Descriptor and Word Soups 🍜: Overcoming the Parameter Efficiency Accuracy Tradeoff for Out-of-Distribution Few-shot Learning

Supplementary Material

Algorithm 1 Descriptor soup pseudo-code, PyTorch-like

```
# '@' means matrix multiplication in Python.
# '+' means concatenation when operating on lists.
# Inputs: L2-normalized image_embeddings, y_truth
# classnames: list of classnames in English
# model: CLIP-style model
# descriptions: list of descriptions from an LLM
# Hyperparameters: m (number of members in the soup)
descriptions = ['which has legs.', 'which can swim.', ... ]

def get_accuracy(image_embeddings, text_embeddings,
                 y_truth):
    scores = image_embeddings @ text_embeddings.T
    return (scores.argmax(dim=1) == y_truth).mean()

def get_description_embeddings(description):
    d = tokenizer(['a photo of ' + classname + ', ' +
        description for classname in classnames])
    return normalize(model.encode_text(d))

accuracies = []
for description in descriptions:
    text_embeddings = get_description_embeddings(
                          description)
    accuracies.append(get_accuracy(
        image_embeddings, text_embeddings, y_truth))
# sort descriptions by accuracies
descriptions_sorted = descriptions[
    accuracies.sort(descending=True).indices]

# initialize with best descriptor
soup, accuracy = [descriptions_sorted[0]], accuracies[0]

# greedy selection
for description in descriptions_sorted:
    soup_embeddings = stack(
        [get_description_embeddings(description)
            for description in soup + [description] ] )
    text_embeddings = normalize(
        soup_embeddings.mean(dim=0))
    if get_accuracy(image_embeddings,
        text_embeddings, y_truth) > current_acc:
        soup = soup + [description]

return soup[:m]
```

Algorithm 2 Word soup pseudo-code, PyTorch-like

```
# Hyperparameters: k0, k1, m, patience
# Inputs: L2-normalized image_embeddings, y_truth

words = ["the", "of", "and", ... ]
accuracies = []
for word in words:
    text_embeddings = get_description_embeddings(word)
    accuracies.append(get_accuracy(
        image_embeddings, text_embeddings, y_truth))
# sort descriptions by accuracies
words = words[accuracies.sort(descending=True).indices]

soup = []
for repeat m times:
    first_word = random.shuffled(words[0:k0])[0]
    word_chain = first_word
    accuracy = get_accuracy(image_embeddings,
        get_description_embeddings(word_chain), y_truth)
    words_k1 = random.shuffled(words[0:k1])[0:patience]

    # greedy selection
    for word in words_k1:
        text_embeddings = get_description_embeddings(
            word_chain + " " + word)
        next_accuracy = get_accuracy(image_embeddings,
            text_embeddings, y_truth)
        if next_accuracy > accuracy:
            word_chain = word_chain + " " + word
    soup = soup + [word_chain]

return soup
```

Limitations

Similar to many related works, the main limitation of our work is that we require the source dataset to cover a broad range of classes (e.g. ImageNet). As a counter example, we cannot hope to train on pets classification and generalize to ImageNet. We highlighted this limitation in Table 1 of the main paper (top) with qualitative examples.

A. Training details

Images are not augmented during the greedy descriptor selection process; image augmentation during finetuning is consistent with prior work. Descriptors are always selected using the pretrained model parameters. Selecting descriptors based on finetuned model weights would be sub-optimal, since the pretrained text encoder captures a richer set of textual information. Remaining details are organized in Table 7. Mini-batches are randomly sampled, but with exactly one sample per label per batch. Cross entropy and CLIPood both tune the last three layers of the image and text encoders, in addition to a shallow text prompt (like CoOp) at a higher learning rate. The only difference between Cross entropy and CLIPood is the loss function; the latter method uses an adaptive margin. We use cross entropy loss for all baselines except ProDA and ProGrad. ProDA and ProGrad consume more GPU memory during training, so we were unable to fit them onto a single A40 GPU when training with cross entropy. Consequently, we were forced to use a CLIP-like contrastive loss for these two methods to reduce the number of text encoder evaluations.

B. Additional Word Soup Motivation

A natural baseline for word soup is soft prompt tuning (CoOp), since the former method can be thought of as "discrete" prompt tuning. Soft prompt tuning optimizes over a continuous parameter space using gradient descent, whereas

General Parameters	
batch size	64
learning rate	tuned per method
weight decay	1e-5
number of iterations	750
learning rate decay	none
softmax temperature	60
optimizer	SGD momentum=0.9
label smoothing	0
EMA weight averaging β	0.995
Prompt Tuning Parameters	
CoOp prompt length	3
CoOp prompt depth	1 (shallow)
MaPLe prompt depth	3
MaPLe prompt length	3
CoOp prompt initialization	"a photo of"
text prompt learning rate multiplier	10 $\times$
Word Soup and Diversity Loss Parameters	
k_0	250
k_1	1000
patience	250
λ	0.25
τ_0	10
Optimal Learning Rates	
Cross entropy	2e-5
CLIPood [51]	2e-5
CoOp [70]	8e-5
MaPLe [25]	0.025
KgCoOp [22]	4e-5
ProDA [31]	3.2e-4
ProGrad [71]	1.28e-3
VPT [20]	0.8
bitfit [64]	1.25e-4
CLIP-adapter [11]	6e-3
SSF [29]	1e-4
adapter [16]	2.5e-3
LoRA [17]	1e-5

