In this supplementary material, we present additional details that we have referred to in the main paper.

A. Datasets

ImageNet to Sketch Dataset. Originally introduced in [23] and referred to as Imagenet to Sketch in [2], our first benchmark consists of 5 datasets namely: CUB [47], Cars [15], WikiArt [41], Flowers [30] and Sketch [7]. Following [24], Cars and CUB are the cropped datasets, while the rest of the datasets are as is. For the Flowers dataset, we combine both the train and validation split as the training data. We resize all images to 256 and use a random resized crop of size 224 followed by a random horizontal flip. This augmentation is applied at training time to all the datasets except Cars and CUB. For these two datasets, we use only the horizontal flip as the data augmentation during training as in [9, 23, 24].

The Sketch dataset is licensed under a Creative Commons Attribution 4.0 International License. The CUBs, WikiArt, and Cars datasets use are restricted to non-commercial research and educational purposes.

Visual Decathlon Challenge. The Visual Decathlon Challenge [33] aims at evaluating visual recognition algorithms on images from multiple visual domains. The challenge consists of 10 datasets: ImageNet [38], Aircraft [22], CIFAR-100 [16], Describable textures [4], Daimler pedestrian classification [28], German traffic signs [43], UCF-101 Dynamic Images [42], SVHN [29], Omniglot [17], and Oxford Flowers [30]. The images of the Visual Decathlon datasets are resized isotropically to have a shorter side of 72 pixels, to alleviate the computational burden for evaluation. Each dataset has a different augmentation for training. We follow the training protocol as used in [33, 34]. Visual Decathlon does not explicitly provide a usage license.

DomainNet. DomainNet [31] is a benchmark for multi-source domain adaptation in object recognition. It contains 0.6M images across six domains (Clipart, Infograph, Painting, Quickdraw, Real, Sketch). All domains include 345 categories (classes) of objects. Each domain is considered as a task and we use the official train/test splits in our experiments. We also use the same augmentations as used in [45]. DomainNet is distributed under fair usage as provided for in section 107 of the US Copyright Law.

B. Detailed Experimental Settings

B.1. Incremental MTL

As mentioned in the main paper, we describe detailed settings for all the experiments in Sec 4.1. **DenseNet-121**

Training. We sweep through learning rate of $\{0.005, 0.01\}$ and use a similar setup to our ResNet-50 model, *i.e*. SGD optimizer with no weight decay. We train for 30 epochs and use a cosine annealing learning rate scheduler. We used the same augmentation as that of ResNet-50 model as stated earlier. To summarize, the learning rate and the network architecture are the only things that change compared to the ResNet-50 model training.

ViT-S/16 Training. We train the ViT-S/16 model with a learning rate of 0.00005, momentum of 0.9, and $\lambda = 1$. We train for 30 epochs with batch size of 32 and cosine annealing learning rate. Data augmentation consists of random resized crop and horizontal flip.

B.2. Joint MTL

In this section, we describe detailed settings for the DomainNet experiments in Sec 4.2. For each of the methods, we provide details regarding both joint and incremental MTL settings.

Fine-tuning. For the joint MTL setting, fine-tuning is done with the adjustment of batch size and learning rate. The shared backbone is trained with a batch size of 72 with 12 images from each task, learning rate 0.005, momentum 0.9, and no weight decay. We train for 60 epochs where an epoch consists of sampling every image across all 6 datasets. We find that balanced sampling over all tasks for each mini-batch is necessary for stable training as in [45].

For the incremental MTL setting, we train 30 epochs with batch size 32, learning rate 0.005, momentum 0.9, and weight decay 0.0001.

AdaShare. The learning process of AdaShare consists of two phases: the policy learning phase and retraining phase. During the policy learning phase, both weights and policies are optimized alternatively for 20K iterations. After the policy learning phase, different architectures are sampled with different seeds and the best results are reported. For the joint MTL setting, we follow the same setting as the original implementation [45]. Note that the original implementation samples 8 architectures from the learned policy and retrains them and report the best result, here for fair comparison with other methods, only 1 sampling is performed. For the incremental setting, we use the same setting as the joint-MTL, except the dataset contains only 1 domain.

