
SUPPLEMENTAL MATERIAL

XtarNet: Learning to Extract Task-Adaptive Representation for Incremental Few-Shot Learning

Sung Whan Yoon*¹ Do-Yeon Kim*² Jun Seo² Jaekyun Moon²

1. Details of Backbone Networks

For the *miniImageNet* experiments of XtarNet, we utilized ResNet12, a residual network with 12 convolutional layers, as architecture of the backbone network. ResNet12 consists of four residual blocks having 64, 96, 128 and 256 respective channels. We use a LeakyReLU for activation function and the slope is 0.1 at $x < 0$. Each residual block contains three 3×3 convolutional layers, where every convolutional layer is followed by a batch normalization and an activation function. For the last layer of the fourth residual block, we do not use an activation function. The residual connection of each block consists of a single 3×3 convolutional layer and a batch normalization layer. At the end of each residual block, 2×2 max-pooling is applied. To reduce the dimension of the feature vector, a global average pooling is applied to the output of the last residual block. For every residual block, we use a dropout layer with a drop ratio of 0.2 after the max-pooling layer. For all methods other than XtarNet, the difference is that we use a ReLU for activation function and do not remove the activation at the last layer of the fourth residual block.

For the *tieredImageNet* experiments of XtarNet, we employed the standard ResNet18, which consists of 18 convolutional layers with identity mapping, as the backbone network. We use a ReLU for the activation function. Down-sampling layer is adopted in residual connection if needed; average pooling and zero-padding are applied to match the feature map size and the number of channel respectively. As the *miniImageNet* cases, we also do not use an activation at the last convolutional layer of the last residual block. ResNet18 is well-known, and we forgo a detailed description here. For all other methods except for XtarNet, the

*Equal contribution ¹School of Electrical and Computer Engineering, Ulsan National Institute of Science and Technology (UNIST), Ulsan, Korea ²School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. Correspondence to: Sung Whan Yoon <shyoon8@unist.ac.kr>.

difference is that we do not remove an activation at the last layer of the last residual block and use a 1×1 convolutional layer followed by batch normalization layer for down-sampling to match the dimension.

2. Details of Meta-Trained Modules

The details of MetaCNN module are described in the main paper. We utilize fully-connected networks for MergeNet and TconNet. For the *miniImageNet* experiments, MergeNet consists of two separate 4-layer fully-connected networks. These two component networks have the same architecture. The size of the input layer of each 4-layer fully-connected network is 512, which is the length of a concatenated feature vector. For the hidden layers and the output layer, the size of each layer is 256. For each of first three fully-connected layers, an ReLU activation is employed. For the last fully-connected layer, a sigmoid layer is employed. TconNet consists of three 3-layer fully-connected networks. Two of them are for computing the conditioning vector for the base classifier. These two fully-connected networks have a skip connection at each layer. The size of each fully-connected layer is 256, which is the length of a combined feature vector. ReLU activation is used at every layer. At each output of these two component fully-connected networks, we multiply the output vector by the constant 0.1 for scaling. The remaining 3-layer fully-connected network is for conditioning of the novel classifier. The size of each fully-connected layer is 64, which is the number of the base categories. The last two fully-connected layers have skip connections. ReLU activation is used at every layer.

For the *tieredImageNet* experiments, the architectures of MergeNet and TconNet are the same as that we used in the *miniImageNet* experiments, except for the size of the fully-connected layers. Because the length of the feature vector becomes double, the size of every fully-connected layer of MergeNet and TconNet also doubles. In TconNet, the size of each fully connected layer for conditioning novel classifier is 200, which is the number of base categories of *tieredImageNet*.

