A EXPERIMENTAL SETUP

We provide further training details for our approaches in the following sections. First we discuss the adapter parameterizations in more detail specifically detailing the adapter locations and their associated trainable parameters.

A.1 ADAPTER PARAMETERIZATIONS

As noted previously in Sub-Section 3.2 we use *bottom*, *middle* and *top* adapters. We discuss the overall number of parameters for each of these adapter types. Table 4 provides an overall count for the number of adapter parameters.

Bottom: For NFNet and ResNet architectures we use the initial convolution, while for ViT we use the initial patch embedding as the only bottom set of layers. We add 1 adapter for this bottom layer. Since the image input only contains 3 channels and the bottom layer consists of only 16 and 256 channels for NFNet and ResNets respectively, bottom adapters require very few (< 1000) parameters. While for ViT since we have a large feature output, it results in a much larger number of parameters $\approx 0.7M$.

Middle: For middle adapters we use 4 and 6 adapters for the convnet and transformer based architectures respectively. For the convnet based architectures we add each adapter to the first convolution in each block group except the last group, where we add it at the end. While for the transformer based architectures we apply them at layers $\{0, 1, 5, 6, 10, 11\}$ which yields around $\sim 400K$ adapter parameters. While better choices and adapter placements may exist we use these uniformly across all tasks. Finally, the output of middle layer consists of a set of spatial features. For NFNet and ResNet these are output of layer 4 and final conv with a spatial size of 7×7 and a feature size of 3096 and 2048 respectively. For ViTs we get 196 patches each with 768 features.

Top: As noted previously in Sub-section 3.2, the high dimensional spatial features from the middle layer can be reduced either via mean pooling or by projecting them onto a lower dimensional space through a single layer. Since prior works use mean pooling we use it to compare our work with them. However, since manipulation tasks are spatial in nature we also investigate down-projecting the high dimensional spatial features into smaller dimensions and concatenating them. This formulation avoids any loss of spatial information. To achieve this for NFNet and ResNets we use 1 convolution layer with 1×1 kernel and 41 output channels. While for ViT we use a shared MLP that projects each patch embedding into a 20 dimensional feature. Finally, to further process these features we *optionally* add 2 MLPs each with 256 parameters, we refer to these as top layer adapters.

Policy Head: We universally use a linear policy head that converts the output of the top layer into the robot action to be executed.

A.2 TRAINING DETAILS

As noted in the main paper we use behavior cloning with mean squared loss as the optimization objective. We use a linear policy head to predict continuous actions. While specific action parameterizations, such as the use of binary gripper can help increase performance, for a fair comparison of representations we avoid using any such techniques. Table 5 lists out the detailed hyperparameters used in our experiments. We uniformly use the same set of hyperparameters across most task settings except the learning rate, wherein we found using a slightly higher learning rate of $1e-3$ works better for Franka-Kitchen tasks.

Network Details: As discussed before our implementation uses three different network architectures – NFNets, ResNets and ViTs. Figure 3 presents the overall architecture. In settings where we use proprioceptive information, we use a single linear layer with 256 dimensions to map the low dimensional proprioceptive information to a higher dimensional space. This high dimensional proprioceptive information is then concatenated with the visual features before being forwarded to the 2 layer MLP (each with 256 units). Further, since we evaluate our approach across very different architectures we use the same policy form across all of them. Thus, we avoid using any normalization techniques such as BatchNorm or LayerNorm in our policy implementation.

Training Parameters	MetaWorld	Franka-Kitchen	RGB Stacking
Loss	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam
Learning Rate	1e-4	1e-3	1e-4
Weight Decay	1e-6	1e-6	1e-6
Gradient Norm Clip	1.0	1.0	1.0
Training Steps	40K	40K	200K
Learning Rate Schedule	cosine	cosine	cosine
Learning Rate Schedule Warmup Steps	5K	5K	10K
Adapter Features Size	32	32	32

Table 5: Training Details for each of the three different task suites used in our work. For each task within the task suite we use the same set of hyperparameters.

	Assembly	Bin-Picking	Button Press	Drawer Open	Hammer	Average
NFNet	0.92	0.7	0.94	0.96	0.94	0.89
ResNet	0.90	0.66	0.96	0.94	0.92	0.88
ViT	0.92	0.8	0.91	0.98	0.92	0.91

Table 6: Task specific results for using bottom, middle and top adapters with proprioceptive information (proprio) for each task in MetaWorld.

