Supplementary materials for Using Shape to Categorize: Low-Shot Learning with an Explicit Shape Bias—Appendix

Stefan Stojanov, Anh Thai, James M. Rehg
Georgia Institute of Technology
{sstojanov, athai6, rehg}@gatech.edu

This supplementary material document is structured as follows: In Section 1 we provide further detail about the training data used in the paper; In section 3 we provide details on the baselines used in the paper, their implementation details and the hyperparameters used for training; In Section 4 we provide empirical evidence about our choice of point-cloud encoding architecture; In Section 5 we provide further details about the training procedure of the shape-biased image embeddings used in the paper.

1. Further Dataset Details

In this section we provide details on the composition of the datasets used in the main paper. We provide example images used to illustrate the data used for training in Figure 1. As a result of using a ray tracing-based renderer Cycles [11], the synthetic image data used for training has high realism. For all algorithms we use 224×224 RGB images as input. For point cloud-based learning we use the 3D (x, y, z) coordinates 1024 randomly sampled points as input. For images we use standard geometric data augmentations e.g. flipping, cropping, slight translation and rotation, as well as color jittering, since we found these result in improved validation performance. For point clouds we use the same augmentation procedures as in [18], which include translation, jittering and dropout.

1.1. Toys4K

We provide further details on the composition of our new Toys4K dataset in Table 1. The 40 train, 10 validation, and 55 test classes split is shown in Table 5. When performing validation and testing on Toys4K, we generate low-shot episodes consisting of up to 5 shots and 10 queries.

1.2. ModelNet40-LS

The 20 train, 10 validation, 10 test classes split for ModelNet40-LS is shown in Table 4. When performing validation and testing on ModelNet40-LS, we generate low-shot episodes consisting of up to 5 shots and 15 queries.

1.3. ShapeNet55-LS

The 25 train, 10 validation, 20 test classes split for ShapeNet55-LS is shown in Table 3. When performing validation and testing on ShapeNet55-LS, we generate low-shot episodes consisting of up to 5 shots and 15 queries.

2. Further Low-Shot Analysis

In this section we provide further analysis of the low-shot performance by presenting confusion matrices and classification performance in individual low-shot episodes.

2.1. Confusion Matrices

Please refer to Figure 2 and Figure 3 for low-shot confusion matrices on ModelNet40-LS and ShapeNet55-LS. The confusion matrices are obtained by evaluation 5K low-shot episodes for each dataset (10-way for ModelNet40-LS and 20-way for ShapeNet55-LS), and counting how each sample was classified. The confusion matrices reflect the results presented in Section 4 in the main text that adding shape bias improves overall low-shot classification performance.

2.2. Per-episode Analysis

We provide a per-episode analysis of low-shot classification in Figure 4 to show qualitative evidence of low-shot learning with shape bias. We see that there are cases in which even though there are no view ambiguities, the image-only model misclassifies whereas the shape-biased model correctly classifies (e.g. in the lower left episode, confusing bicycle for sheep).

3. Baseline Algorithm Details

All algorithms in this paper are implemented using PyTorch [9]. In this section we provide further detail about the baseline implementations and hyperparameters used for training.