Table 7. Miscellaneous training details for training on 16-shot ImageNet-1K in the OOD setting.

word soup optimizes over a discrete parameter space using a greedy algorithm. Many prior works (e.g. [58, 59]) observe that gradient descent is limited to a narrow convex basin around the initialization, when finetuning a pretrained deep model. This can be shown by linearly interpolating between the pretrained and finetuned parameters, similar to Fig. 6. In this figure, we plot in orange both the source and target error for interpolations between a *randomly initialized* descriptor

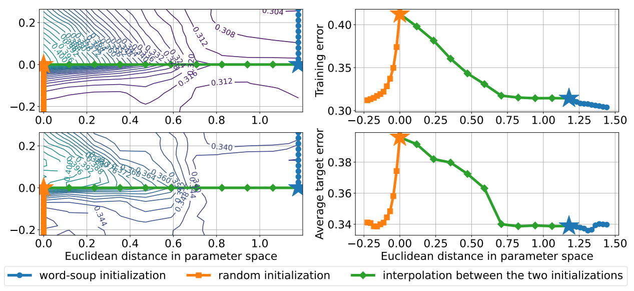

Figure 6. Contour plot of the 0-1 loss over the 2D parameter space spanned by two initializations (indicated by stars) and the fine-tuned parameters. The orange and blue stars indicate the random initialization and word soup initialization, resp. The top and bottom rows plot the 0-1 loss on the training and test data (average of 10 test datasets), resp. For this figure, we train 10 descriptor tokens. The plots on the right indicate the loss value at the corresponding locations in the contour plots on the left, for better visualization. We observe that the word soup initialization lies in a lower and flatter region, compared to the random initialization. Consequently, finetuning from the word soup initialization results in lower training and test errors compared to finetuning from the random initialization.

(orange star) and the finetuned soft descriptor. The resulting soft descriptor lies at the bottom of a sharp loss basin. On the other hand, the *word soup initialized* descriptor (blue star) lies at an equally low but much flatter region of the loss landscape. Finetuning from this initialization leads to a lower error on both source and target data, as indicated in blue. This visualization suggests that our word soup algorithm finds robust flat minima, since it is not limited to a narrow loss basin like gradient descent methods.

C. Token offset trick (for Descriptor Soup)

We propose a novel trick to augment/diversify the descriptors at test time to further increase the target accuracy of descriptor soups. This trick does not improve the performance of word soups significantly. Unlike the vision encoder, which has a cls token at a fixed position (either prepended or appended to the image tokens), the CLIP text encoder does not have a separate cls token. Instead, CLIP uses the output embedding which corresponds to the position of the end-of-sentence token in the input. In classification problems, the text inputs are generally short compared to the context size (number of total tokens). Consequently, the end-of-sentence token is always near the beginning of the sequence, with the remainder padded by null tokens. In this regime, there is never any information at the end of the input token sequence to attend to, so a large portion of the information in the pre-trained model is not used. We remedy this inefficient use of pretrained parameters by shifting the description toward the end of the sequence by t tokens. For example, if $t = 5$, we

	Source	Cross-dataset Evaluation Targets										Domain Generalization Targets				
	INet	Caltech	Pets	Cars	Flowers	Food	Aircraft	SUN	DTD	EuroSAT	UCF	**Mean**	INet-V2	Sketch	INet-A	INet-R	**Mean**
CLIP ZS	67.1	93.3	89.0	65.4	71.0	85.7	25.0	63.2	43.6	46.7	67.4	65.02	61.0	46.6	47.2	74.1	57.22
Word soup	68.8	94.1	89.5	65.9	**72.6**	**86.3**	**26.1**	67.2	45.3	**53.9**	67.8	**66.87**	62.6	49.0	**50.4**	77.0	59.73
Vanilla CoOp	68.7	94.4	90.2	**66.1**	70.9	85.8	26.0	66.7	**47.4**	50.1	68.9	66.63	61.9	48.6	49.8	76.7	59.26
+ Word soup	**69.1**	**94.6**	**91.1**	65.2	71.8	86.0	25.1	**67.4**	46.0	51.9	**69.1**	66.82	**62.7**	**49.4**	50.3	**78.0**	**60.09**

Table 8. Experiments using a different source dataset (a 16-shot subset of LAION-2B queried using ImageNet label names). Settings are identical to Table 10 (the expanded form of Table 6 in the main paper).

have:

- **original:** a photo of a dog, which may be large or small.
- **augmented:** a photo of a dog, ! ! ! ! ! which may be large or small. ("!" denotes the null token)

For all experiments with token offsets, we set $t = \{0, 5, 10, 15, 20, 25\}$ for a total of 6 augmented copies per descriptor. This diversifies the text embeddings at the expense of increasing the text centroid evaluation time 6-folds.