TAPS. For all experiments we use learning rate 0.005, momentum 0.9, no weight decay, and $\lambda = 1$. Data augmentation consists of a random resized crop and horizontal flip. For both the joint and incremental setting we train with a batch size of 32 and 30 epochs. The difference being for the joint variant we report in the main paper we start with the jointly trained backbone then train with the incremental variant of TAPS as opposed to an generic model for incremental training. When we use the joint version in which the backbone is trained simultaneously with the weight deltas, we train with batch size 72 and sample 12 images per task

for each batch for 60 epochs.

C. Additional Results

C.1. Imagenet to Sketch Benchmark

Amount of Additional Parameters. Tab. 6 shows the per-

Flowers	WikiArt	Sketch	Cars	CUB
% Addl. Params.				
65.50	52.82	75.87	41.85	70.61

Table 6. **Additional parameters for ResNet-50 model on ImageNet-to-Sketch benchmark**. Amount of Additional parameters (percentage) for TAPS are shown for the different datasets.

centage of additional parameters needed for each dataset of this benchmark. In total we use about $4.12\times$ the number of parameters, with Cars using the least parameters and Sketch using the most.

Layers active for the Sketch dataset for various λ. We show in Fig. 6, the layers that are active for various values of λ for the Sketch dataset. From this figure, we see that the first layer is task specific across different values of λ. This shows that the first layer is crucial for fine-tuning. This intuitively makes sense as the Sketch dataset has very different low level statistics compared to ImageNet.

Effect of Pretraining on number of Task specific Parameters. Show in Tab. 7 is the number of task specific layers and parameters that are needed for each dataset. From this table, we see that for all datasets the number of layers that are tasks specific are more for Places pretraining compared to an Imagenet pretraining. One could attribute this to the fact that the Places model specializes in scenes whereas the Imagenet model in objects. Hence, the Imagenet model is "closer" to the tasks as opposed to the Places model.

Layers active for DenseNet-121. Fig. 7 shows the layers that are active for the DenseNet-121 model. From this figure we see that the number of layers that are task specific are much greater compared to the ResNet-50 model. Our hypothesis is that since DenseNet-121 has more skip connections, changing a layer has an effect on more layers as compared to ResNet-50.

C.2. Visual Decathalon

In Tab. 8, we show the percentage of task specific layers as well as parameters for $\lambda = 1.0$ and $\lambda = 0.25$. From this table we see that as λ increases, the percentage of task specific layers and parameters decreases. For the datasets where there are no task specific layers, the increase in parameters is due to the storage of batch norm parameters.

We also show the layers where task specific adaptation is needed for different values of λ for the Visual Decathlon dataset. This can be seen in Fig. 8 ($\lambda = 0.25$), Fig. 9 ($\lambda = 0.5$), Fig. 10 ($\lambda = 0.75$) and Fig. 11 ($\lambda = 1.0$). For all the different values of λ, we see that almost all of the layers below layer 9 are not adapted. For the Aircraft dataset, layer 12 is consistently adapted. Similarly for the DTD dataset we consistently observe that no layers are adapted. This shows that some adaptations are independent of $lambda$ and hence critical for the task.

D. Memory Efficient Joint Variant

See table 9 for comparison between the memory efficient variant and standard TAPS on the DomainNet benchmark for joint multi-task learning.

E. Batch Norm and Manual Freezing

In Tab. 10 we report the results of only changing batch norm parameters and manually freezing layers to match the parameter cost of TAPS. We find that adaptively modifying layers outperforms manual freezing.

F. PyTorch Implementation

We show the code snippet of a PyTorch implementation of TAPS in Algorithm 1 and Algorithm 2. The AdaptiveConv conv layers can be used to replace the normal Conv2d layers in an existing model. This shows how easily existing architectures can be used for training TAPS.

