Supplementary Materials

7. Pseudo-Codes for Calculating $\bar{Q}_c$

Eq. 10: $\bar{Q}_c = \rho\bar{W}S_c = \rho Q(S_c^T S_c + \lambda I)^{-1} S_c^T S_c$. It is more computation-efficient than Eq. 8 when $kr > d$. For consistency, we use this formula in our implementation for all experiments. The corresponding Pytorch-style pseudo-code can be found in Listing 1.

```
# input tensors and scalars:
# query: feature maps of query images, the tensor shape is [way*query_shot*r, d]
# support: feature maps of support images, the tensor shape is [way, support_shot*r, d]
# lam: lambda
# rho

# output: Q_bar


def get_Q_bar(query,support,lam,rho):

    st = support.permute(0,2,1)
    sts = st.matmul(support)
    m_inv = (sts+torch.eye(sts.size(-1)).unsqueeze(0).mul(lam)).inverse()
    hat = m_inv.matmul(sts)
    Q_bar = query.matmul(hat).mul(rho)

    return Q_bar
```

Listing 1. Pytorch pseudo-code for feature map reconstruction in Eq. 10 for a single meta-training episode. The whole calculation can be performed in parallel via batched matrix multiplication and inversion.

Eq. 8: $\bar{Q}_c = \rho\bar{W}S_c = \rho Q S_c^T (S_c S_c^T + \lambda I)^{-1} S_c$. It is more computation-efficient than Eq. 10 when $d > kr$. The corresponding Pytorch-style pseudo-code can be found in Listing 2.

```
def get_Q_bar(query,support,lam,rho):

    st = support.permute(0,2,1)
    sst = support.matmul(st)
    m_inv = (sst+torch.eye(sst.size(-1)).unsqueeze(0).mul(lam)).inverse()
    hat = st.matmul(m_inv).matmul(support)
    proj = query.matmul(hat).mul(rho)

    return Q_bar
```

Listing 2. Pytorch pseudo-code for feature map reconstruction in Eq. 8 for a single meta-training episode. The whole calculation can be performed in parallel via batched matrix multiplication and inversion. Details about inputs can be found in Listing 2.

8. Ablation Studies

Numerical results and deeper discussion from the main paper.

8.1. Episodic Training Shot Number

In our experiments, we surprisingly found that FRN trained with 5-shot episodes consistently outperforms FRN trained with 1-shot episodes, even on 1-shot evaluation. For the sake of brevity, we only report the superior performance from the FRN trained with 5-shot episodes in the main paper and include the performance of FRN trained with 1-shot episodes as an ablation in Table 11. We also include the best-performing baseline from each experimental setting in the main paper for comparison. Under the standard 5-way 1-shot evaluation, 1-shot trained FRN outperforms the baseline in most fine-grained settings, but also underperforms the 5-shot trained models in all settings.

8.2. Auxiliary Loss and Regularization Parameters

Ablation results for these components on cropped CUB are given in Table 12. In this case, 1-shot results come from models trained with 1-shot episodes (as in Sec. 8.1), rather than 5-shot (as in the main paper). We find that the auxiliary loss has little to no consistent impact on FRN performance. We include it for our experimental models only for the sake of

Model (Conv-4)	Training scheme	CUB (crop)		Aircraft	meta-iNat	tiered meta-iNat
Best Baseline	1-shot	69.64±0.23		49.67±0.22	60.03±0.23	36.83±0.18
FRN (ours)	1-shot	68.27±0.23		50.25±0.21	60.04±0.22	40.11±0.18
FRN (ours)	5-shot	73.48±0.21		53.20±0.21	62.42±0.22	43.91±0.19
Model (ResNet-12)	**Training scheme**	**CUB (crop)**	**CUB (uncrop)**	**Aircraft**	**mini-ImageNet**	**tiered-ImageNet**
Best Baseline	1-shot	80.80±0.20	79.96±0.21	68.16±0.23	66.78±0.20	71.52±0.69
FRN (ours)	1-shot	82.62±0.19	82.02±0.20	69.40±0.23	65.33±0.20	70.88±0.22
FRN (ours)	5-shot	83.16±0.19	83.55±0.19	70.17±0.22	66.45±0.19	72.06±0.22

Table 11. 5-way 1-shot performance comparison for FRN trained with 1-shot and 5-shot episodes, along with the best reported baseline for reference. Results for FRN are averaged over 10,000 trials with 95% confidence interval.