D. Centroid vs. Score Mean Evaluation

In this work, we presented both centroid and score mean results for both our soup methods and ensemble baselines. Centroid evaluation refers to averaging the text features among descriptors before calculating the cosine similarity between image and text features. Score mean evaluation refers to calculating the cosine similarity between image and text features and then averaging the similarity scores among descriptors.

Concretely, let there be m descriptors and c classes. Let $\mathbf{x}_I$ denote a normalized image feature and $\mathbf{x}_{T,k}^j$ denote the normalized text feature corresponding to class k and descriptor j; $k \in [1:c]$ and $j \in [1:m]$.

The predicted score for class k using centroid evaluation, s_k, is defined as:

$$\overline{\mathbf{x}}_{T,k} = \frac{1}{m} \sum_{j=1}^{m} \mathbf{x}_{T,k}^j$$

$$s_k = \left\langle \mathbf{x}_I, \frac{\overline{\mathbf{x}}_{T,k}}{\|\overline{\mathbf{x}}_{T,k}\|} \right\rangle$$

The predicted score for class k using score mean evaluation is defined as:

$$s_k = \frac{1}{m} \sum_{j=1}^{m} \left\langle \mathbf{x}_I, \mathbf{x}_{T,k}^j \right\rangle$$

Empirically, we found that score mean evaluation usually leads to small numerical improvements. However, in large scale applications where retrieval speed is crucial, centroid evaluation can be more efficiently implemented than score mean evaluation, due to the existence of fast nearest neighbor retrieval frameworks.

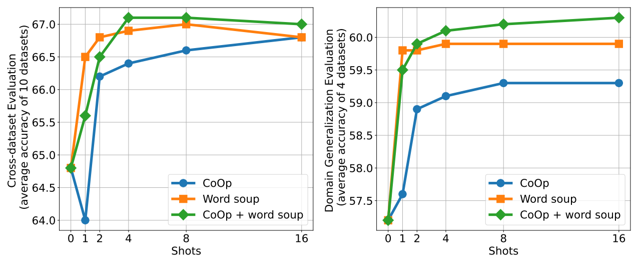

Figure 7. Different number of shots. We experiment with the same 14 datasets as the main paper and report average of 3 random trials. We report average target accuracies over 10 diverse datasets (left) and 4 ImageNet shifts (right). Here we verify that the improvements of both word soup and CoOp + word soup over CoOp are resilient to the number of shots. Indeed, we emphasize that **word soup is very resilient in extreme low shot scenarios due to the low number of parameters.**

E. Additional Ablation Studies

We present additional ablation studies in Table 8 and Figure 7. Table 8 presents OOD generalization results with a different source data set. Figure 7 presents results with different number of shots.

Parameter Efficiency Fig. 2 compares the parameter efficiency of our word soups against PEFT baselines. We observe that word soup can achieve the maximal CoOp accuracy using 25$\times$ and 70 $\times$ fewer parameters on the XD and DG benchmarks, resp. This impressive reduction in parameter storage requirements is due to the discrete nature of word soup parameters. A discrete token requires only one integer parameter, while a soft token requires 512 floating-point parameters.

Computational Efficiency We emphasize that our method adds negligible test time computation, despite requiring m

text encoder evaluations per label. For classification tasks, more time is spent processing image data compared to text data. For example, the evaluation of the $m = 8$ word soup in Table 6 took 239 seconds, of which 234 seconds were spent evaluating image embeddings and only 4.6 seconds were spent evaluating text embeddings.

		Source	Cross-dataset Evaluation Targets											Domain Generalization Targets				
	m	INet	Caltech	Pets	Cars	Flowers	Food	Aircraft	SUN	DTD	EuroSAT	UCF	**Mean**	INet-V2	Sketch	INet-A	INet-R	**Mean**
CLIP ZS	1	67.1	93.3	89.0	65.4	71.0	85.7	25.0	63.2	43.6	46.7	67.4	65.02	61.0	46.6	47.2	74.1	57.22
Vanilla CoOp	1	70.0	94.6	91.2	65.4	71.2	86.3	24.6	66.9	**48.0**	48.3	68.7	66.52	63.2	48.4	49.2	76.2	59.25
+ word soup	8	69.6	94.6	90.8	65.2	70.3	86.0	24.8	66.9	47.6	50.7	69.0	66.59	62.9	48.2	49.6	76.3	59.26
CoOp ensemble	8	69.8	94.4	**91.5**	**66.2**	**72.6**	**86.6**	**25.7**	67.7	46.4	47.9	67.8	66.68	63.0	48.4	49.6	75.8	59.18
CoOp regularized towards initialization	1	70.2	**94.8**	91.1	65.4	72.1	86.2	24.8	67.6	46.2	52.7	69.0	66.97	**63.6**	49.1	49.6	77.5	59.94
+ word soup	8	69.9	94.7	90.1	64.7	71.8	85.5	25.0	67.4	45.5	53.6	68.7	66.69	63.4	49.2	49.9	77.7	60.05
CoOp with label smoothing	1	70.1	94.5	90.6	64.9	72.0	85.8	24.6	67.3	45.4	50.0	68.6	66.37	63.4	49.1	**50.2**	77.6	60.09
+ word soup	8	69.9	94.5	89.9	64.9	71.7	85.2	25.0	66.8	44.8	50.0	68.3	66.13	63.6	49.3	50.1	77.7	60.16
CoOp + word soup ($\lambda = 0$)	8	69.8	94.3	90.8	64.8	71.1	86.0	24.1	67.2	46.8	48.4	68.8	66.21	63.2	48.3	49.0	76.1	59.15
+ **our diversity loss** ($\lambda = 0.25$)	8	**70.2**	94.7	91.0	65.4	72.3	86.0	24.8	**67.8**	45.9	**55.2**	**69.2**	**67.23**	63.6	**49.3**	50.1	**77.9**	**60.20**