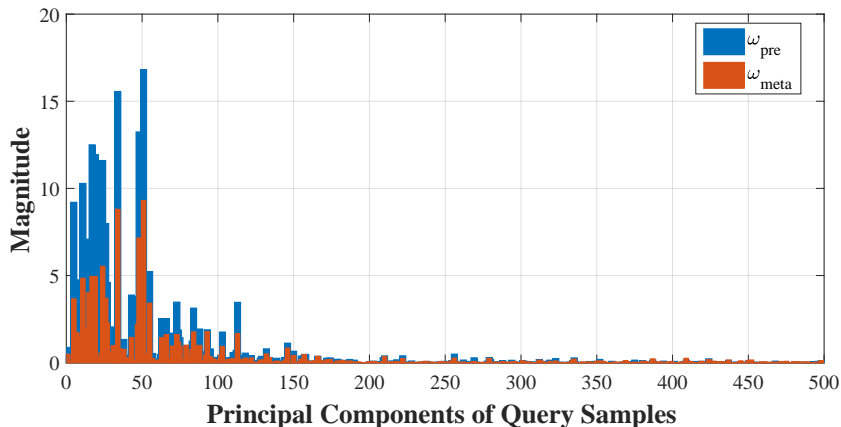


Figure 1. Principal component analysis for mixture weight vectors

3. Hyperparameter Settings

We use $25 \times 2 = 50$ queries in each training episode for the proposed XtarNet algorithm. In the *tieredImageNet* experiments, we also use 50 queries for the TAR method used in conjunction with Imprint of (Qi et al., 2018), LwoF of (Gidaris & Komodakis, 2018) and Attention Attractor Network (AAN) of (Ren et al., 2019), respectively. On the other hand, we used 5×5 for base categories and 25×5 for novel categories in TAR plugged-in Imprint and LwoF for the *miniImageNet* experiments. For all cases, we use the Momentum SGD optimizer with an initial learning rate of 0.1. The learning rate decays by a factor of 10 at every step of 4,000 episodes. l_2 weight decay is set to be 3.0×10^{-3} for all *miniImageNet* experiments and 7.0×10^{-4} for all *tieredImageNet* experiments. Also, the scaling hyperparameters a and b for outputs of meta-trained modules h_γ and h_β of TconNet are both set to 0.1. The 1-shot and 5-shot experiments are done with the same settings.

4. Study on Mixture Weight Vectors

4.1. Principal Component Analysis of Mixture Weight Vectors

We analyze the mixture weight vectors in the combined feature space constructed by query samples. The 5-shot 200+5 *tieredImageNet* experiment is also considered. First, we did a principal component analysis (PCA) of 500 query samples for a test episode. The magnitude of the projected mixture weight vector to each principal component is computed in order to figure out how much power of mixture weight vector aligns with each principal component. In Fig. 1, it is shown that the magnitudes of mixture weight vectors for a backbone and MetaCNN are focused mainly on the first 150 or so principal axes of query samples. Consequently, the mixture weight vectors aim much of their attention at

the critical dimensions where query samples show larger variances.

4.2. Magnitude Analysis

Fig. 2 shows the magnitude of the mixture weight vectors ω_{pre} and ω_{meta} during the meta-training phase. We consider the 5-shot 200+5 *tieredImageNet* case. The mixture weight vector for the feature from the pretrained backbone is seen to have a larger magnitude than the weight vector for the feature from the MetaCNN. The magnitude of ω_{pre} fluctuates and slightly decreases until the learning rate decay at 4,000 episodes. Afterwards, the magnitude gradually increases and eventually saturates. For the mixture weight vector ω_{meta} from the MetaCNN, the magnitude steadily increases over time and also saturates soon. An obvious observation is that the magnitudes of the mixture weight vectors become quite stable before the meta-training phase is over. Also, given the difference in the magnitudes between the two weight vectors, we conjecture that XtarNet more actively utilizes the feature from the pretrained backbone, while also significantly leveraging the feature from MetaCNN.