	Conv-4		ResNet-12	
Model	**1-shot**	**5-shot**	**1-shot**	**5-shot**
no Aux	**68.38±0.23**	88.13±0.13	82.20±0.20	92.65±0.10
fixed λ	66.44±0.23	84.05±0.15	82.40±0.20	93.12±0.10
fixed ρ	67.34±0.23	87.60±0.13	82.58±0.20	92.79±0.10
fixed λ, ρ	66.19±0.23	83.27±0.15	**82.80±0.19**	**93.33±0.10**
whole model	68.27±0.23	**88.43±0.13**	82.62±0.19	92.59±0.10

Table 12. Ablation study on FRN regularization parameters and auxiliary loss using the cropped CUB benchmark.

apples-to-apples comparison with baseline models that do rely on it, such as DSN [29] (we verified in early experiments that our proxy DSN† implementation also suffers when the auxiliary loss is removed).

To analyze the contribution of the learned regularization terms λ and ρ, we disable their learnability, setting α, β, or both equal to zero over the course of training. The impact of these terms is mixed. The 4-layer network clearly benefits from learning these terms, but the ResNet-12 architecture benefits from removing them – though by only a small, possibly insignificant margin. It seems that the more powerful network is already able to overcome any regularization issues, by massaging the feature space in a more elegant way than the individual λ, ρ terms can provide.

9. Training Details

Pre-processing: During training on Aircraft, CUB, mini-ImageNet and tiered-ImageNet (data from DeepEMD [43]'s implementation[5]), images are randomly cropped and then resized into 84×84[6]. For meta-iNat, tiered meta-iNat, and tiered-ImageNet using the original release by Ren *et al*. [27][7], images are randomly padded then cropped into 84×84[8]. Then all images are augmented by color-jittering and random horizontal flipping.

During inference on Aircraft and CUB (using bounding-box cropped images as input), images are resized into 92×92 then center-cropped into 84×84[9]. For mini-ImageNet, tiered-ImageNet (data from DeepEMD [43]'s implementation) and CUB (using raw images as input), images are resized such that the smaller edges will match the length of 92[10] then center-cropped into 84×84. This prevents distortion artifacts at test time. For meta-iNat, tiered meta-iNat, and tiered-ImageNet using the original release by Ren *et al*. [27], images are already at size 84×84 and are therefore fed into models directly.

Many of these choices were inherited from the original datasets and code. They do not reflect hyperparameter tuning on any of our models.

Network Backbones: We use two neural architectures in our experiments: Conv-4 and ResNet-12. Conv-4 consists of four convolutional layers with kernel size 3×3 and output size 64, followed by BatchNorm, ReLU, and 2×2 max-pooling. ResNet-12 consists of four residual blocks, each with three convolutional layers, with leaky ReLU (0.1) and 2×2 max-pooling on the main stem. We use drop-block as in the original implementation [34, 42, 18][11][12][13]. Output sizes for each residual block are: $64, 160, 320, 640$.

Hyperparameters: Unless otherwise stated, all models are trained with a weight decay of 5e-4. We temperature scale

[5] https://github.com/icoz69/DeepEMD

[6] i.e., `torchvision.transforms.RandomResizedCrop(84)`

[7] https://github.com/renmengye/few-shot-ssl-public

[8] i.e., `torchvision.transforms.RandomCrop(84, padding=8)`

[9] i.e., `torchvision.transforms.Resize([92,92])` followed by `torchvision.transforms.CenterCrop(84)`

[10] i.e., `torchvision.transforms.Resize(92)`

[11] https://github.com/WangYueFt/rfs

[12] https://github.com/Sha-Lab/FEAT

[13] https://github.com/kjunelee/MetaOptNet

all output probability logits, and also normalize these logits for our baseline models, dividing by 64 for Conv-4 and 640 for ResNet-12. FRN models do not benefit from this normalization, as logit scale does not correspond well to embedding scale due to the large influence of the regularization term. We found it necessary for stable training to instead normalize the FRN ResNet-12 *embeddings* by downscaling by a factor of $\sqrt{640}$. Note that this is algebraically equivalent to the existing logit-based normalization for the ProtoNet† and DSN† baselines, as well as CTX† up to a rescaling factor in the inner softmax term (see Eq. 14).