Table 9. Ablation results to support the diversity loss. "Vanilla CoOp + word soup" refers to naively appending the word soup descriptors trained on the pretrained model to the separately trained soft CoOp prompts. "CoOp ensemble" refers to ensembling m randomly-initialized soft descriptors. This requires running CoOp m times, but offers negligible gains in accuracy. In the second half of the table, we fix the descriptor tokens and train the prompt tokens only. We first run CoOp with standard CE training ($\lambda = 0$) and observe a decrease in accuracy compared to the naive "Vanilla CoOp + word soup" baseline, caused by the diversity collapse issue observed in Figure 4. We then attempt to simply minimize the KL divergence between the training prediction and the initial prediction; this shows that the diversity loss is not simply a form of regularization towards the initialization as in MIRO [4] and ProGrad [71]. Finally, we train using our diversity loss with $\lambda = 0.25$, which achieves a 1% increase in accuracy on average. Average of 3 trials. This is an expanded version of Table 4 in the main paper.

		Source	Cross-dataset Evaluation Targets											Domain Generalization Targets				
	m	INet	Caltech	Pets	Cars	Flowers	Food	Aircraft	SUN	DTD	EuroSAT	UCF	**Mean**	INet-V2	Sketch	INet-A	INet-R	**Mean**
CLIP ZS [43]	1	67.1	93.3	89.0	65.4	71.0	85.7	25.0	63.2	43.6	46.7	67.4	65.02	61.0	46.6	47.2	74.1	57.22
CoOp [70]†		71.51	93.70	89.14	64.51	68.71	85.30	18.47	64.15	41.92	46.39	66.55	63.88	64.20	47.99	49.71	75.21	59.3
Co-CoOp [69]†		71.02	94.43	90.14	65.32	71.88	86.06	22.94	67.36	45.73	45.37	68.21	65.74	64.07	48.75	50.63	76.18	59.9
MaPLe [25]†		70.72	93.53	90.49	65.57	72.23	86.20	24.74	67.01	46.49	48.06	68.69	66.30	64.07	49.15	50.90	76.98	60.3
CLIPood [51]†		71.6												64.9	49.3	50.4	77.2	60.5
Cross Entropy (CE)	1	**72.3**	94.6	89.8	64.9	72.4	**86.3**	25.3	68.1	45.7	51.5	69.4	66.80	**65.4**	49.4	49.8	77.0	60.39
+ GPT score mean [35]	5.8	71.7	94.3	89.9	64.5	72.1	86.0	24.5	**68.6**	**46.6**	53.8	68.4	66.86	64.9	49.4	48.8	76.6	59.92
+ Random descriptors	32	71.6	94.6	89.3	64.7	72.1	86.0	25.3	67.5	45.4	55.2	68.8	66.89	64.8	49.9	50.2	77.9	60.69
+ Waffle CLIP [45]	32	71.6	94.1	89.8	65.0	72.6	86.1	26.1	67.7	45.0	50.9	68.4	66.58	65.1	49.7	50.3	77.4	60.65
+ Descriptor soup (ours)	16.7	72.1	**94.7**	89.9	65.0	72.4	**86.3**	25.6	68.0	45.6	53.9	**69.5**	67.10	65.3	49.7	50.1	77.7	60.70
+ offset trick (ours)	100	72.1	94.1	**90.4**	**66.3**	**73.3**	**86.3**	**26.1**	67.8	46.4	55.0	69.4	**67.51**	65.3	49.8	50.8	78.2	61.01
+ Word soup centroids (ours)	8	71.8	94.4	**90.4**	65.0	72.3	86.1	25.3	68.2	45.5	55.4	69.1	67.16	65.2	50.2	50.7	**78.7**	61.22
+ Word soup score mean (ours)	8	71.7	94.5	90.2	65.1	72.4	86.2	25.6	68.1	45.6	**57.3**	69.3	67.43	65.3	**50.3**	**50.9**	**78.7**	**61.32**
+ Descriptor soup upper bound	11	71.7	94.4	90.2	66.5	72.9	86.1	26.3	67.4	46.4	57.2	68.6	67.62	64.9	49.7	50.9	78.6	61.01
ProGrad [71]	1	69.8	94.4	**91.5**	**65.8**	**72.4**	**86.4**	**25.3**	66.6	47.2	46.3	69.0	66.48	63.2	48.2	48.6	75.9	58.96
KgCoOp [22]	1	69.2	94.3	89.9	63.9	71.0	85.7	23.7	66.2	44.4	54.4	68.3	66.16	62.3	48.0	48.8	75.5	58.64
ProDA [31]	32	70.0	94.2	90.2	64.7	70.8	85.7	23.1	67.0	45.8	51.4	**69.4**	66.23	63.0	48.1	48.4	75.7	58.83
Vanilla CoOp [70]	1	70.0	94.6	91.2	65.4	71.2	86.3	24.6	66.9	**48.0**	48.3	68.7	66.52	63.2	48.4	49.2	76.2	59.25
+ Word soup score mean (ours)	8	**70.2**	**94.7**	90.9	65.4	72.0	86.0	25.0	**67.7**	45.9	**56.2**	69.2	**67.30**	**63.6**	**49.3**	**50.1**	**77.9**	**60.25**
Vanilla MaPLe [25]	1	70.7	93.7	**91.2**	**65.4**	**71.9**	**86.2**	**25.0**	**67.2**	**46.2**	48.6	**68.9**	66.44	63.9	48.6	48.4	76.3	59.32
+ Word soup score mean (ours)	8	**70.8**	**94.1**	91.2	65.2	71.8	85.8	24.0	67.0	46.0	**53.5**	68.0	**66.65**	**64.0**	**49.6**	**49.2**	**77.9**	**60.20**
Vanilla CLIPood [51]	1	**72.9**	**94.8**	89.8	**64.9**	72.2	85.9	**25.8**	67.8	**46.4**	48.7	68.7	66.50	**66.0**	49.5	49.5	76.9	60.47
+ Word soup score mean (ours)	8	72.0	94.4	**90.8**	64.8	**72.4**	**86.0**	25.4	**67.9**	46.0	**57.6**	**68.9**	**67.42**	65.5	**50.2**	**50.8**	**78.5**	**61.23**