5. Dataset Statistics

In this section, we present the detailed statistics of the *miniImageNet* and *tieredImageNet* datasets that we use in the incremental few-shot learning tests. We follow the data splits used in prior works of (Gidaris & Komodakis, 2018) and (Ren et al., 2019). The data splits of *miniImageNet* are presented in Table 1. For the *miniImageNet* experiments, the training set $\mathcal{D}_{\text{base/train}}$ (denoted by ‘Train-Train’), which is for pretraining of base classes, is reused as the training set $\mathcal{D}_{\text{novel/train}}$ (denoted by ‘Train-Val’) for novel classes. Since the data split $\mathcal{D}_{\text{novel/train}}$ contains only base classes, some randomly selected base classes are used as fake novel classes. This process is suggested in the prior work (Gidaris

Table 1. Dataset statistics of *miniImageNet*

Classes	Purpose	Split	N_{classes}	N_{samples}
Base ($\mathcal{D}_{\text{base}}$)	Train	Train-Train ($\mathcal{D}_{\text{base/train}}$)	64	38,400
	Val	Train-Val ($\mathcal{D}_{\text{base/val}}$)	64	18,748
	Test	Train-Test ($\mathcal{D}_{\text{base/test}}$)	64	19,200
Novel ($\mathcal{D}_{\text{novel}}$)	Train	Train-Train ($\mathcal{D}_{\text{novel/train}} \leftarrow \mathcal{D}_{\text{base/train}}$)	64	38,400
	Val	Val ($\mathcal{D}_{\text{novel/val}}$)	16	9,600
	Test	Test ($\mathcal{D}_{\text{novel/test}}$)	20	12,000

 Table 2. Dataset statistics of *tieredImageNet*

	Purpose	Split	N_{classes}	N_{samples}
Base ($\mathcal{D}_{\text{base}}$)	Train	Train-A-Train ($\mathcal{D}_{\text{base/train}}$)	200	203,751
	Val	Train-A-Val ($\mathcal{D}_{\text{base/val}}$)	200	25,460
	Test	Train-A-Test ($\mathcal{D}_{\text{base/test}}$)	200	25,488
Novel ($\mathcal{D}_{\text{novel}}$)	Train	Train-B ($\mathcal{D}_{\text{novel/train}}$)	151	193,996
	Val	Val ($\mathcal{D}_{\text{novel/val}}$)	97	124,261
	Test	Test ($\mathcal{D}_{\text{novel/test}}$)	160	206,209

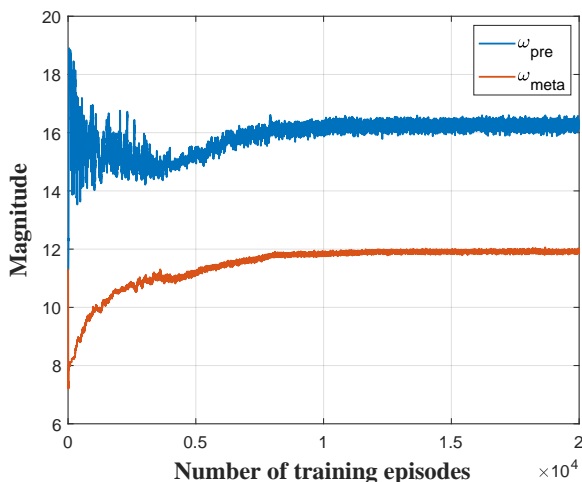


Figure 2. Magnitude of mixture weight vectors during the meta-training phase

& Komodakis, 2018) and (Ren et al., 2019).

On the other hand, for the *tieredImageNet* experiments, the $\mathcal{D}_{\text{base/train}}$ and $\mathcal{D}_{\text{novel/train}}$ are different. The original training split of *tieredImageNet* which contains 351 classes is divided into two splits of ‘Train-A’ and ‘Train-B’ containing disjoint classes. The ‘Train-A’ is then further divided into training/validation/test sets of ‘Train-A-Train’, ‘Train-A-Val’ and ‘Train-A-Test’ with disjoint samples. These splits are for base classes. For novel categories, 151 extra classes of a set ‘Train-B’ are used for novel classes in the meta-training phase. For validation and test, sets denoted by ‘Val’ and

‘Test’ are used respectively. These splits for novel categories contain disjoint classes.