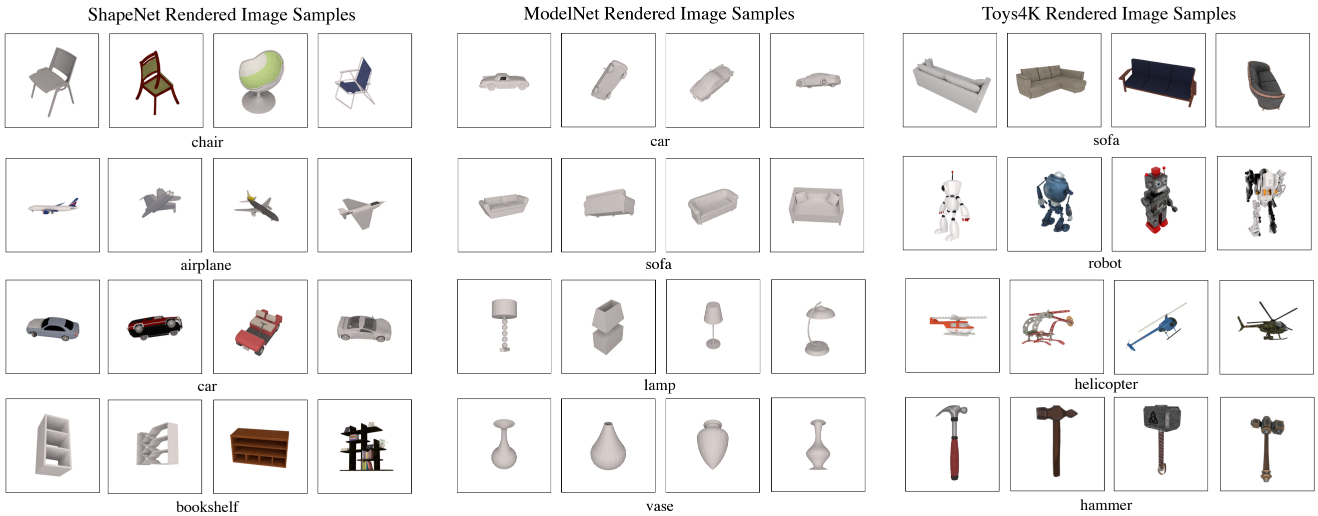

Figure 1. Rendered image samples from multiple categories of ShapeNet, ModelNet and Toys4K. Note the high image quality as a result of using ray-tracing based rendering.

chair	210	tree	57	knife	45	piano	39	shark	30	panda	24	submarine	18
bottle	111	candy	56	trashcan	44	boat	38	stove	29	orange	24	helmet	17
robot	105	guitar	55	ball	44	bread	38	bowl	28	mushroom	23	bicycle	16
dog	103	apple	54	frog	43	fish	37	car	28	phone	23	lion	16
mug	97	flower	54	ice cream	43	horse	36	cookie	28	train	22	motorcycle	16
hammer	94	ladder	53	dragon	43	spade	36	cupcake	28	tv	21	hamburger	16
cat	79	penguin	51	pan	42	banana	35	bunny	27	toaster	21	grapes	16
dinosaur	76	keyboard	51	battery cell	41	airplane	35	drum	26	helicopter	20	tractor	16
deer/moose	65	pencil	50	whale	41	donut	34	pizza	26	lizard	20	monkey	16
fox	64	plate	50	shoe	40	truck	34	mouse	25	saw	19	pc mouse	15
hat	64	key	49	laptop	40	coin	33	chicken	25	marker	19	light bulb	15
sofa	63	chess piece	49	pig	40	snake	32	sink	25	microwave	18	closet	15
glass	63	cake	48	sheep	39	fridge	32	cow	25	bus	18	fries	15
cup	60	screwdriver	46	crab	38	octopus	31	dolphin	25	pear	18	sandwich	15
monitor	57	elephant	46	radio	38	fan	31	violin	25	butterfly	18	giraffe	15

Table 1. The category composition of the Toys4K dataset.

3.1. SimpleShot

The implementation in our codebase for SimpleShot [16] is based on the code release by the authors in [1]. The authors report a 1-shot 5-way accuracy of $49.69(0.19)$ and a 5-shot 5-way accuracy of $66.92(0.17)$ on miniImageNet [15] with the Conv4 architecture. The reimplementation of SimpleShot in our codebase with the same dataset and architecture results in 1-shot 5-way accuracy of $50.60(0.34)$ and a 5-shot 5-way accuracy of $68.06(0.23)$.

In all our experiments we train SimpleShot with SGD with an initial learning rate of 0.01 and a learning rate decay of 0.1 at epochs 300 and 360, out of a total of 400 epochs. SimpleShot employs three different feature normalization strategies, no normalization, L_2 normalization and L_2 normalization and training set mean subtraction. In experiments with SimpleShot we report the result of the best of these three normalization strategies.

3.2. RFS

The implementation in our codebase for RFS [14] is based on the code release by the authors in [2]. The original codebase obtains a 1-shot 5-way accuracy of $53.73(0.81)$ on miniImageNet [15] with the Conv4 architecture. The reimplementation of RFS in our codebase with the same dataset and architecture results in 1-shot 5-way accuracy of $54.59(0.86)$. RFS requires training an embedding on the training dataset using cross-entropy. We train this embedding space with SGD using a learning rate of 0.001, momentum of 0.9 and L_2 weight penalty weight parameter of 0.0005. For each low-shot episode we train a logistic regression classifier using Scikit-learn[10], as in the original RFS.