Table 10. Comparison with few-shot methods and few-shot methods stacked with ZS methods. † indicates author-reported numbers on the same datasets with the same train-test splits. Other numbers are from our reproductions using our github code. We tune all baselines on a withheld validation set, so our numbers are different from published numbers. The descriptor soup upper bound was trained to maximize average cross-dataset accuracy (on test data); this loosely approximates the maximally achievable accuracy on these benchmarks without using extra information. All other methods were *trained on 3 random 16-shot splits of ImageNet*. m indicates number of descriptors used. All methods are evaluated on top of 3 models finetuned with different random seeds. Due to space limitations, we only compare with ZS baselines stacked on top of the CE-finetuned few-shot model, since this is the best finetuned model. Either our descriptor soup with the offset trick or our word soup achieves the best accuracy on most datasets. Finally, we stack our word soup method on top of CoOp, MaPLe, and CLIPood finetuned models to show that word soup is complementary to most existing robust finetuning methods. Average of 3 trials. This is an expanded version of Table 6 in the main paper.

	parameters (thousands)	INet	Caltech	Pets	Cars	Flowers	Food	Aircraft	SUN	DTD	EuroSAT	UCF	Average	INet-V2	INet-Sketch	INet-A	INet-R	Average
		Source	**Cross-dataset Evaluation Targets**											**Domain Generalization Targets**				
VPT shallow 1 token	0.768	68.7	93.8	90.0	65.1	69.5	85.3	24.2	66.0	44.7	41.9	67.8	64.84	62.1	47.9	47.9	76.7	58.67
VPT shallow 2 tokens	2	68.7	93.8	90.0	65.2	69.5	85.2	24.2	66.2	44.8	42.3	67.1	64.84	62.2	48.0	47.3	76.7	58.54
VPT shallow 3 tokens	2	68.7	93.9	90.0	65.6	70.2	85.3	24.8	66.2	44.7	43.8	67.5	65.20	62.4	48.1	47.0	76.6	58.52
VPT shallow 3 tokens	2	68.6	93.8	89.5	64.8	70.1	85.3	24.1	66.1	44.5	45.4	67.7	65.12	62.1	48.0	47.1	76.4	58.41
VPT deep 2 layers	5	68.8	93.5	89.7	65.0	70.3	85.4	24.0	65.9	44.7	49.3	67.6	65.54	62.2	48.2	46.9	76.6	58.47
VPT deep 3 layers	7	68.7	93.5	89.4	65.3	70.4	85.3	24.2	66.2	44.8	45.0	67.5	65.16	62.3	48.2	46.8	76.4	58.42
MaPLe 1 layer	396	70.1	94.2	91.1	64.3	71.1	86.1	24.5	67.0	47.3	51.8	68.6	66.61	63.4	48.4	48.8	76.3	59.22
MaPLe 2 layers	397	70.4	93.6	**91.8**	64.3	71.3	85.9	24.7	67.0	46.9	48.1	68.5	66.21	63.7	48.3	49.2	76.1	59.34
MaPLe 3 layers	399	**70.7**	93.7	91.2	65.4	71.9	86.2	25.0	67.2	46.2	48.6	68.9	66.44	**63.9**	48.6	48.4	76.3	59.32
bitfit last layer	17	68.3	94.1	89.5	65.2	71.4	85.9	24.9	65.7	44.7	46.9	67.9	65.62	61.7	48.0	48.5	75.9	58.51