Training details for each individual benchmark are provided below. We found that our prototypical network hyperparameters worked well for all baselines, and for fair comparison, do not tune FRN hyperparameters separately (beyond addressing obvious cases of training instability). Thus we give one set of hyperparameters for all models, with exceptions noted.

While we provide training details for 1-shot models, note that in our benchmark experiments these are only relevant to our implemented baselines (i.e. ProtoNet†, DSN† and CTX†). 1-shot trained FRN models appear only in the ablation studies of Sec. 5.1 and Sec. 8. FRN results reported in the main paper all come from models trained with 5-shot episodes.

9.1. Fine-Grained Few-Shot Benchmarks

CUB: Our implementation follows [13, 18]. We train all Conv-4 models for 800 epochs using SGD with Nesterov momentum of 0.9 and an initial learning rate of 0.1. The learning rate decreases by a factor of 10 at epoch 400. ResNet-12 models train for 1200 epochs and scale down the learning rate at epochs 400 and 800. We use the validation set to select the best-performing model over the course of training, and validate every 20 epochs.

Conv-4 models are trained with the standard episodic setup: 20-way 5-shot for 5-shot models, and 30-way 1-shot for 1-shot models. We use 15 query images per class in both settings. In order to save memory we cut these values in half for ResNet-12 models: 5-shot models train on 10-way episodes, while 1-shot models train on 15-way episodes.

Aircraft: Our implementation for the aircraft dataset roughly follows the CUB setup. Conv-4 models train for 1200 epochs, however, and we found that the 1-shot ResNet-12 ablation FRN is highly unstable at the very beginning of training and frequently collapses to the suboptimal uniform solution. We found it necessary to train this model using Adam with an initial learning rate of 1e-3, and no weight decay. All other hyperparameters remain the same as for CUB.

meta-iNat and tiered meta-iNat: Our implementation follows [41], only we train for twice as long, as we found that the loss curves frequently failed to stabilize before each decrease in learning rate. We therefore train our models for 100 epochs using Adam with initial learning rate 1e-3. We cut the learning rate by a factor of two every 20 episodes. Because there is no validation set available, we simply use the final model at the end of training. Episode setup is as in CUB and Aircraft.

9.2. General Few-Shot Benchmarks

mini-ImageNet: For non-episodic FRN pre-training, we run 350 epochs with batch size 128, using SGD with initial learning rate 0.1 and Nesterov momentum 0.9. We cut the learning rate by a factor of 10 at epochs 200 and 300. We also use the validation set to select the best-performing model over the course of pre-training, and validate every 25 epochs. Subsequent episodic fine-tuning uses the same optimizer for 150 epochs, but with initial learning rate 1e-3, decreased by a factor of 10 at epochs 70 and 120. For the 5-shot FRN model, we use the standard 20-way episodes. The 1-shot ablation model uses 25-way episodes.

FRN models trained from scratch (for the ablation study) use the same optimizer for 300 epochs, with initial learning rate 0.1, decreased by a factor of ten at epochs 160 and 250. Similar to CUB, we cut the way of the training episodes in half in order to reduce memory footprint: 10-way for 5-shot models, and 15-way for 1-shot ablation models.

For all the episodic meta-training processes mentioned above, we continue to use the validation set to select the best-performing model by validating every 5 epochs for fine-tuning and every 20 epochs for training from scratch.

tiered-ImageNet: For non-episodic FRN pre-training, we run 90 epochs with batch size 128, using SGD with initial learning rate 0.1 and Nesterov momentum 0.9. We cut the learning rate by a factor of 10 at epochs 30 and 60. We also use the validation set to select the best-performing model over the course of pre-training, and validate every 5 epochs. Subsequent episodic fine-tuning uses the same optimizer for 60 epochs, but with initial learning rate 1e-3, decreased by a factor of 10 at epochs 30 and 50. For the 5-shot FRN model, we use the standard 20-way episodes. The 1-shot ablation model uses 25-way episodes. We also use the validation set to select the best-performing model during episodic fine-tuning, validating every 5 epochs.