6. Few-shot Classification Accuracies

Mainly out of curiosity, we also evaluate few-shot classification accuracies of XtarNet, which is trained as an incremental few-shot learner supposed to handle both base and novel categories. The accuracy rates on *miniImageNet* datasets, as shown in Table 3, are all from previously published results, except for our XtarNet.

It can be seen that the known incremental few-shot learning methods, LwoF and Attention Attractor Networks, show significantly degraded performance compared to the ResNet12-based learners specialized to few-shot learning. Compared to these prior incremental few-shot learners, our XtarNet shows better accuracies, indicating relatively strong few-shot learning performance although its design aimed at incremental few-shot learning tasks.

7. Algorithmic Description

Algorithm 1 shows a pseudocode-style description of the meta-training stage for XtarNet. For setting the initial novel classifier weights, we adopt meta-learnable references of TapNet (Yoon et al., 2019). Also, the classification of queries is carried out in the task-adaptive projection space.

Table 3. Few-shot classification accuracies for 5-way *miniImageNet*

Methods	backbone	5-way <i>miniImageNet</i>	
		1-shot	5-shot
MAML (Finn et al., 2017)	Conv4	48.70 \pm 1.84%	63.15 \pm 0.91%
Prototypical Nets (Snell et al., 2017)	Conv4	49.42 \pm 0.78%	68.20 \pm 0.66%
Transductive Propagation Nets (Liu et al., 2018)	Conv4	55.51 \pm 0.86%	69.86 \pm 0.65%
TADAM (Oreshkin et al., 2018)	ResNet12	58.5 \pm 0.3%	76.7 \pm 0.3%
TapNet (Yoon et al., 2019)	ResNet12	61.65 \pm 0.15%	76.36 \pm 0.10%
MetaOpt (Lee et al., 2019)	ResNet12	62.64 \pm 0.62%	78.63 \pm 0.46%
Cross Attention Nets* (Hou et al., 2019)	ResNet12	63.85 \pm 0.48%	79.44 \pm 0.34%
LwoF (Gidaris & Komodakis, 2018)	ResNet12	55.45 \pm 0.89%	70.92 \pm 0.35%
Attention Attractor Networks (Ren et al., 2019)	ResNet12	55.75 \pm 0.51%	70.14 \pm 0.44%
XtarNet (Ours)	ResNet12	57.29 \pm 0.54%	75.39 \pm 0.43%

*The Cross Attention Networks version with high-confidence query samples yields higher accuracy but is not shown here for fairness reasons.

Table 4. Results of combining AAN with our TAR for *tieredImageNet*

Methods	1-shot		5-shot	
	Accuracy	Δ	Accuracy	Δ
Attention Attractor Networks* (Ren et al., 2019)	56.11 \pm 0.33%	-6.11%	65.52 \pm 0.31%	-4.48%
Proposed method with AAN	60.48 \pm 0.35%	-1.94%	68.34 \pm 0.33%	-1.69%

*The Attention Attractor Networks results reflect the reported numbers of (Ren et al., 2019).

8. Time Complexity

In Tables 1 and 2 of the main paper, XtarNet shows state-of-the-art accuracies on incremental few-shot learning tasks. AANs show the second best performance. We further remark that our XtarNet has significant advantage in time complexity over AANs. The meta-trained modules in XtarNet operate without any recursive optimization procedures, whereas AANs require several tens of inner-loop optimization steps for obtaining novel classifier weights. See Step 6 through 9 in Algorithm 1 of (Ren et al., 2019). This recursive process in AANs causes a substantial latency relative to XtarNet during query processing in both training and inference. Furthermore, during training, AANs actually employ additional inner-loop for computing gradients based on the recurrent back-propagation (RBP) algorithm. Steps 15 through 17 of Algorithm 1 of (Ren et al., 2019) represent the second inner loop. This further slows down meta-training of the learner relative to our method.