bitfit last 2 layers	34	68.8	93.9	89.9	65.3	71.4	85.9	25.1	66.4	45.1	47.4	68.4	65.88	62.1	48.6	48.5	76.6	58.93
bitfit last 3 layers	51	69.1	93.9	90.0	65.3	71.7	85.8	25.0	66.7	45.4	48.3	68.4	66.05	62.6	48.7	48.5	76.8	59.12
CoOp 1 token	0.512	69.4	94.3	91.4	64.4	71.7	86.3	24.6	67.2	47.3	49.1	68.5	66.49	63.1	48.2	49.0	76.1	59.08
CoOp 2 tokens	1	69.9	**94.6**	91.6	65.5	72.0	86.1	25.0	66.8	**48.2**	49.6	**69.4**	66.89	63.2	48.5	48.8	76.3	59.20
CoOp 3 tokens	2	70.2	94.5	91.0	**66.0**	71.6	86.3	24.6	66.8	47.6	49.0	68.9	66.63	63.4	48.5	49.5	76.3	59.45
ProGrad 1 token	0.512	69.4	94.2	91.0	65.6	**72.7**	86.4	25.1	66.2	46.0	48.2	68.5	66.39	62.8	48.1	48.5	75.7	58.77
ProGrad 2 tokens	1	69.5	94.1	90.8	65.7	72.6	86.3	24.8	66.5	45.5	47.7	68.7	66.28	62.8	48.0	48.5	75.7	58.75
ProGrad 3 tokens	2	69.8	94.4	91.5	65.8	72.4	86.4	25.3	66.6	47.2	46.3	69.0	66.48	63.2	48.2	48.6	75.9	58.96
KgCoOp 1 token	0.512	68.6	93.4	89.4	63.4	70.9	85.9	23.8	65.6	44.9	52.5	68.1	65.80	62.0	47.8	49.1	75.7	58.63
KgCoOp 2 tokens	1	69.0	93.3	89.3	62.8	70.2	85.8	23.8	66.0	45.4	53.0	69.0	65.85	62.4	48.0	49.1	75.9	58.85
KgCoOp 3 tokens	2	69.2	94.3	89.9	63.9	71.0	85.7	23.7	66.2	44.4	54.4	68.3	66.16	62.3	48.0	48.8	75.5	58.64
ProDA ensemble size 4	20	70.5	94.3	90.4	65.3	71.2	86.1	24.9	67.2	46.4	50.4	69.4	66.54	63.6	48.6	49.4	76.0	59.43
ProDA ensemble size 8	41	70.1	93.8	90.3	65.1	71.0	85.8	24.9	67.4	45.5	49.4	68.4	66.15	63.3	48.8	49.5	76.6	59.55
ProDA ensemble size 16	82	69.9	94.3	90.5	64.5	70.8	85.6	24.3	66.6	45.2	48.4	68.8	65.90	63.1	48.4	48.9	76.1	59.13
ProDA ensemble size 32	164	70.0	94.2	90.2	64.7	70.8	85.7	23.1	67.0	45.8	51.4	69.4	66.23	63.0	48.1	48.4	75.7	58.83
ProDA ensemble size 64	328	69.4	94.4	90.0	64.5	69.5	85.1	22.7	66.4	44.9	49.6	67.8	65.49	62.7	48.0	48.7	76.2	58.91
CLIP-adapter reduction=128	4	67.1	93.3	89.0	65.3	70.9	85.7	25.1	63.3	43.5	46.6	67.4	65.00	60.9	46.6	47.2	74.1	57.18
CLIP-adapter reduction=64	8	67.1	93.3	88.8	65.4	71.1	85.7	24.9	63.3	43.5	46.5	67.2	64.97	60.9	46.5	47.2	74.0	57.17
CLIP-adapter reduction=32	16	67.4	93.2	88.4	65.2	70.1	85.6	24.9	64.1	44.0	46.3	66.8	64.84	60.9	46.9	47.9	74.5	57.55
CLIP-adapter reduction=16	33	67.6	93.3	88.3	64.9	70.1	85.6	24.5	64.4	43.9	46.7	66.8	64.86	61.2	47.2	48.4	75.1	57.98
CLIP-adapter reduction=8	66	67.9	93.4	88.7	65.4	70.2	85.7	24.8	65.1	44.3	46.6	66.7	65.09	61.5	47.5	48.5	75.3	58.21
CLIP-adapter reduction=4	131	67.8	93.4	89.0	65.2	70.2	85.7	24.5	65.2	44.2	46.0	66.8	65.02	61.5	47.5	48.3	75.1	58.12
SSF last layer	12	68.1	94.0	89.5	65.4	71.0	85.7	24.7	65.6	45.3	51.6	68.5	66.13	61.6	47.8	46.4	75.7	57.87