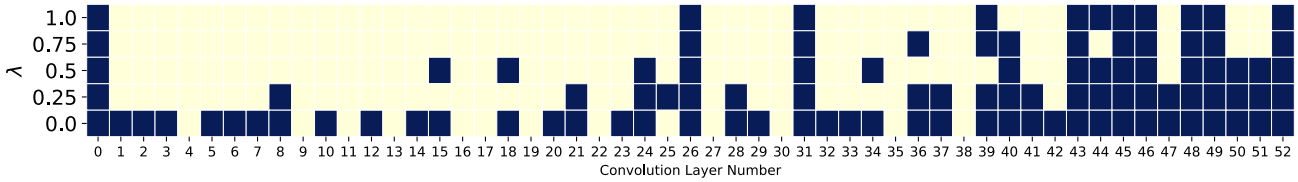

Figure 6. **When do we use task specific weights?.** The plot shows the different task specific layers for the a model trained on the Sketch dataset. Each row represents the value of λ for which the network uses task specific weights. Blue boxes indicate task specific layers while yellow boxes indicate base model layers. From this figure we can see that the lowest layers are active for any value of λ. Most other lower layers are turned off when $\lambda > 0$

	Flowers	WikiArt	Sketch	Cars	CUB
Resnet - 50					
% Addl. Parameters					
Imagenet	65.50	52.82	75.87	41.85	70.61
Places	75.08	68.72	74.59	75.02	74.72
% Task Specific Layers					
Imagenet	22.64	20.75	43.40	14.47	28.30
Places	35.85	28.30	50.94	33.96	33.96

Table 7. **Effect of Pretrained Model on Task specific parameters.** Amount of parameters that are additionally needed for each task. The comparison between Imagenet pretrained model and Places pretrained model is shown. Across the board we see that the Places pretrained model uses more parameters as well as layers.

Method	Airc.	C100	DPed	DTD	GTSR	Flwr.	Oglt.	SVHN	UCF	Mean.	S-Score
TAPS(λ=1.0)	63.43	81.04	96.99	58.19	98.38	84.08	89.16	94.99	51.10	77.77	3088
% Addl. Parameters	32.95	30.43	0.13	20.38	0.13	20.33	47.45	20.41	40.53		
% Task specifc layer	16.00	12.00	0.00	8.00	0.00	8.00	20.00	8.00	16.00		
TAPS((λ=0.25)	66.58	81.76	97.07	58.83	99.07	86.99	88.79	95.72	51.92	78.703	3532
% Addl. Parameters	80.92	60.73	0.13	53.28	20.38	40.53	66.38	40.69	60.72		
% Task specific layer	44.00	24.00	0.00	24.00	8.00	16.00	28.00	16.00	24.00		

Table 8. **Visual Decathalon additional parameter count.** Shown in this table is the percentage of task specific layers as well as parameters needed for each task in addition to the base model. We show this for $\lambda = 1.0$ and $\lambda = 0.25$

Method	Params	Real	Painting	Quickdraw	Clipart	Infograph	Sketch	Mean
TAPS (Standard)	1.43×	78.45	**68.23**	**70.32**	**77.00**	**39.35**	67.95	**66.88**
TAPS (Mem. Efficient)	1.46×	**78.91**	67.91	70.18	76.98	39.30	**67.81**	66.84

Table 9. **Comparison between the memory efficient and standard version of TAPS on DomainNet in the joint MTL setting.** The memory efficient version performs comparably in accuracy and parameter cost while only needing constant memory during training.

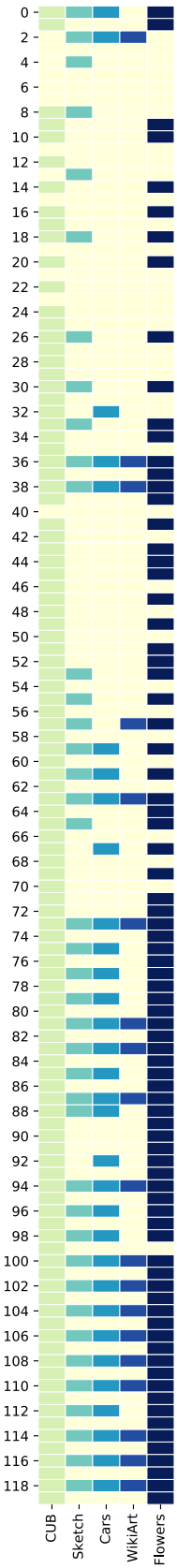

Figure 7. **Shared layers for different tasks.** The figure shows the task specific layers that are active for different datasets using a DenseNet-121 model. We observe that compared to the ResNet-50 model, many more layers are active for the same dataset.