9. Additional Experiments for Combining Attention Attractor Network with TAR

In an effort to further evaluate the expandability of the proposed TAR, we run our method on top of AANs (Ren et al., 2019) for *tieredImageNet* as shown in Table 4. As mentioned in the main paper, combining AANs with our proposed method is possible, although the combining strategy is not straightforward. The most challenging part is optimizing

all of our modules along with the meta-learnable parameters of AANs integrated with the inner loop optimization procedure. In AANs, the attention-based regularizer is meta-trained to regularize a novel classifier in a way that novel classes predictions do not interfere the base classes during the inner loop optimization for each episode. Our strategy is to use TAR constructed by MetaCNN and MergeNet, and the base classifier conditioned by TconNet for computing this regularization term. We drop the meta-module for conditioning novel classifier h_λ for simplicity. Naively computing the gradients with respect to all of these modules during the RBP step does not work well in practice, perhaps due to the large number of parameters to be optimized. In order to get around this computational burden, we compute the gradients only with respect to the meta parameters of AANs in the RBP step, and then backpropagate with respect to our meta-modules outside of the inner loop independently, given a novel classifier prepared from AANs. We ran experiment on the same backbone architecture used in (Ren et al., 2019) to gauge the effectiveness of our method. As shown in Table 4, we obtained a 68.34% in accuracy and -1.69% in Δ value for the 5-shot *tieredImageNet* experiments. For the 1-shot case, TAR combined with AANs shows 60.48% in accuracy and -1.94% in Δ value. These are considerable gains in both accuracies and Δ values compared to AANs alone.

Algorithm 1 Description of episodic learning during meta-training of XtarNet. The pretrained backbone and base classification weights are prepared before meta-training begins.