3.3. FEAT

The implementation for FEAT is based on the code release by the authors in [3]. The original codebase obtains a 1-shot 5-way accuracy of $54.85(0.20)$ and 5-shot 5-way accuracy of 71.61 on miniImageNet [15] with the Conv4 architecture. The reimplementation of FEAT in our codebase with the same dataset and architecture results in 1-shot 5-way accuracy of $54.85(0.20)$ 5-shot 5-way accuracy of $71.45(0.73)$. We train FEAT with the default hyperparameters recommended by the authors, training separate models for 5-way and 10-way classification, and separate models for 1-shot and 5-shot, as recommended by the authors. For the shape biased FEAT we do not use learning rate scheduling and momentum, since they have a negative effect on performance for shape-biased training. Removing them for image-only training does not affect performance.

3.4. Protoypical Networks

The implementation in our codebase for Prototypical Networks is based on the code release by the SimpleShot authors in [1]. In [6] the authors report that their reimplementation obtains a 5-shot 5-way accuracy of $66.68(0.68)$ on miniImageNet [15] with the Conv4 architecture. The reimplementation of Prototypical Networks in our codebase with the same dataset and architecture results in 5-shot 5-way accuracy of $66.94(0.71)$. We train separate Prototypical Networks models for 5-shot classification and 1-shot classification. As recommended by the authors of the original paper, we perform 20-way training. We use the Adam [8] optimizer, 400 low-shot iterations per epoch, 200 epochs total, and a learning rate of 0.0001 L_2 and weight penalty weight parameter of 0.00001. We perform a learning rate decay of 0.5 every 20 epochs.

3.5. Triplet Model

We implement a joint triplet model using both point cloud (DGCNN [17] and image (ResNet18 [7]) encoders, which can use both image and shape information during training. Let f_i denote the image encoder, f_p denote the point cloud encoder, and ϕ_p^k and ϕ_i^k denote the point cloud and image encodings of object instance k respectively. We learn a joint image/shape embedding by minimizing a standard triplet loss

$$\mathcal{L}(\phi_i^k, \phi_p^k, \phi_p^l) = \max\left\{\mathbf{d}(\phi_i^k, \phi_p^k) - \mathbf{d}(\phi_p^k, \phi_p^l) + \text{margin}, 0\right\}$$
$$\mathbf{d}(x, y) = ||x - y||_2$$

where the anchor is an image embedding of instance k, ϕ_i^k, the positive sample is a point cloud encoding of the same object instance, ϕ_i^k, and the negative sample is a ϕ_p^l is a point cloud embedding of a different object instance l. We perform L_2 normalization of the embeddings prior to computing the loss. Note that it is possible to build (anchor, positive, negative) pairs using category information, but we empirically found that this leads to worse performance.

We train the triplet model using the Adam [8] optimizer with a learning rate of 0.0001, L_2 weight penalty weight parameter of 0.0001, and margin of 0.1. We use a batch size of 72 and train for 600 epochs, each epoch consisting of 20K random samples.

4. Learning a Point Cloud Shape Embedding

In this section we describe the algorithm for learning a point-cloud based embedding space, and present an empirical study for our point cloud architecture choice.

4.1. Algorithm

The algorithm we use to train a point-cloud embedding space is based on SimpleShot [16] and is described with pseudocode in Algorithm 1. Note that the routine AccAccumulator is used denotes a function to collect the validation accuracy of each low-shot episode and compute summary statistics. The NNCLASSIFY routine takes support features and labels, and classifies each test query feature based on a nearest neighbors rule using cosine similarity. The point-cloud embedding model is trained using SGD with a learning rate of 0.01, batch size of 129, and L_2 weight penalty weight parameter of 0.0001. We perform learning rate decay by 0.1 at epochs 300 and 360. In all models we use features from the output of the pooling layer in the architecture.

4.2. Architecture Study

We perform an empirical study on the point cloud architectures to determine which is capable of the best low-shot generalization performance. Our PointNet [12] implementation is based on [4], our PointNet++ [13] is based on [18] and our DGCNN [17] implementation is based on [5]. We use a DGCNN architecture with a reduced embedding dimension (size after the pooling operation) of 512 rather than the original 1024, to match the dimensionality of the ResNet18 embeddings. We find no decrease in performance by this reduction. We present the results of this study on ModelNet in Table 2. The DGCNN [17] architecture outperforms other point cloud architectures at low-shot generalization to novel categories. We find that randomly rotating the input point cloud about the origin during training (random rotation about all axes of rotation, indicated by SO3 in the table) results in a performance improvement. We use this SO3 strategy for all shape-embedding space learning experiments.

