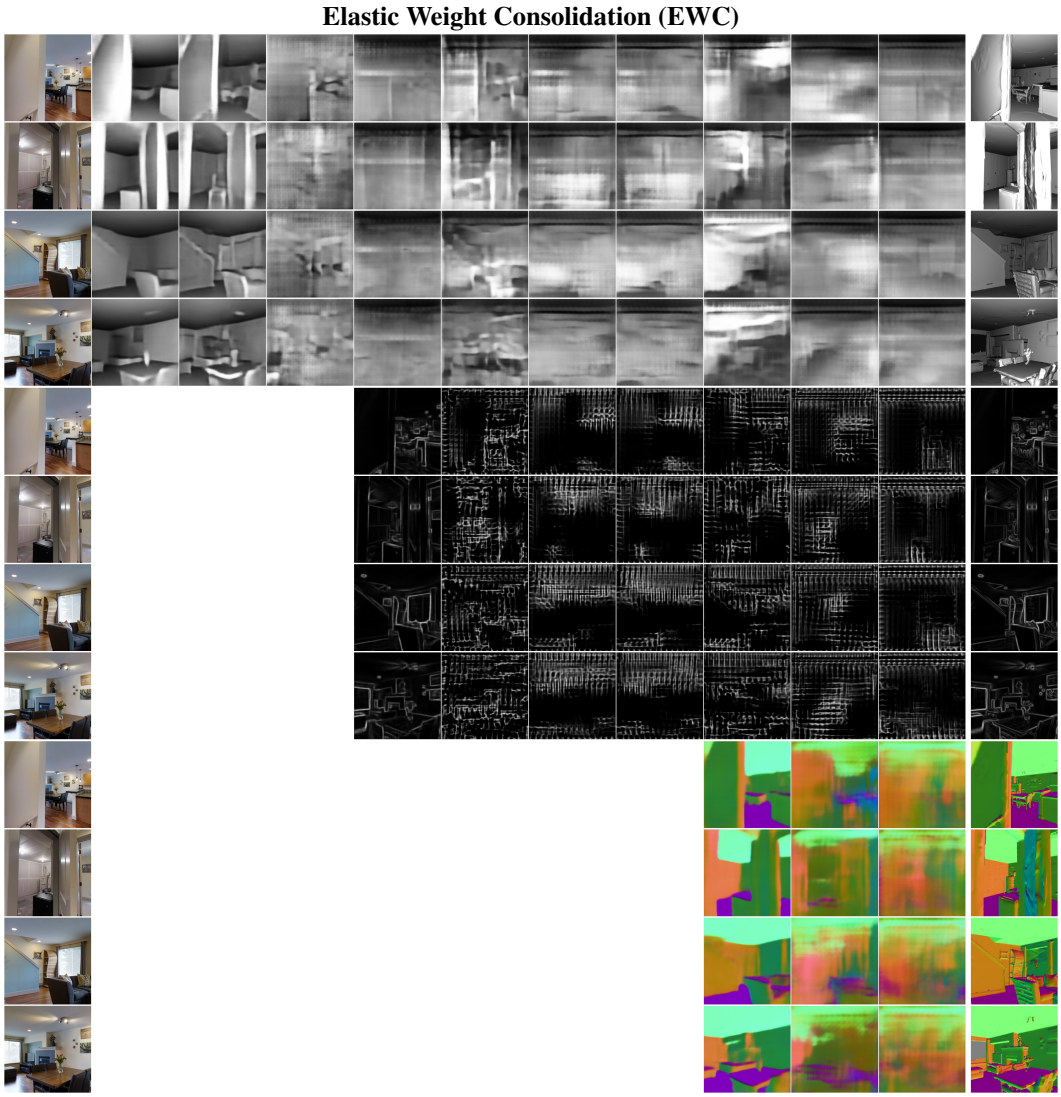


Figure 11: **More qualitative results for EWC.** These images were randomly selected from the validation set. Left-hand column is input, rightmost-column is ground truth. Images from left to right show predictions as training progresses. Each block of 4 rows shows predictions on a different task (*Reshading*, *2D Edges*. *Surface Normals*.)