SSF last 2 layers	25	68.5	94.1	89.9	65.1	71.2	85.8	24.8	66.3	45.9	49.1	68.2	66.04	62.1	48.3	47.2	76.3	58.46
SSF last 3 layers	37	68.5	94.2	89.5	64.9	71.2	85.3	24.4	66.2	45.8	49.3	67.8	65.86	62.1	48.1	47.2	76.3	58.44
LoRA rank=1	18	67.3	93.5	89.3	65.4	71.3	85.7	25.1	64.2	44.4	47.9	67.6	65.43	61.4	47.1	46.9	74.9	57.59
LoRA rank=2	37	67.6	93.7	90.0	65.7	71.2	85.7	25.3	65.6	45.9	49.6	67.8	66.05	61.9	47.7	45.3	75.6	57.62
LoRA rank=4	74	67.6	93.8	90.1	65.7	71.5	85.7	25.2	65.4	46.0	50.9	67.7	66.19	61.8	47.7	46.2	76.0	57.93
LoRA rank=8	147	68.0	93.9	90.0	65.7	71.4	85.4	25.5	65.9	46.3	52.6	67.2	66.39	61.9	47.1	42.2	74.4	56.40
ResBlock-adapter reduction=128	55	68.0	93.8	89.2	64.0	71.1	84.7	23.3	65.1	45.3	46.0	67.6	65.01	61.2	47.4	47.2	75.5	57.81
ResBlock-adapter reduction=64	111	68.8	94.0	89.7	64.2	70.8	85.0	23.5	65.8	45.5	46.9	68.0	65.35	61.8	48.0	48.0	76.3	58.52
ResBlock-adapter reduction=32	221	69.1	94.2	90.0	64.4	71.4	85.3	23.2	66.1	45.2	46.8	67.4	65.41	62.5	48.1	48.3	76.8	58.94
ResBlock-adapter reduction=16	442	69.3	94.2	89.9	64.2	71.3	85.3	23.8	66.4	45.6	47.5	67.9	65.60	62.8	48.4	48.4	76.9	59.12
ResBlock-adapter reduction=8	885	69.5	94.1	89.5	64.6	71.3	85.6	23.6	66.6	44.8	45.3	67.9	65.33	63.0	48.6	48.8	77.0	59.36
ResBlock-adapter reduction=4	1769	69.7	94.1	89.5	64.8	71.2	85.5	24.0	66.8	44.9	46.8	67.8	65.55	63.1	48.7	49.0	77.1	59.48
Word Soup $m = 1$	**0.012**	68.6	93.9	89.2	64.6	71.8	86.0	24.7	65.9	44.2	48.0	67.7	65.61	62.1	47.9	49.7	76.3	59.01
Word Soup $m = 2$	0.024	69.0	94.1	90.3	65.6	72.5	86.0	25.5	66.9	45.0	52.0	68.6	66.64	62.4	48.8	50.2	76.6	59.50
Word Soup $m = 4$	0.048	69.3	94.1	89.9	65.9	72.4	**86.5**	25.7	67.1	45.8	53.6	68.7	66.96	62.9	48.9	50.3	77.2	59.80
Word Soup $m = 8$	0.096	69.4	94.1	89.9	65.7	72.5	86.4	25.9	67.0	44.9	54.6	68.8	66.99	63.1	49.0	50.5	77.3	59.95
Word Soup $m = 16$	0.192	69.5	94.0	89.9	65.9	72.5	86.3	26.1	67.4	45.2	54.8	68.8	67.08	63.2	49.0	50.7	77.2	60.02
Word Soup $m = 32$	0.384	69.6	94.2	89.9	65.9	72.4	86.5	**26.2**	67.4	45.1	54.7	69.0	67.12	63.2	49.0	50.6	77.3	60.04
Word Soup $m = 64$	0.767	69.5	94.1	90.0	65.9	72.5	86.4	26.2	67.4	45.2	55.1	69.0	67.17	63.3	49.1	**50.7**	77.4	60.11
Word Soup + CoOp $m = 4$	2	70.2	94.5	91.0	65.6	72.3	86.0	25.1	67.7	45.7	**56.1**	68.6	67.26	63.7	49.3	50.1	77.9	60.26
Word Soup + CoOp $m = 8$	2	70.2	94.4	91.0	65.3	72.1	86.1	25.2	67.7	45.5	55.5	68.7	67.15	63.5	49.3	50.2	**78.0**	60.25
Word Soup + CoOp $m = 16$	2	70.2	94.5	91.0	65.7	72.6	86.1	24.9	**67.8**	45.6	55.5	69.2	**67.30**	63.7	**49.5**	50.5	77.9	**60.39**