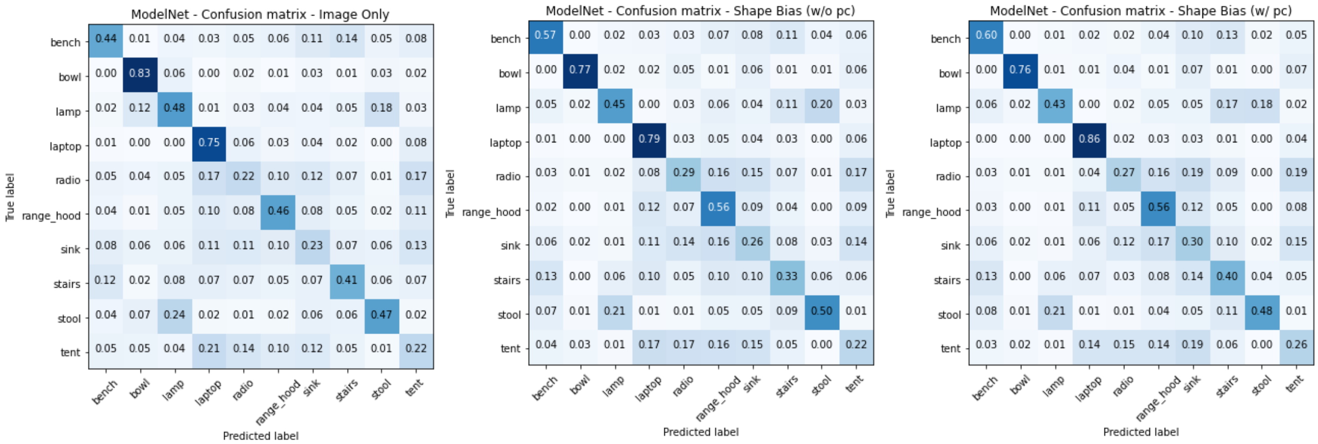

Figure 2. Confusion matrices over 5K low-shot episodes of SimpleShot for Image Only, Shape-Biased without access to point clouds (w/o pc) at test time and Shape-Biased with (w/ pc) access to point clouds at test time on the ModelNet-LS dataset. Even without access to point clouds (w/o pc) for building class prototypes, the shape-biased image embedding leads to improvements. Adding point cloud support information (w/ pc) improves performance further. See Table 3 in the main text for aggregate results.

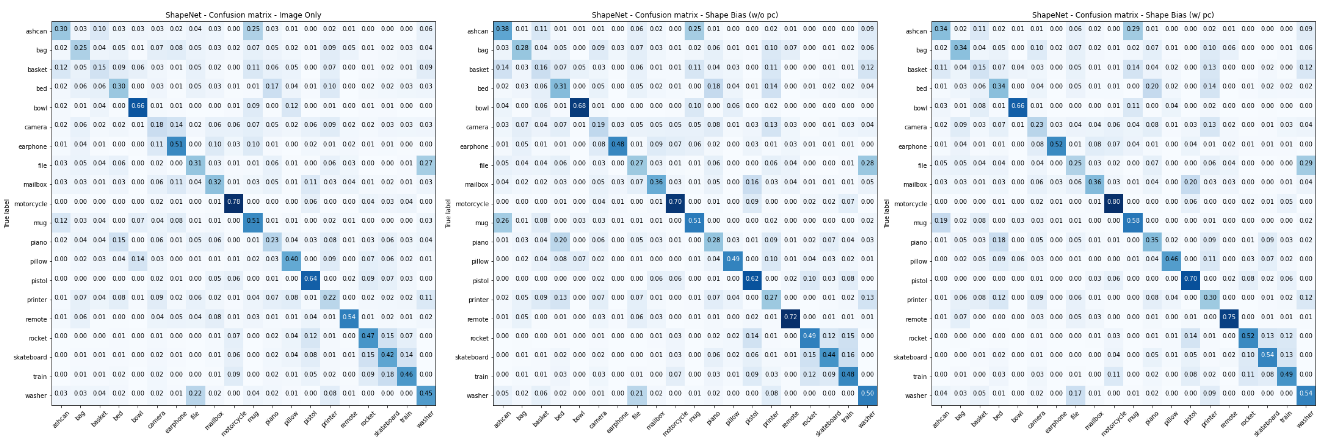

Figure 3. Confusion matrices over 5K low-shot episodes of SimpleShot for Image Only, Shape-Biased without access to point clouds (w/o pc) at test time and Shape-Biased with (w/ pc) access to point clouds at test time on the ShapeNet-LS dataset. As in ModelNet40-LS, without access to point clouds (w/o pc) for building class prototypes, the shape-biased image embedding leads to improvements. Adding point cloud support information (w/ pc) improves performance further. See Table 5 in the main text for aggregate results. Best viewed with zoom.