For the experiment on the tiered-ImageNet dataset provided by DeepEMD [43]'s implementation[14], since images have larger resolution, we train slightly longer during the episodic fine-tuning: 70 epochs in total, learning rate decreased by a factor of 10 at epochs 40 and 60. All other hyperparameters remain the same as above.

[14] https://github.com/icoz69/DeepEMD

Decoder Network:	
Layer	**Output Size**
Input	$640 \times 5 \times 5$
ResBlock	$512 \times 10 \times 10$
ResBlock	$256 \times 20 \times 20$
ResBlock	$128 \times 40 \times 40$
ResBlock	$64 \times 80 \times 80$
Upsample	$64 \times 84 \times 84$
Conv3x3	$3 \times 84 \times 84$
Tanh	$3 \times 84 \times 84$

Residual Block:	
Input	
TranspConv4x4, stride 2	Conv3x3
	BatchNorm
	ReLU
	TranspConv4x4, stride 2
BatchNorm	BatchNorm
	ReLU
	Conv3x3
	BatchNorm
Addition	
ELU	

Table 13. Neural architecture for the image decoder network (left) and residual block (right). Transposed convolution layers have half as many output channels as input channels.

9.3. Implemented Baselines

While our implemented baselines closely mirror the models that appear in prior work, we do introduce slight differences in order to provide the most direct comparison to FRN as possible. Generally, we found these modifications actually improved performance. We enumerate differences baseline by baseline.

ProtoNet†: All of our baselines, including prototypical networks, incorporate temperature scaling with initial value set to $\frac{1}{d}$, where d is the latent dimensionality of the feature extractor. While the original prototypical network paper [30] did not incorporate this practice, it has become common in subsequent work and has been shown to improve performance [13, 18, 42].

DSN†: Our implementation of DSN is mathematically equivalent to the original [29], but is implemented differently. Our method calculates the closed-form solution to the regression objective, which is equivalent to calculating the projection of the query point into the subspace defined by the supports (for sufficiently small regularizer λ; we use 0.01). [29] instead calculates an orthonormal basis that spans the support points using SVD, and zeros out the components of the query point that fall into that subspace, yielding the displacement from the projection. While the underlying mechanics are different, the end result is the same.

The original DSN also recenters the projection problem about each class centroid – that is, DSN projects each query point onto the hyperplane containing the support points but not necessarily the origin. In our implementation and in FRN, we define the projection subspace as the hyperplane containing the support points and the origin (that is, we maintain the origin as a common reference point). We found this choice made little difference in practice, and including the origin / not recentering actually improved DSN performance slightly.

Finally, the true DSN auxiliary loss encourages orthogonality between the calculated orthonormal bases for each class. This is a cheap operation for the original DSN model, as these bases are pre-calculated from the prediction step. This is not true for our model; we instead utilize the same loss formula in Eq. 11 but compare the actual (normalized) support points rather than their orthonormal bases.

CTX†: For fair comparison, we do not include SimCLR training episodes as in the original implementation [9], as this can also be employed by our other baselines and by FRN itself. Instead, we re-use the auxiliary loss adapted from DSN.

9.4. Image Decoder

The decoder network for the visualizations in Section 5.2 takes the form of an inverted ResNet-12, with four residual blocks and a final projection layer to three channels. Upsampling is performed using strided transposed convolution in both the residual stem and the main stem of each residual block. These transposed convolution layers also downsize the number of channels. The final residual block outputs of size 80×80 are rescaled to the original 84×84 resolution using bilinear sampling before the final projection layer. The full architecture is described in Table 13.

The decoder network is trained through L-1 reconstruction loss using Adam with an initial learning rate of 0.01 and batch size of 200. The CUB decoder is trained for 500 epochs, with learning rate decreasing by a factor of 4 every 100 epochs. mini-ImageNet is a larger dataset than CUB, so the corresponding decoder is trained for 200 epochs, with learning rate decreasing every 40 epochs.