B Additional Results

We provide further results for the use of adapters in the three different environment suites considered in the main paper. We then discuss the ablation results on adapter locations for all suites and network architectures. For these results, in addition to average metrics across all enviroments, we also provide task specific metrics.

B.1 Adapter Results with Proprioceptive Information

In this section we present detailed task-specific success rate using our proposed adapters for each task in the three manipulation suites. For these results we use all top, bottom and middle adapters in our implementation. Further, in addition to visual features we also utilize proprioceptive information for these results. Table 6, 7 and Table 8 report task-specific results for MetaWorld, Franka-Kitchen and RGB Stacking suites using all three different architectures with imagenet pretrained weights. Comparing Table 6 with previous results in Table 2 we see that adding proprioceptive information results in $\approx 10\%$ increase in the average success rate. This increase holds consistently across all architectures. More interestingly we also find that for most tasks (except Bin-Picking) the agent can reach greater than 90% performance, while for some tasks such as *Button-Press* and *Drawer-Open* it can even reach close to 100% performance.

Table 7 shows the results for each task in the Franka-Kitchen suite. Compared to previous results in Table 2 we see a much larger increase in the performance ($\approx 60\%$ relative performance increase on average) of each architecture in the Franka-Kitchen suite. One reason for such a large increase is the very limited state space distribution for these tasks. Since all objects in the environment are fixed and only the initial robot configuration changes, it is much easier for the robot to memorize the proprioceptive information and map it to observed expert actions for improved task performance. Additionally, while both MetaWorld and Franka-Kitchen use 25 demonstrations, each demonstration in MetaWorld has 500 steps while in Franka-Kitchen each demonstration is only 50 steps. This results in $10\times$ difference in the amount of training data. However, since prior works use these settings for a fair comparison we follow similar evaluation protocols.

B.2 Effects of Adapter Locations

In this section we investigate the effect of inserting adapters in each of the different network layers as discussed in Subsection 3.2 and initially explored in Subsection 5.2. Due to space constraints in Subsection 5.2 we only provide results for the RGB Stacking task. In this subsection we show

	Knob1-On	LDoor-Open	Light-On	Micro-Open	SDoor-Open	Average
NFNet	0.46	0.44	0.72	0.32	0.94	0.58
ResNet	0.48	0.46	0.60	0.30	0.88	0.54
ViT	0.6	0.48	0.59	0.36	0.83	0.57

Table 7: Task specific results for using bottom, middle and top adapters with proprioceptive information (proprio) for each task in Franka-Kitchen suite.

	Triplet 1	Triplet 2	Triplet 3	Triplet 4	Triplet 5	Average
NFNet	0.40	0.18	0.11	0.67	0.90	0.45
ResNet	0.48	0.34	0.11	0.62	0.86	0.48
ViT	0.25	0.40	0.13	0.80	0.85	0.49

Table 8: Task specific results for using bottom, middle and top adapters for each task in RGB-Stacking suite.

results across all manipulation suites and network architectures. For ease of comparison we also plot the RGB Stacking results from before.

Figure 7 shows results for inserting adapters in each network layer across all 3 task suites and architectures. As noted before, we split the results in each plot into two parts. 1) without using top layer adapters (i.e. directly using a linear policy head), and 2) using a top layer adapter (i.e. using 2 additional MLPs before the linear policy head). Moreover, in addition to the different adapter locations we also show results for fixed pretrained representations (Pretrain Feat.) and full fine-tuning (Full FT.) both with and without top adapters. As noted in the main paper, prior works always use such top adapters in their implementations.

Top Adapters: In our discussion in Subsection 5.2 we showed that for the RGB Stacking task top adapters are quite important to achieve close to optimal task performance. We note that the greyed out plots in Figure 7 indicate methods that do not use top adapters. Our results in Figure 7 show that this holds true for both MetaWorld and Franka-Kitchen suites as well. For both of these suites we find that using top adapters improves the downstream manipulation performance. However, as seen in the metaworld results (top row of Figure 7), full fine-tuning approaches (last bar in each plot) can reach good performance even without top adapters. However, this does not hold for the Franka-Kitchen tasks (middle row in Figure 7). We hypothesize this is because of the metaworld setup, wherein there is usually a single object centered on an otherwise empty table, which presents an easier visual setting and simply fine-tuning the high capacity pretrained visual model can extract the appropriate task representation. However, we do note that our use of adapters is able to closely match the full fine-tuning performance across all architectures.