Architecture	1-shot 5-way accuracy
PointNet [12]	66.13
PointNet++ [13]	67.49
DGCNN [17]	75.2
DGCNN (SO3)	77.5

Table 2. Empirical study for choosing the best point cloud architecture. Reported is 1-shot 5-way classification accuracy on the ModelNet40-LS validation set. We find that DGCNN performs the best, and that randomly rotating each input point cloud during training (indicated with SO3) results in a improvement in low-shot generalization performance as well.

5. Details for Learning a Shape Biased Image Embedding

The algorithm we use to train a shape-biased image embedding is described with pseudocode in Algorithm 2. We use the Adam optimizer with a batch size of 256, an initial learning rate of 0.001 and a L_2 weight penalty weight parameter of 0.0001. The model is trained for 400 epochs, with a learning rate decay of 0.1 at epochs 300 and 360.

5.1. SimpleShot with Shape Bias

The SimpleShot [16] approach does not require any learning (parameter updates) during the low-shot phase.

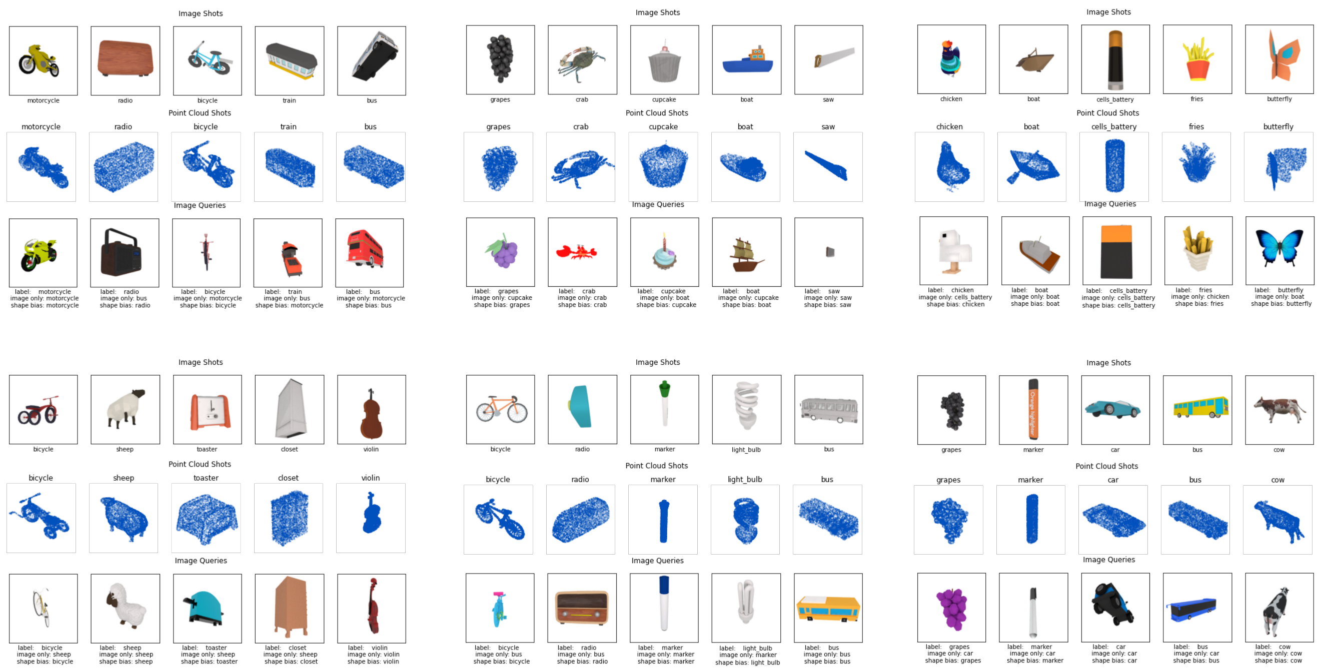

Figure 4. Six low-shot episodes for 5 ways, 1 shot and 1 query on Toys4K for shape-biased SimpleShot. We visually display the composition of the image and point cloud shots and the image queries, as well as the models' predictions, illustrating cases where shape bias allows for improved performance. Best viewed with zoom.

Classification is done using nearest centroid classification in the embedding space. The image embedding function f_i is trained as described in Algorithm 2, and low-shot testing is done following the same procedure as described in L8-16 in Algorithm 1 but using nearest centroid rather than nearest neighbor classification.