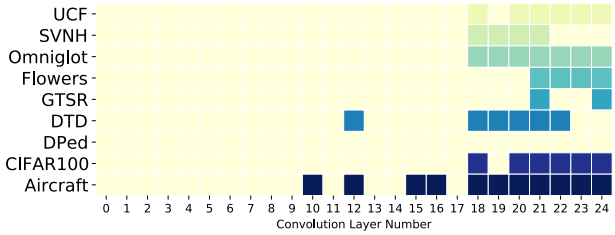

Figure 8. **Task specific layers for different tasks.** The figure shows the task specific layers that are adapted for different datasets in the Visual Decathlon challenge for $\lambda = 0.25$

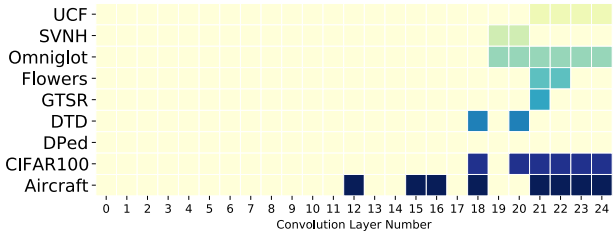

Figure 9. **Task specific layers for different tasks.** The figure shows the task specific layers that are adapted for different datasets in the Visual Decathlon challenge for $\lambda = 0.50$

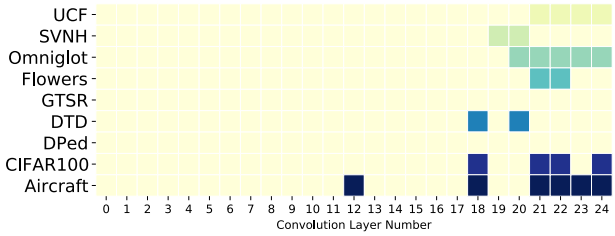

Figure 10. **Task specific layers for different tasks.** The figure shows the task specific layers that are adapted for different datasets in the Visual Decathlon challenge for $\lambda = 0.75$

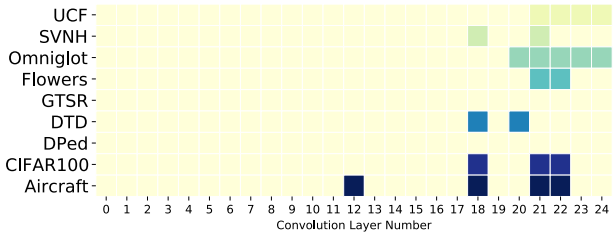

Figure 11. **Task specific layers for different tasks.** The figure shows the task specific layers that are adapted for different datasets in the Visual Decathlon challenge for $\lambda = 1.0$

Algorithm 1 Pytorch Code for Gating Function with Straight Through Estimator

```
class BinarizeIndictator(autograd.Function):
    @staticmethod
    def forward(ctx, indicator, threshold=0.1):
        out = (indicator >= threshold).float()
        return out
    @staticmethod
    def backward(ctx, g):
        # send the gradient g straight-through on the backward pass.
        return g, None
```

Algorithm 2 Pytorch Code for Incremental Version of Task-Adaptive Convolutional Layers

```
class AdaptiveConv(nn.Conv2d):
    def __init__(self, *args, **kwargs):
        super().__init__(initial_val, *args, **kwargs)
        weight_shape = self.weight.shape
        #Initialize residual weights and indicator scores.
        self.residual = torch.nn.Parameter(torch.zeros(weight_shape))
        self.indicator = torch.nn.Parameter(torch.ones([initial_val]))
        #Freeze base network weights
        self.weight.requires_grad = False
        self.weight.grad = None


    def forward(self, x):
        I = BinarizeIndictator.apply(self.indicator)
        w = self.weight + I * self.residual
        x = F.conv2d(x, w)
        return x
```

	Param	Flowers	WikiArt	Sketch	Cars	CUB
Feat. Extractor	1×	89.14	61.74	65.90	55.52	63.46
BN	1.01×	91.07	70.06	78.47	79.41	76.25
Man. Freeze	4.15×	92.35	73.32	78.68	87.62	81.01
TAPS	4.12×	96.68	76.94	80.74	89.76	82.65

Table 10. **Performance of various method using a ResNet-50 model on ImageNet-to-Sketch benchmark.**