Over the course of training, we found that the FRN classifier learns to regularize the reconstruction problem heavily. This is problematic in that the regularized reconstructions all fall off the input manifold for the decoder network, producing uniformly flat grey-brown images. To prevent this, we removed the normalization on the classifier latent embeddings (multiplying all

Model	Implementation	Phase	Episode	Time per Episode (ms)
DeepEMD [43]	qpth [2]	meta-train	5-way 1-shot	26,000
FRN (ours)	Eq. 10	meta-train	5-way 1-shot	**281**
FRN (ours)	Eq. 8	meta-train	5-way 1-shot	**279**
DeepEMD [43]	qpth [2]	meta-train	5-way 5-shot	>990,000
FRN (ours)	Eq. 10	meta-train	5-way 5-shot	**357**
FRN (ours)	Eq. 8	meta-train	5-way 5-shot	**346**
DeepEMD [43]	qpth [2]	meta-test	5-way 1-shot	23,275
DeepEMD [43]	OpenCV [5]	meta-test	5-way 1-shot	178
FRN (ours)	Eq. 10	meta-test	5-way 1-shot	**73**
FRN (ours)	Eq. 8	meta-test	5-way 1-shot	**63**
DeepEMD [43]	qpth [2]	meta-test	5-way 5-shot	>800,000
DeepEMD [43]	OpenCV [5]	meta-test	5-way 5-shot	18,292
FRN (ours)	Eq. 10	meta-test	5-way 5-shot	**88**
FRN (ours)	Eq. 8	meta-test	5-way 5-shot	**79**

Table 14. Speed comparison between DeepEMD and FRN on mini-ImageNet with ResNet-12 as the backbone. FRN outperforms DeepEMD with a large margin for different shot numbers under both meta-training and meta-testing phases.

inputs by $\sqrt{640}$). This was sufficient to eliminate the negative impact of regularization and produce meaningful image reconstructions.

10. Computational Efficiency

Speed comparison to DeepEMD: As shown in Table 14, we compare the computation speed between FRN and DeepEMD [43] with different episode shot numbers and implementation variants. Both use ResNet-12 as backbones and are evaluated on mini-ImageNet. For each episode, each class contains 15 query images during meta-training, and 16 query images during meta-testing. FRN is much more efficient in both training and evaluation scenarios.

For DeepEMD, the OpenCV [5] implementation of the EMD solver uses a modified Simplex algorithm to calculate the transport matrix, while qpth [2] uses an interior point method. It should be noted that while the OpenCV solver is faster, its gradients cannot be accessed. Thus it should not be used for meta-training (while it is possible, the calculated transport matrix must be treated like a constant and so gradient quality – and downstream performance – suffers). Note that for meta-testing episodes with shot number above one, DeepEMD learns a structured fully-connected layer via SGD. This leads to a significant slowdown compared to the 1-shot case – both because multiple forward and backward passes are required, and because once again OpenCV cannot provide gradients and thus should not be used.

For the sake of completeness, we provide speed results for both options of FRN's implementation (Eq. 8 and Eq. 10). For ResNet-12, $d = 640$, $k = 1$ or 5, and $r = 5 \times 5 = 25$, thus $d > kr$. From the analysis in Sec. 3.4, Eq. 8 should then be more efficient than Eq. 10, and this can be verified in Table 14. All other experiments use Eq. 10 as FRN's implementation, however, only for the sake of consistency.

Memory comparisons to CTX†: As shown in Table 15, we compare the GPU memory usage between FRN and CTX† during training on CUB, following the setup described in Sec. 9.1. We use Conv-4 models and upscale the feature map resolution to 10×10 to investigate how performance changes as kr increases. Memory is reported on a single GPU using the `nvidia-smi` resource tracker. While differences in the 5×5 setting are mostly insignificant, the 10×10 setting shows a rapid blowup in memory usage for CTX† that FRN is able to mitigate. We include both formulations of FRN for completeness. Interestingly, the analysis in Sec. 3.4 breaks down in this setting: contrary to expectations, 1-shot memory usage is the same in both FRN variants, and Eq. 8 is more efficient than Eq. 10 in the 5-shot 5×5 setting, despite the fact that it is inverting a larger matrix (125×125 vs 64×64). This could be due to particular implementations of matrix inversion, multiplication, and axis permutation on the GPU. However, we need not choose one variant over the other *a priori*. As both variants are algebraically equivalent, they can be substituted on the fly.