5.2. FEAT with Shape Bias

The algorithm we use to train a shape-biased FEAT [19] architecture is described in Algorithm 3. Note that the f_i used in this algorithm is being fine tuned from a mapping already trained with Algorithm 2 while the FEAT set-to-set function $\mathbf{E}$ is trained from scratch. For this experiment we use the default hyperparameters recommended by the FEAT authors. Low shot testing is done following the same procedure as described in L13-22 in Algorithm 3 but using the test set. The procedure we refer to as FEATClassify is described in Eq. 4 on pg. 4 of the FEAT paper [19]. In the pseudocode FEATClassify performs classification and directly outputs the per-episode classification accuracy.

Algorithm 1: Training Shape Embedding f_p

Input: Randomly initialized point-cloud classifier architecture f_p with embedding function f_p^E
Total number of epochs N_e
Total number of mini-batches per epoch N_b
Total number of low-shot iterations for validation N_{it}

Data: (point cloud, label) pair datasets $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{val}}$

Define: ℓ : cross-entropy loss

foreach epoch in $1, 2, \ldots, N_e$ **do**
  foreach mini-batch $(\boldsymbol{o}_p, \boldsymbol{y}) \sim \mathcal{D}^{\text{train}}$ of N_b **do**
    Predict $\hat{\boldsymbol{y}} = f_p(\boldsymbol{o}_p)$
    Compute $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$
    Compute $\nabla\ell$ with respect to f_p
    Update f_p with SGD
  end
  A = ACCACCUMULATOR
  foreach validation episode in $1, 2, \ldots, N_e$ **do**
    Sample 5-way 1-shot $(\boldsymbol{o}_p^{\text{train}}, \boldsymbol{y}^{\text{train}}, \boldsymbol{o}_p^{\text{test}}) \sim \mathcal{D}^{\text{val}}$
    Predict $\boldsymbol{\phi}_p^{\text{train}} = f_p^E(\boldsymbol{o}_p^{\text{train}})$
    Predict $\boldsymbol{\phi}_p^{\text{test}} = f_p^E(\boldsymbol{o}_p^{\text{test}})$
    acc = NNCLASSIFY($\boldsymbol{\phi}_p^{\text{train}}, \boldsymbol{y}^{\text{train}}, \boldsymbol{\phi}_p^{\text{test}}$)
    A(acc)
  end
  val accuracy = A.average()
  if val accuracy $>$ best accuracy **then**
    best acuracy $\leftarrow$ val accuracy
    $f_p^{\text{best}} \leftarrow f_p$
  end
end

Result: Trained f_p^{best}

Algorithm 2: Training Shape-Biased Image Embedding Function f_i

Input: Randomly initialized image embedding architecture f_i
Point-cloud embedding function f_p (1)
Total number of epochs N_e
Total number of mini-batches per epoch N_b
Total number of low-shot iterations for validation N_{it}

Data: (image, point cloud, label) pair datasets $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{val}}$

Define: $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$ (see main text for def.)

foreach epoch in $1, 2, \ldots, N_e$ **do**
  foreach mini-batch $(\boldsymbol{o}_i, \boldsymbol{o}_p, \boldsymbol{y}) \sim \mathcal{D}^{\text{train}}$ of N_b **do**
    Predict shape embedding $\boldsymbol{\phi}_p = f_p(\boldsymbol{o}_p)$
    Predict image embedding $\boldsymbol{\phi}_i = f_i(\boldsymbol{o}_i)$
    Compute $\mathcal{L}$ using $\boldsymbol{\phi}_p$ and $\boldsymbol{\phi}_i$
    Compute $\nabla\mathcal{L}$ with respect to f_i
    Update f_i with Adam
  end
  A = ACCACCUMULATOR
  foreach validation episode in $1, 2, \ldots, N_e$ **do**
    Sample 5-way 1-shot $(\boldsymbol{o}_i^{\text{train}}, \boldsymbol{o}_p^{\text{train}}, \boldsymbol{y}^{\text{train}}, \boldsymbol{o}_i^{\text{test}}) \sim \mathcal{D}^{\text{val}}$
    Predict $\boldsymbol{\phi}_p^{\text{train}} = f_p(\boldsymbol{o}_p^{\text{train}})$
    Predict $\boldsymbol{\phi}_i^{\text{train}} = f_i(\boldsymbol{o}_i^{\text{train}})$
    Predict $\boldsymbol{\phi}_i^{\text{test}} = f_i(\boldsymbol{o}_i^{\text{test}})$
    $\boldsymbol{\phi}^{\text{train}} \leftarrow$ AVERAGE($\boldsymbol{\phi}_p^{\text{train}}, \boldsymbol{\phi}_i^{\text{train}}$)
    acc = NNCLASSIFY($\boldsymbol{\phi}^{\text{train}}, \boldsymbol{y}^{\text{train}}, \boldsymbol{\phi}_i^{\text{test}}$)
    A(acc)
  end
  val accuracy = A.average()
  if val accuracy $>$ best accuracy **then**
    best acuracy $\leftarrow$ val accuracy
    $f_i^{\text{best}} \leftarrow f_i$
  end
end