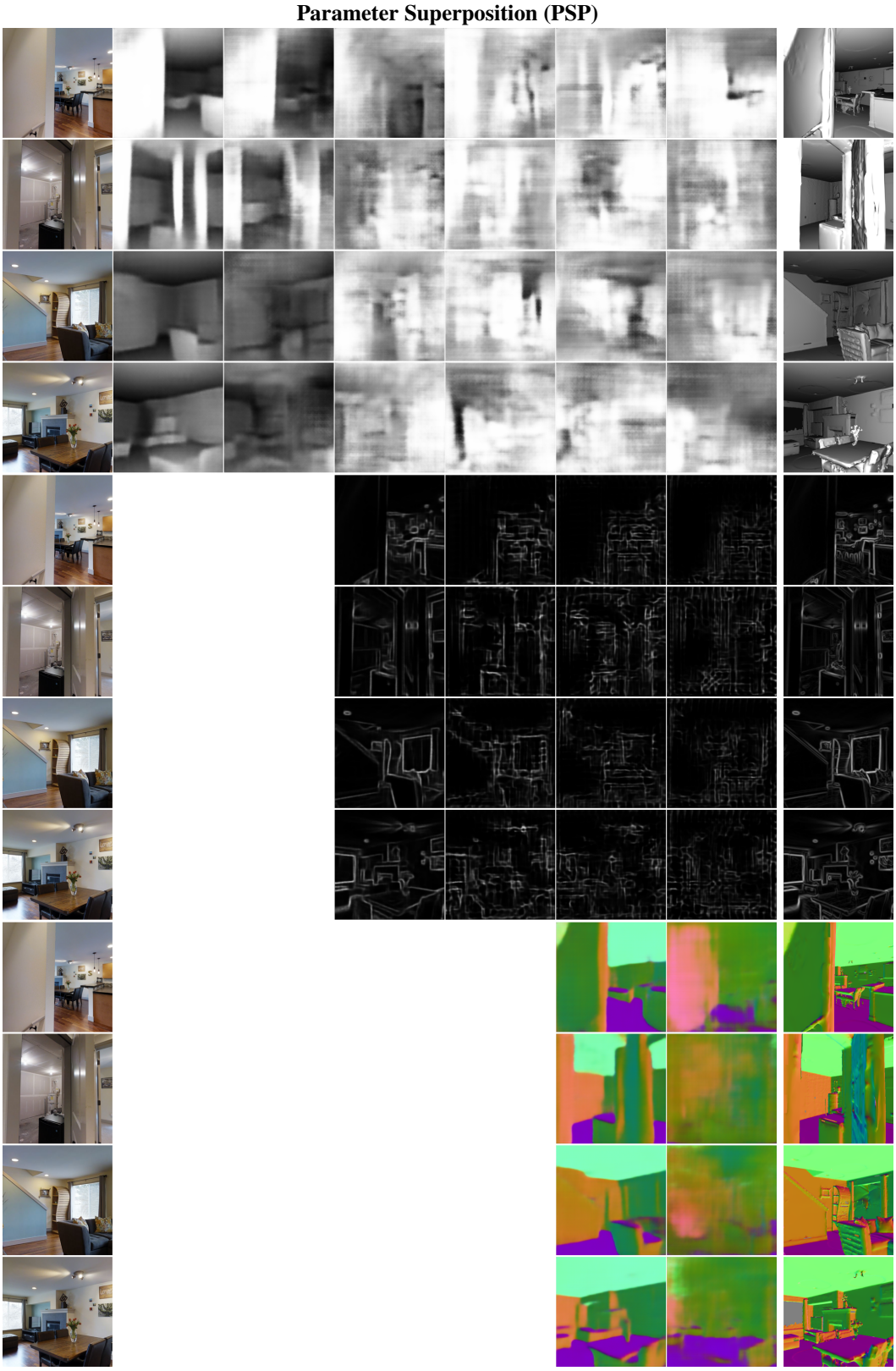


Figure 12: **More qualitative results for PSP.** These images were randomly selected from the validation set. Left-hand column is input, rightmost-column is ground truth. Images from left to right show predictions as training progresses. Each block of 4 rows shows predictions on a different task (*Reshading*, *2D Edges*. *Surface Normals*.)