11. Additional Results on CUB

11.1. Class Split

Unlike mini-ImageNet, for which researchers tend to use the same class split as [25], CUB doesn't have an official split. Although practitioners agree on the train/validation/test split ratio (i.e., 100/50/50), there exist many different specific class split implementations [26, 35, 7, 42, 33]. For our CUB experiments in the main paper, we use the same random

	5×5		10×10	
Model	1 shot	5 shot	1 shot	5 shot
CTX†	6225	5449	8979	9693
FRN (Eq. 8)	6305	5317	7125	6767
FRN (Eq. 10)	6305	6529	7125	6727

Table 15. Memory usage in megabytes for training CTX† and FRN. While differences are small in the 5×5 feature map resolution setting, increasing the resolution to 10×10 is sufficient to produce large differences in memory usage. Note that these numbers reflect convolutional network layers as well as the final reconstruction layer.

Model	**class split**	**Backbone**	**1-shot**	**5-shot**
FRN (ours)	Tang *et al*. [33]	ResNet-12	**83.55±0.19**	**92.92±0.10**
FRN (ours)	Chen *et al*. [7]	ResNet-12	**83.35±0.18**	**93.28±0.09**
Baseline++♭ [7]	Chen *et al*. [7]	ResNet-34	68.00±0.83	84.50±0.51
ProtoNet [7, 30]	Chen *et al*. [7]	ResNet-34	72.94±0.91	87.86±0.47
S2M2♭ [22]	Chen *et al*. [7]	WRN-28-10	80.68±0.81	90.85±0.44
Neg-Cosine♭ [19]	Chen *et al*. [7]	ResNet-18	72.66±0.85	89.40±0.43

Table 16. Performance comparison for FRN under two different class split settings. We include some competitive baselines using the same split as Chen *et al*. [7] for reference. Results of FRN are averaged over 10,000 trials with 95% confidence interval. ♭ denotes the use of non-episodic pre-training.

training setting	**1-shot**	**5-shot**
from scratch	**83.55±0.19**	92.92±0.10
after pre-train	73.46±0.20	84.20±0.13
finetune	83.23±0.18	**93.59±0.09**

Table 17. Impact of pre-training for FRN on CUB using raw images as input. Pre-training can slightly boost the performance compared to training from scratch.

train/validation/test split as Tang *et al*. [33]. Many recent open-sourced works [44, 19, 22] use the dataset-generating code[15] from Chen *et al*. [7], using raw, non-cropped images as input. Therefore, we re-run our method on CUB under the uncropped setting using the same class split as Chen *et al*. [7]. As shown in Table 16, FRN performance is not impacted by the choice of class split.

11.2. Pre-Training

We introduce the pre-training technique for FRN in Sec. 3.6 and apply it to mini-ImageNet and tiered-ImageNet in Table 7. Here, we apply it to FRN with ResNet-12 backbone for un-cropped CUB, as in Sec. 4.1 and Table 4.

For non-episodic FRN pre-training, we run 1200 epochs using SGD with initial learning rate 0.1 and Nesterov momentum 0.9. We cut the learning rate by a factor of 10 at epochs 600 and 900. Subsequent episodic fine-tuning uses the same optimizer for 600 epochs, but with initial learning rate 1e-3, decreased by a factor of 10 at epochs 300 and 500. For the 5-shot FRN model, we use the standard 20-way episodes. The 1-shot model uses 25-way episodes.

As shown in Table 17, pre-training still improves the final accuracy, but the gain compared to training from scratch becomes much smaller than for mini-ImageNet (see Table 9). This is reasonable, as CUB is only about $\frac{1}{6}$ the size of mini-ImageNet. It is thus comparatively easier in this setting for the FRN trained episodically from scratch to find a good optimum, and more difficult for pre-training to improve.

12. Additional Visualizations

Additional image reconstruction trials as in Fig. 3 of the main paper begin on the following page.