Result: Trained f_i^{best}

Algorithm 3: Training FEAT with Shape Bias

Input: Shape-biased image encoder f_i (2)
Point-cloud embedding function f_p (1)
Randomly initialized FEAT [19] set-to-set function $\mathbf{E}$–see p3 in [19].
Total number of epochs N_e
Total number of low-shot iterations per training epoch N_{it}
Total number of low-shot iterations for validation N_{v-it}

Data: (image, point cloud, label) pair datasets $\mathcal{D}^{\text{train}}$, $\mathcal{D}^{\text{val}}$

Define: $\mathcal{L}_{\text{FEAT}}$ – Eq. 7 in [19]

foreach epoch in $1, 2, \ldots, N_e$ **do**
  foreach training episode in of $1, 2, \ldots, N_{it}$ **do**
    Sample m-way n-shot $(\boldsymbol{o}_i^{\text{train}}, \boldsymbol{o}_p^{\text{train}}, \boldsymbol{o}_p^{\text{query}}, \boldsymbol{y}^{\text{train}}, \boldsymbol{y}^{\text{query}}) \sim \mathcal{D}^{\text{train}}$
    Predict ptcld. support $\boldsymbol{\phi}_p^{\text{train}} = f_p(\boldsymbol{o}_p^{\text{train}})$
    Predict image support $\boldsymbol{\phi}_i^{\text{train}} = f_i(\boldsymbol{o}_i^{\text{train}})$
    Predict image queries $\boldsymbol{\phi}_i^{\text{query}} = f_i(\boldsymbol{o}_i^{\text{query}})$
    $\boldsymbol{\phi}^{\text{train}} \leftarrow \textsc{Average}(\boldsymbol{\phi}_p^{\text{train}}, \boldsymbol{\phi}_i^{\text{train}})$
    $\hat{\boldsymbol{\phi}}^{\text{train}}, \hat{\boldsymbol{\phi}}_i^{\text{query}} \leftarrow \mathbf{E}(\boldsymbol{\phi}^{\text{train}}, \boldsymbol{\phi}_i^{\text{query}})$
    Compute $\mathcal{L}$ using $\hat{\boldsymbol{\phi}}^{\text{train}}, \hat{\boldsymbol{\phi}}_i^{\text{query}}$ and $\boldsymbol{y}^{\text{train}}, \boldsymbol{y}^{\text{query}}$
    Compute $\nabla\mathcal{L}$ with respect to f_i and $\mathbf{E}$
    Update $f_i, \mathbf{E}$ with SGD
  end
  A = AccAccumulator
  foreach validation episode in $1, 2, \ldots, N_{v-it}$ **do**
    Sample 5-way 1-shot $(\boldsymbol{o}_i^{\text{train}}, \boldsymbol{o}_p^{\text{train}}, \boldsymbol{y}^{\text{train}}, \boldsymbol{o}_i^{\text{test}}) \sim \mathcal{D}^{\text{val}}$
    Predict ptcld. support $\boldsymbol{\phi}_p^{\text{train}} = f_p(\mathbf{o}_p^{\text{train}})$
    Predict image support $\boldsymbol{\phi}_i^{\text{train}} = f_i(\mathbf{o}_i^{\text{train}})$
    Predict image queries $\boldsymbol{\phi}_i^{\text{test}} = f_i(\mathbf{o}_i^{\text{test}})$
    $\boldsymbol{\phi}^{\text{train}} \leftarrow \textsc{Average}(\boldsymbol{\phi}_p^{\text{train}}, \boldsymbol{\phi}_i^{\text{train}})$
    acc = FEATClassify($\boldsymbol{\phi}^{\text{train}}, \boldsymbol{y}^{\text{train}}, \boldsymbol{\phi}_i^{\text{test}}$)
    A(acc)
  end
  val accuracy = A.average()
  if val accuracy $>$ best accuracy **then**
    best acuracy $\leftarrow$ val accuracy
    $f_i^{\text{best}} \leftarrow f_i$
    $\mathbf{E}^{\text{best}} \leftarrow \mathbf{E}$
  end
end