Input: Backbone f_θ ; base classifier weights $\{\mathbf{w}_i\}_{i=1}^{N_b}$; MetaCNN g ; MergeNet r_{pre} and r_{meta} ; TconNet h_γ , h_β and h_λ ; novel classifier weights $\{\mathbf{w}_i\}_{i=N_b+1}^{N_b+N}$

```

1: for  $t = 1 \dots T$  do
2:    $L_{\text{train}} \leftarrow 0$ 
3:    $\mathcal{S}, \mathcal{Q} \leftarrow \text{GetEpisode}(\mathcal{D}_{\text{base/train}}, \mathcal{D}_{\text{novel/train}})$  // support set  $\mathcal{S}$  and query set  $\mathcal{Q}$ 
4:    $\mathbf{c} \leftarrow \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x} \in \mathcal{S}} [f_\theta(\mathbf{x}), g(a_\theta(\mathbf{x}))]$  //  $a_\theta(\mathbf{x})$  is an intermediate layer output of  $f_\theta(\mathbf{x})$ .
5:    $\boldsymbol{\omega}_{\text{pre}} \leftarrow r_{\text{pre}}(\mathbf{c}), \boldsymbol{\omega}_{\text{meta}} \leftarrow r_{\text{meta}}(\mathbf{c})$ 
6:   for  $k = N_b + 1, \dots, N_b + N$  do
7:      $\mathbf{c}_k^* \leftarrow \frac{1}{|\mathcal{S}_k|} \sum_{\mathbf{x} \in \mathcal{S}_k} [\boldsymbol{\omega}_{\text{pre}} \odot f_\theta(\mathbf{x}) + \boldsymbol{\omega}_{\text{meta}} \odot g(a_\theta(\mathbf{x}))]$  // per-class averages of combined features
8:   end for
9:    $\mathbf{c}^* \leftarrow \frac{1}{N} \sum_{k=N_b+1}^{N_b+N} \mathbf{c}_k^*$ 
10:  for  $i = 1, \dots, N_b$  do
11:     $\mathbf{w}_i^* \leftarrow (\mathbb{1} + ah_\gamma(\mathbf{c}^*)) \odot \mathbf{w}_i + bh_\beta(\mathbf{c}^*)$  // hyperparameters  $a$  and  $b$  are set to 0.1.
12:  end for
13:  for  $k = N_b + 1 \dots N_b + N$  do
14:    for  $i = 1, \dots, N_b$  do
15:       $\sigma_k^i \leftarrow \mathbf{c}_k^* \cdot \mathbf{w}_i^*$ 
16:    end for
17:     $\boldsymbol{\sigma}_k \leftarrow [\sigma_k^1, \dots, \sigma_k^{N_b}]$ 
18:     $\boldsymbol{\lambda}_k \leftarrow h_\lambda(\boldsymbol{\sigma}_k)$ 
19:  end for
20:  for  $i = N_b + 1 \dots N_b + N$  do
21:     $\mathbf{w}_i^* = \mathbf{w}_i - \sum_{j=1}^{N_b} \frac{\exp(\lambda_i^j)}{\sum_l \exp(\lambda_i^l)} \mathbf{w}_j^*$  //  $\lambda_i^j$  is  $i^{\text{th}}$  element of  $\boldsymbol{\lambda}_k$ .
22:  end for
23:  for  $k = N_b + 1, \dots, N_b + N$  do
24:     $\boldsymbol{\epsilon}_i \leftarrow \mathbf{w}_k^* / \|\mathbf{w}_k^*\| - \mathbf{c}_k^* / \|\mathbf{c}_k^*\|$ 
25:  end for
26:   $\mathbf{M} \leftarrow \text{null}\{[\boldsymbol{\epsilon}_{N_b+1}, \dots, \boldsymbol{\epsilon}_{N_b+N}]\}$  // task-adaptive projection space
27:  for  $i = 1, \dots, N_b + N$  do
28:    for  $(\mathbf{x}, y = i) \in \mathcal{Q}$  do
29:       $D_i(\mathbf{x}) = d\left(\mathbf{w}_i^* \mathbf{M}, [\boldsymbol{\omega}_{\text{pre}} \odot f_\theta(\mathbf{x}) + \boldsymbol{\omega}_{\text{meta}} \odot g(a_\theta(\mathbf{x}))] \mathbf{M}\right)$  //  $d(\cdot, \cdot)$ : Euclidean distance
30:       $L_{\text{train}} = L_{\text{train}} + \frac{1}{|\mathcal{Q}|} \left[ D_i(\mathbf{x}) + \log \sum_l \exp(-D_l(\mathbf{x})) \right]$  // cross entropy loss
31:    end for
32:  end for
33:  Update all parameters of  $g, r_{\text{pre}}, r_{\text{meta}}, h_\gamma, h_\beta, h_\lambda, \{\mathbf{w}_i\}_{i=N_b+1}^{N_b+N}$  minimizing  $L_{\text{train}}$  via optimizer
34: end for

```

References

- Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pp. 1126–1135, 2017.
- Gidaris, S. and Komodakis, N. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4367–4375, 2018.
- Hou, R., Chang, H., Bingpeng, M., Shan, S., and Chen, X. Cross attention network for few-shot classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4003–4014, 2019.
- Lee, K., Maji, S., Ravichandran, A., and Soatto, S. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10657–10665, 2019.
- Liu, Y., Lee, J., Park, M., Kim, S., Yang, E., Hwang, S. J., and Yang, Y. Learning to propagate labels: Transductive propagation network for few-shot learning. In *International Conference on Learning Representations (ICLR)*, 2018.
- Oreshkin, B. N., Lacoste, A., and Rodriguez, P. TADAM: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 721–731, 2018.
- Qi, H., Brown, M., and Lowe, D. G. Low-shot learning with imprinted weights. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 5822–5830, 2018.
- Ren, M., Liao, R., Fetaya, E., and Zemel, R. Incremental few-shot learning with attention attractor networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 5276–5286, 2019.
- Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4080–4090, 2017.
- Yoon, S. W., Seo, J., and Moon, J. TapNet: Neural network augmented with task-adaptive projection for few-shot learning. In *International Conference on Machine Learning (ICML)*, pp. 7115–7123, 2019.