[15] https://github.com/wyharveychen/CloserLookFewShot/blob/master/filelists/CUB/write_CUB_filelist.py

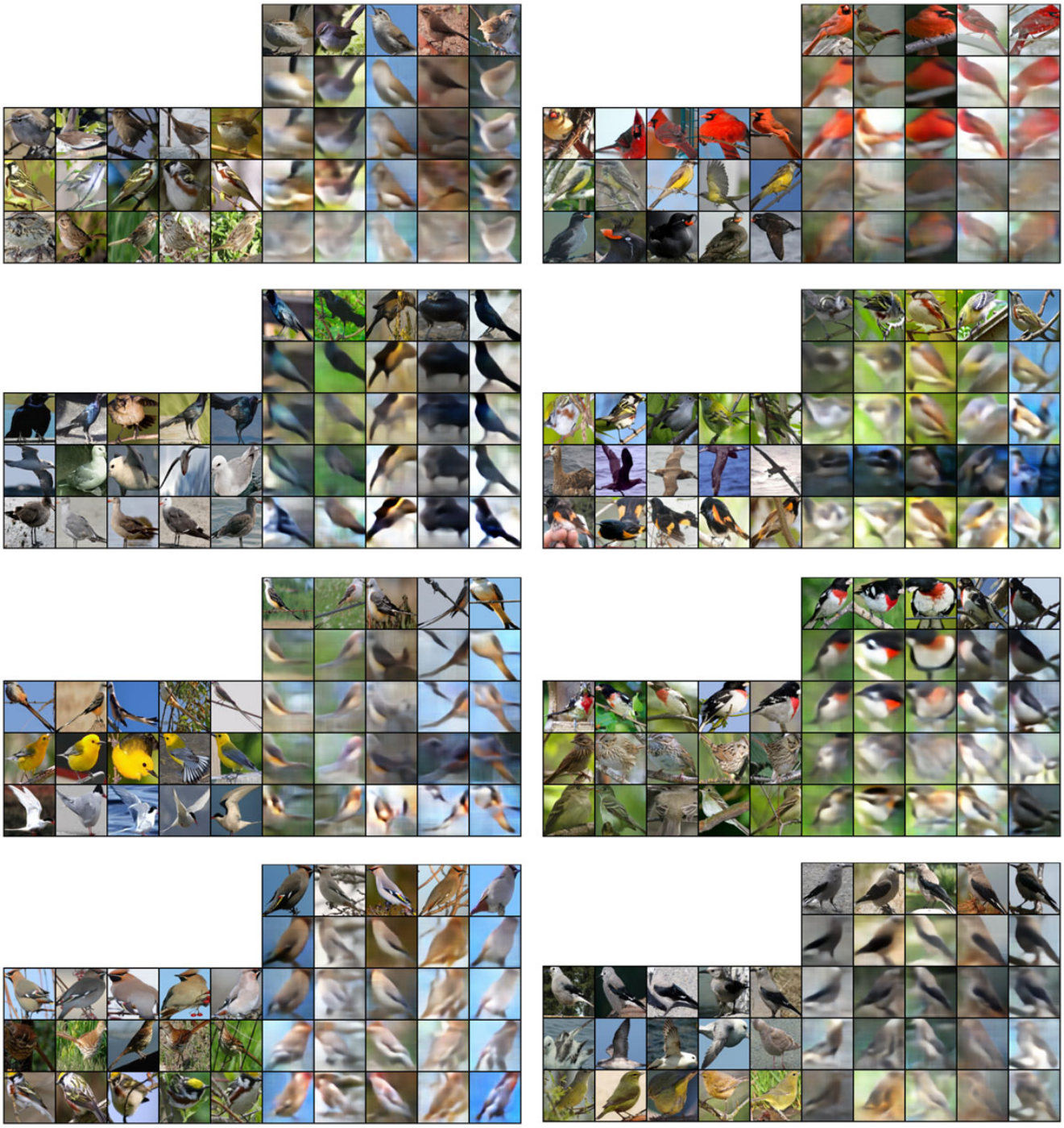

Figure 4. Additional image reconstruction visualizations for CUB. Formatting follows Fig. 3: support images are given on the left while target images and reconstructions are on the right. First row images are targets, second row images are autoencoded, third row images are reconstructed from the same class, and fourth and fifth row images are reconstructed from different classes. Same-class reconstructions are clearly superior to those from different classes.