Result: Trained $f_i^{\text{best}}, \mathbf{E}^{\text{best}}$

Training	# samples	Validation	# samples	Testing	# samples
vessel	873	train	389	mug	214
car	530	bed	233	tower	133
sofa	500	stove	218	motorcycle	337
lamp	500	bowl	186	cap	56
cellular	500	pillow	96	pistol	307
faucet	500	mailbox	94	earphone	73
pot	500	rocket	85	skateboard	152
guitar	500	birdhouse	73	camera	113
airplane	500	microphone	67	piano	239
bus	500	keyboard	65	printer	166
chair	500			bag	83
rifle	500			trashcan	343
cabinet	500			file	298
bench	499			dishwasher	93
bathtub	499			microwave	152
telephone	499			washer	169
jar	499			remote	66
bottle	498			helmet	162
display	496			basket	113
clock	496			can	108
loudspeaker	496				
table	495				
laptop	460				
bookshelf	452				
knife	423				
Total					
25 classes	12716	10 classes	1506	20 classes	3377

Table 3. Split composition of ShapeNet55-LS

Training	# samples	Validation	# samples	Testing	# samples
bed	615	cup	99	range hood	215
car	297	xbox	123	bowl	84
guitar	255	bathtub	156	stool	110
bottle	435	cone	187	radio	124
desk	286	curtain	158	stairs	144
night stand	286	door	129	lamp	144
glass box	271	flower pot	169	tent	183
sofa	780	person	108	sink	148
piano	331	wardrobe	107	bench	193
toilet	444	keyboard	165	laptop	169
monitor	565				
table	492				
dresser	286				
airplane	726				
tv stand	367				
chair	989				
bookshelf	672				
vase	575				
plant	340				
mantel	384				
Total					
20 classes	9396	10 classes	1401	10 classes	1514

Table 4. Split composition of ModelNet40-LS

Training	# samples	Validation	# samples	Testing	# samples
candy	56	airplane	35	boat	38
flower	54	shark	30	lion	17
dragon	43	truck	34	whale	41
apple	54	phone	23	cupcake	28
guitar	55	giraffe	15	train	22
tree	57	horse	37	pizza	26
glass	63	fish	37	marker	19
cup	60	fan	31	cookie	28
pig	41	shoe	41	sandwich	15
cat	79	snake	32	octopus	31
chair	210			monkey	16
ice cream	43			fries	15
hat	64			violin	25
deer moose	65			mushroom	23
penguin	53			closet	15
ball	44			tractor	16
fox	64			submarine	18
dog	103			butterfly	18
knife	45			pear	18
laptop	41			bicycle	17
pen	42			dolphin	25
mug	97			bunny	27
plate	50			coin	33
chess piece	49			radio	40
cake	48			grapes	16
frog	43			banana	35
ladder	53			cow	25
keyboard	51			donut	34
sofa	63			stove	29
trashcan	44			sink	25
dinosaur	76			orange	24
bottle	111			saw	19
elephant	46			chicken	25
pencil	50			hamburger	16
key	49			piano	39
monitor	57			light bulb	15
hammer	94			spade	36
screwdriver	46			crab	40
robot	105			sheep	40
bread	38			toaster	21
				lizard	20
				motorcycle	16
				mouse	25
				pc mouse	15
				bus	18
				helicopter	20
				microwave	18
				cells battery	41
				drum	26
				panda	24
				tv	21
				car	28
				helmet	17
				fridge	31
				bowl	28
Total					
40 classes	2506	10 classes	315	55 classes	1358

Table 5. Split composition of Toys4K

Appendix References

[1] https://github.com/mileyan/simple_shot.

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

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

[4] https://github.com/yanx27/Pointnet_Pointnet2_pytorch.

[5] https://github.com/AnTao97/dgcnn.pytorch.

[6] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2018.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Blender Proejct. https://blender.org.

[12] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[13] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[14] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision (ECCV) 2020*, August 2020.

[15] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[16] Yan Wang, Wei-Lun Chao, Kilian Q Weinberger, and Laurens van der Maaten. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *arXiv preprint arXiv:1911.04623*, 2019.

[17] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[18] Erik Wijmans. Pointnet++ pytorch. https://github.com/erikwijmans/Pointnet2_PyTorch, 2018. 1

[19] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.