9 Appendix

9.1 Related Work

Domain Adaptation There is a long history of research in domain adaptation techniques, which aim at transferring knowledge from one or multiple source domains to a target domain with a different data distribution. Early methods have generally relied on the adaptation of *shallow* classification models, using techniques such as instance re-weighting [12] and model parameter adaptation [65]. More recently, many methods have been proposed to address the problem of domain adaptation using deep neural networks, including discrepancy-based methods, designed to align marginal distributions between the domains [38,54,23,30], adversarial-based approaches, which rely on a domain discriminator to encourage domain-independent feature learning [59,14], and reconstruction-based techniques, which generally use encoder-decoder models or GANs to reconstruct data in the new domain [4,69,22]. All these approaches, however, consider the case that the training and test sets have the *same classes*. One work considers the scenario where some classes may be disjoint, but still requires class overlap for successful alignment [50]. In contrast, we study the problem of cross-domain few-shot learning, where the source and target domains have completely *disjoint label sets*.

9.2 Visualization

In this section, we visualize for each dataset which pre-trained models are selected in the studied incremental multi-model selection. The experiments are conducted on 5-way 50-shot with all five pre-trained models. For each dataset, we repeat the experiments for 600 episodes and calculate the frequency of each model being selected. The results are shown in Figure 3. We observe the distribution of the frequency differs significantly across datasets, as target datasets can benefit from different pre-trained models.

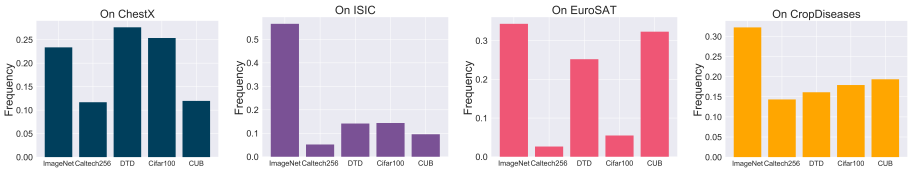

Fig. 3: Frequency of source model selection for each dataset in the benchmark.

We further analyze how layers are changed during transfer. We use θ to denote the original pre-trained parameters and $\hat{\theta}$ to denote the parameters after fine-tuning. Figure 4 shows the relative parameter change of the ResNet10 mini-ImageNet pre-trained model as $\frac{|\theta - \hat{\theta}|}{|\theta|}$, averaged over all parameters per layer, and 100 runs. Several interesting observations can be made from these results. First, across all the datasets and all the shots, the first layer of the pre-trained

model changes most. This indicates that if the target domain is different from the source domain, the lower layers of the pre-trained models still need to be adjusted. Second, while the datasets are drastically different, we observe that some layers are consistently more *transferable* than other layers. One plausible explanation for this phenomenon is the heterogeneous characteristic of layers in overparameterized deep neural networks [68].

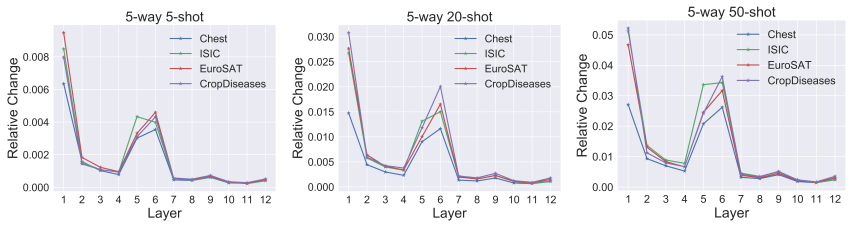

Fig. 4: Relative change of pre-trained network layers for single model transfer.

9.3 Incremental Multi-model Selection

Algorithm 1: *Incremental Multi-model Selection.* $S = \{x_i, y_i\}_{i=1}^{K \times N}$ is a support set consisting of N examples from K novel classes. Assume there is a library of C pre-trained models $\{M_c\}_{c=1}^{C}$. Each model has L layers and l is used to denote one particular layer. Let $CW(S, I)$ be a function which returns the average cross-validation error given a dataset S and a set of layers I which are used to generate feature vector.

```
   /* First stage                                                    */
 1 I_1 = {}
   /* Iterate over each pre-trained model                            */
 2 for c = 1 → C do
 3     min_loss = -1
 4     best_l = None
       /* Iterate over each layer of the pre-trained model           */
 5     for l = 1 → L do
 6         if CW(S, {l}) < min_loss then
 7             best_l = l
 8             min_loss = CW(S, {l})
 9     I_l = I_l ⋃ best_l
   /* Second stage                                                   */
10 I = {}
11 min_loss = -1
12 for each l in I_1 do
13     if CW(S, I ⋃ l) < min_loss then
14         min_loss = CW(S, I ⋃ l)
15         I = I ⋃ l
16 Concatenate the feature vectors generated by the layers in I and train a linear
     classifier.
```