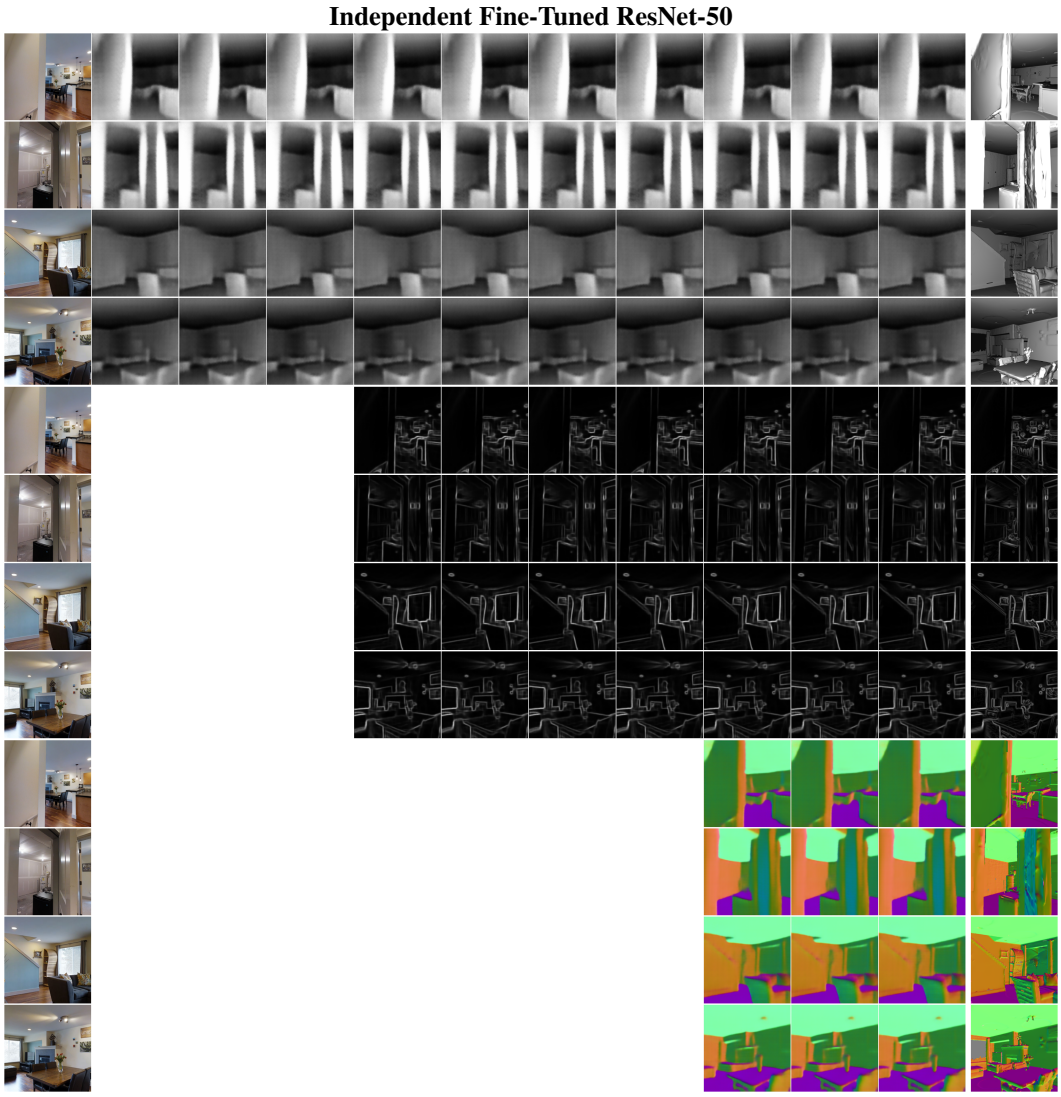


Figure 13: **More qualitative results for *independent*** These images were randomly selected from the validation set. Left-hand column is input, rightmost-column is ground truth. Images from left to right show predictions as training progresses. Each block of 4 rows shows predictions on a different task (*Reshading*, *2D Edges*. *Surface Normals*.)

A.3 Additional Experiments

A.3.1 Network Size

Method	Transfer Learning in Taskonomy: From Curvature (100/4M ims.) Normals (MSE ↓)	Obj. Cls. (Acc. ↑)
Standard	0.20/0.010	24.8/63.3
Small Base	0.20/0.09	25.3/63.2
Large Side	0.21/0.11	25.3/55.6

(a)

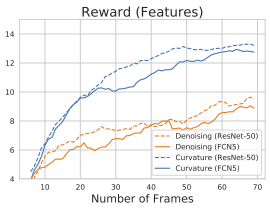
Transfer Learning in Taskonomy: From Obj. Class. (100) Normals (MSE ↓)
0.25
0.22
0.23

(b)

Navigation (IL): Nav. Rew. (49/4900 epi) (↑) Curvature	Denoise
4.9/9.5	2.3/9.3
4.3/9.5	2.2/8.4
X	X

(c)

Figure 14: **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.



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.

A.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 15).

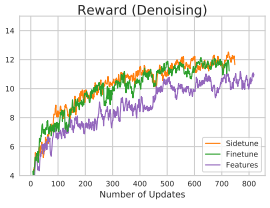


Figure 15: **Reinforcement Learning)** *Side-tuning* matches the performance of the best method when using denoising features as well.

A.3.3 IMITATION LEARNING

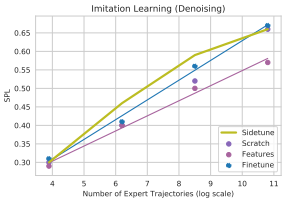


Figure 16: **Imitation Learning (Denoising) SPL**

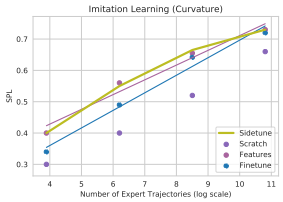


Figure 17: **Imitation Learning (Curvature) SPL**

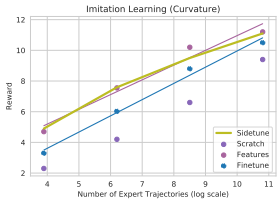


Figure 18: **Imitation Learning (Curvature) Reward**

Figure 19: 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.

A.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) (↑)		Loss (5 ims) (↓)
	Curvature	Denoise	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

A.4 EXPERIMENTAL SETUP

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

A.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.

A.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

A.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 (Fig. **??**). 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) algo- rithm 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.

A.4.4 Experimental Setup for Learning Mechanics (Sec. 4.4)

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