Bottom Adapters: Similar to RGB-stacking results before we note that the bottom adapters (plotted in Green) with very few parameters (around a few thousand) can lead to substantially better results than simply using fixed pretrained models. This holds when bottom adapters are combined with top adapters and even in the absence of top adapters. Although, as noted before the overall results are much poorer without top adapters. From Figure 7 we see that bottom adapters help for both NFNet (green bar in column 1, row 1 and column 1, row2) and ResNet (green bar in column 1, row 1 and column 1, row2). Thus, broadly similar results hold across environment suites.

Middle Adapters: From Figure 7 also shows that while bottom and top adapters *together* (green bar on the right plots) can achieve good performance there still exists a significant gap compared to the full fine-tuning approach. However, inserting middle adapters, either alone (shown by orange) or together with bottom adapters (shown in purple) leads to a much more improved performance. Overall, using adapters in all the layers is closely able to match the full fine-tuning performance. This substantial effect of middle adapters is not unexpected since the middle part of the network contains a large part of the pretrained network and thus has significant affect on the output representation.

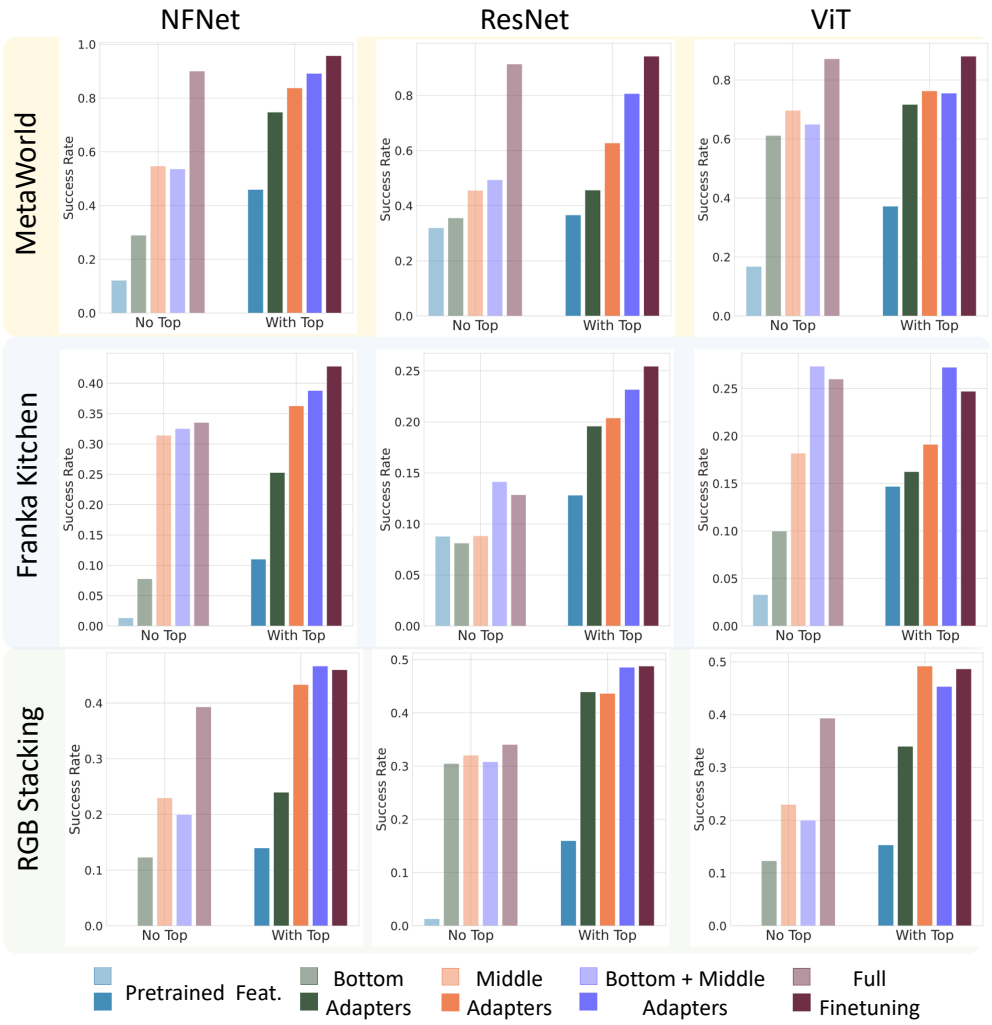

Figure 7: Results on the RGB-Stacking environment for 3 different type of model architectures.