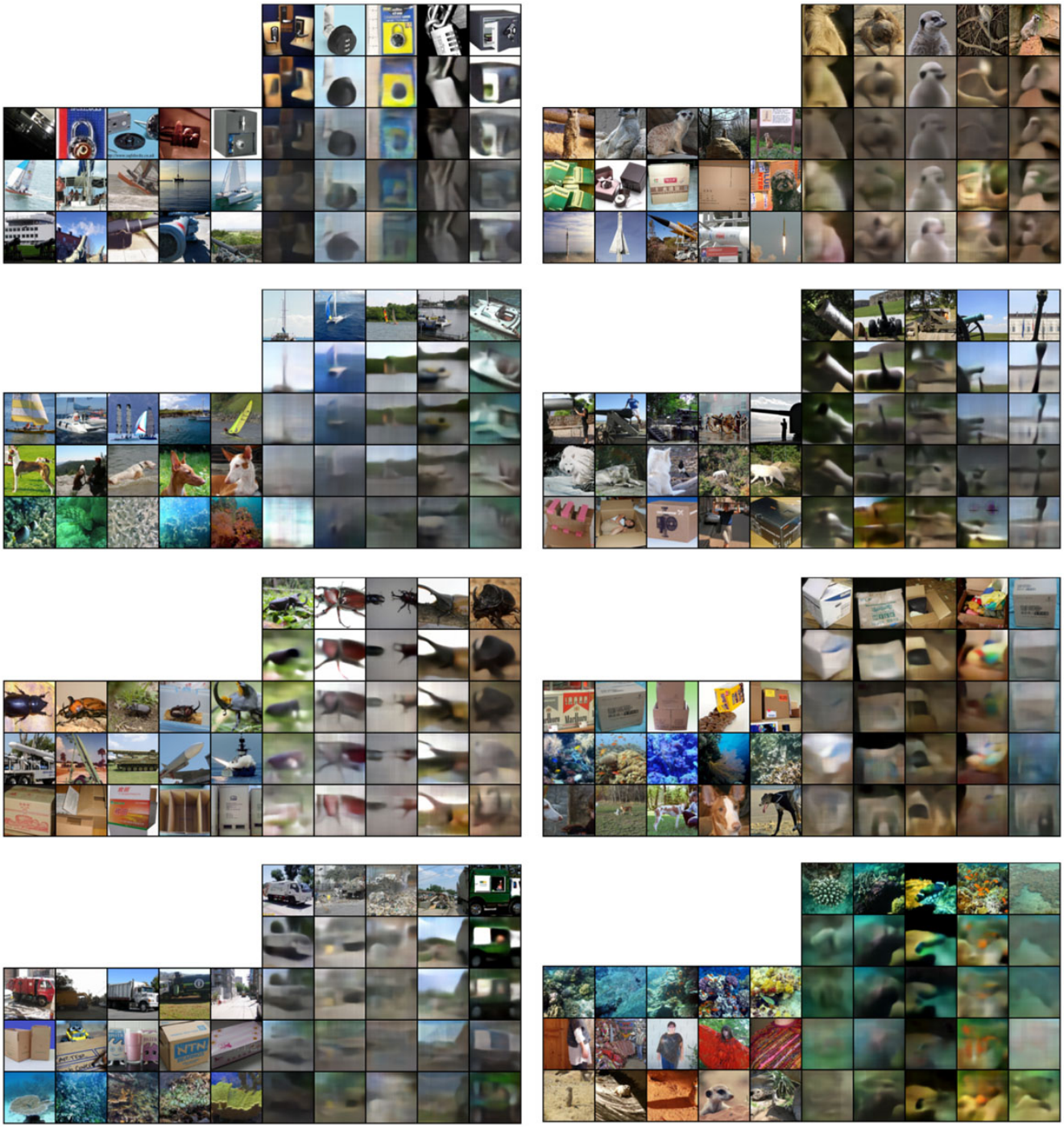

Figure 5. Additional image reconstruction visualizations for mini-ImageNet. Formatting follows Fig. 3: support images are given on the left while target images and reconstructions are on the right. First row images are targets, second row images are autoencoded, third row images are reconstructed from the same class, and fourth and fifth row images are reconstructed from different classes. Same-class reconstructions tend to gray out or darken the colors, but are much more faithful shape-wise than those from different classes.