Table 11. Detailed numerical results for PEFT comparison in Fig. 2. Average of 3 trials. These results are plotted in Figure 2 of the main paper. Also reference Section 7 (Results) for a discussion.

		Source	**Cross-dataset Evaluation Targets**											**Domain Generalization Targets**				
	m	INet	Caltech	Pets	Cars	Flowers	Food	Aircraft	SUN	DTD	EuroSAT	UCF	**Mean**	INet-V2	Sketch	INet-A	INet-R	**Mean**
Open-AI CLIP ViT-B/32																		
ZS	1	61.9	91.5	87.4	60.3	66.4	80.2	19.1	62.2	42.3	40.3	63.5	61.32	54.6	40.7	29.1	66.3	47.68
GPT score mean	5.8	63.0	91.8	**88.1**	60.0	66.6	80.2	19.1	64.4	43.1	36.2	62.7	61.22	55.4	41.0	29.4	65.9	47.95
Waffle CLIP	16	63.3	**91.8**	88.0	**60.9**	**67.4**	80.4	19.6	63.8	41.7	44.8	63.0	62.13	55.8	41.6	31.1	67.8	49.07
Desc. soup + offsets	100	64.1	91.5	87.7	60.7	66.9	80.4	**19.9**	64.4	**43.6**	**48.3**	**64.5**	**62.79**	56.5	**42.6**	31.8	**69.3**	**50.05**
Word soup	8	**64.5**	91.5	88.0	60.4	67.0	**80.9**	19.3	**64.6**	42.0	45.5	63.2	62.24	**56.9**	42.5	**32.0**	68.7	50.00
Open CLIP ViT-L/14																		
ZS	1	73.3	96.4	**92.9**	92.0	75.8	85.7	34.1	72.7	57.3	52.1	72.1	73.11	65.6	61.0	47.2	85.7	64.88
GPT score mean	5.8	73.6	**96.7**	92.8	91.2	**76.5**	85.3	33.7	72.7	58.6	51.6	71.7	73.08	66.1	61.2	47.5	85.1	64.96
Waffle CLIP	16	72.7	96.1	92.4	91.7	76.4	85.8	34.4	72.4	58.6	52.2	72.5	73.25	65.3	60.7	46.5	85.4	64.47
Desc. soup + offsets	100	74.0	96.6	92.8	92.0	76.3	85.5	34.5	72.7	**59.1**	50.0	72.3	73.19	66.0	**61.9**	**48.7**	**86.6**	**65.81**
Word soup	8	**74.3**	96.5	92.1	**92.2**	76.0	**86.0**	**35.0**	**73.6**	58.5	**52.9**	**73.0**	**73.56**	**66.8**	61.6	48.2	86.3	65.73
Open CLIP CoCa-L/14																		
ZS	1	75.1	**97.6**	93.8	92.7	77.3	87.5	36.6	73.6	57.2	58.5	73.4	74.82	67.5	63.5	53.8	87.0	67.94
GPT score mean	5.8	74.9	**97.6**	93.7	92.4	76.2	87.3	36.3	73.9	58.9	**64.9**	73.6	75.48	67.6	63.5	52.8	86.8	67.67
Waffle CLIP	16	75.0	97.5	93.9	92.7	77.3	87.5	37.4	73.1	57.5	63.0	73.9	75.37	67.5	63.8	52.8	87.3	67.85
Desc. soup + offsets	100	75.5	97.5	**93.9**	92.6	77.5	87.3	37.2	73.8	**61.1**	63.6	**75.0**	75.95	68.0	**64.2**	53.2	87.9	68.32
Word soup	8	**75.9**	97.5	93.8	**92.8**	**77.8**	**87.7**	**38.4**	**74.1**	60.5	63.5	74.7	**76.08**	**68.8**	64.0	**54.3**	**87.9**	**68.73**
Open CLIP ViT-g/14																		
ZS	1	77.7	97.7	93.6	93.5	**81.6**	**90.0**	44.1	74.3	65.3	55.8	**80.0**	77.58	70.4	66.4	59.7	89.0	71.37
GPT score mean	5.8	77.6	97.2	93.7	93.6	81.4	89.6	43.1	74.7	63.1	58.7	76.3	77.14	71.0	66.3	58.8	88.9	71.26
Waffle CLIP	16	77.3	97.8	93.5	93.7	81.3	89.8	**44.1**	74.1	65.8	58.0	78.9	77.72	70.1	65.9	59.0	88.9	70.99
Desc. soup + offsets	100	78.0	**97.8**	**94.1**	**93.9**	80.7	89.2	43.1	**75.0**	**67.0**	**60.4**	79.2	78.04	71.5	**67.2**	**60.2**	**90.0**	**72.21**
Word soup	8	**78.4**	97.6	93.7	93.9	81.4	89.8	44.0	75.0	66.0	60.0	79.5	**78.09**	**71.6**	67.1	60.0	89.6	72.05

Table 12. Detailed numerical results for different model scales. This is an expanded version of Table 5. Average of 3 trials.