Abstract

Few-shot recognition involves training an image classifier to distinguish novel concepts at test time using few examples (shot). Existing approaches generally assume that the shot number at test time is known in advance. This is not realistic, and the performance of a popular and foundational method has been shown to suffer when train and test shots do not match. We conduct a systematic empirical study of this phenomenon. In line with prior work, we find that shot sensitivity is broadly present across metric-based few-shot learners, but in contrast to prior work, larger neural architectures provide a degree of built-in robustness to varying test shot. More importantly, a simple, previously known but greatly overlooked class of approaches based on cosine distance consistently and greatly improves robustness to shot variation, by removing sensitivity to sample noise. We derive cosine alternatives to popular and recent few-shot classifiers, broadening their applicability to realistic settings. These cosine models consistently improve shot-robustness, outperform prior shot-robust state of the art, and provide competitive accuracy on a range of benchmarks and architectures, including notable gains in the very-low-shot regime.

1 Introduction

Modern ConvNets have achieved impressive performance [12, 23, 22] but rely on millions of labelled images [7, 15, 18]. In many contexts this high level of data availability cannot be assumed. Rare or fine-grained categories may require prohibitively expensive expert gatherers or annotators, respectively. High-stakes, open-world settings like robotics compound the issue: models will likely need to learn and adapt rapidly in deployment, without waiting for offline data collection or annotation.

These issues have spawned interest in neural models that rapidly adapt to novel concepts using few labelled examples, or few-shot learning. In the most highly-studied incarnation of this problem (few-shot image classification), a classifier is pre-trained on a set of training classes, and then must adapt to a distinct set of unseen test classes, with only a few (typically 1 or 5) labelled images representing each novel class. Most approaches to few-shot training involve repeated simulations (episodes) of this test-time task: batches include 1 or 5 labelled images, which the network uses to make predictions. Optimizing over repeated trials, networks learn to generalize across classes, eventually including novel ones. The episodic training scheme has proven overwhelmingly popular [3, 26, 17, 5, 34].

Inherent to the episodic training scheme is the assumption that the number of images per class, or shot, is known in advance. Models for 5-shot problems are trained on 5-shot episodes; models for 1-shot problems use 1-shot episodes. Clearly this is not realistic, as the practitioner cannot control or know in advance how available as yet unseen data will be in deployment. Prior work shows that this

---

*Work performed while a graduate student at Cornell University
†Equal contribution
assumption does impact performance. Prototypical networks \[26\] (ProtoNet), a simple, foundational, and enduringly popular approach to few-shot classification, overfit severely to training shot and suffer when test shot does not match \[3\]. This is because models trained with different shot numbers require different degrees of clustering to be effective. Models trained on low-shot episodes must learn tight class clusters, otherwise different training samples will shift decision boundaries greatly (Fig. 1 top). High shot models aggregate class statistics over many images, regressing nicely to the mean, and so learn diffuse clusters that are information-rich but susceptible to sample noise when shot is low \[3\].

We propose a simple yet greatly overlooked solution to the problem of shot-sensitivity: simply replace Euclidean distance logits with the cosine distance. The existence of cosine alternatives to Euclidean-based few-shot learners has long been known - the original ProtoNet paper even contains a discussion of this topic \[26\, 1\]. However, when cosine and Euclidean alternatives are compared, they are always evaluated in the episodic setting without shot mismatch, and reported differences are generally small \[26\, 35\]. We find that when train and test shot differ, these differences become dramatic: cosine models are much more robust to shot variation at test time. This is because the cosine distance provides ample robustness to sample noise, by anchoring all decision boundaries through the origin. This makes decision boundaries more stable (Fig. 1 bottom), and by training in this stabilized regime, cosine models become effectively agnostic to shot.

Motivated by this finding, we implement cosine alternatives to two popular few-shot classifiers (ProtoNet \[26\] and FEAT \[35\]), and derive a novel alternative for a recent state of the art (FRN \[34\]). We then conduct a systematic comparison across benchmarks and network backbones. We find that:

1. Similar to \[3\], overfitting to training shot is a consistent problem for metric few-shot learning.
2. Contrary to \[3\], larger feature extractor backbones offer some degree of built-in shot robustness.
3. Our cosine alternatives provide consistent and superior test time shot sensitivity reduction...
4. ...without sacrificing model accuracy in expectation, and in fact frequently yield substantial (up to double-digit) accuracy gains in the very-low-shot regime.

These results suggest that the shot-overfitting problem, while ubiquitous to few-shot learning, also has a simple, elegant, and widely applicable solution.

## 2 Background and Prior Work

### 2.1 Few-Shot Classification

In the few-shot classification problem we are given labelled images from training classes, and asked to train a classifier that can adapt at test time to a disjoint set of novel classes, using only a few examples. Typical episodic training involves sampling tasks from an overarching task distribution - usually, by selecting subsets of the training classes to populate each batch or episode. Episodes are partitioned into a small support set and larger query set, and the classifier uses labels from the support set to classify the query set. During training, query set performance produces a loss used to optimize the model. At test time, queries are used to evaluate model accuracy. The number of support images per class is referred to as the shot, while the number of classes in an episode is the way. Thus an episode with five classes and a single labelled image for each class is a 5-way/1-shot problem.

At this point many approaches exist for tackling few-shot classification, broadly falling into two categories. **Optimization-based** approaches learn a good initialization for the classifier, such that rapid adaptation at test time over few examples is effective \[8\, 24\, 19\]. **Metric-based** approaches, meanwhile, learn a class-agnostic feature extractor such that under a chosen distance metric, images...
with high semantic similarity cluster together and become separable from those without. Established metrics include Euclidean distance [26, 35], cosine distance [29, 9], hyperbolic distance [14], Earth Mover’s Distance [36], projection distance [25, 34] and many others. For a more detailed overview of the many approaches to few-shot classification we refer the reader to [32].

We focus on the metric family of approaches, as these have proven both popular and successful at the few-shot classification task. A representative baseline is ProtoNet [26], where each class is represented by the average latent embedding over all the images belonging to that class. We begin our investigation with ProtoNet and then expand to more sophisticated architectures (see Sec. 3).

2.2 Shot-Robustness

Despite the obvious impracticality of knowing test shot in advance, prior research on robustness to shot variation is limited. Most relevant is [3], which shows that ProtoNet overfits to training shot because different shots demand different ratios of inter- to intra-class variance. Low-shot models cluster points tightly to overcome sample noise; test-time performance saturates rapidly with respect to shot, as class centroids regress to the mean and effectively ignore further samples. High-shot models tolerate high intra-class variance, as the average prototype over many images regresses nicely to the mean. The result is a higher ceiling for performance but a lower floor, as prototypes do not converge stably when test shot is low. [3] proposes a test-time projection layer (EST) maximizing the inter-/intra-class variance ratio, removing these concerns and serving as our competitive baseline.

Both [16] and [10] also analyze training shot as a hyperparameter, with [10] again identifying clustering behavior as the mediating mechanism. But both compare to non-episodic training and ignore shot mismatch. [33] investigates generalization of few-shot metric-based approaches to much higher way and shot, but does not disentangle these factors or identify any shot-overfitting behavior, besides generally lower performance on a more difficult task. In our work we undertake a systematic evaluation of shot mismatch across architectures, benchmarks, and metric-based approaches. We find that cosine metrics are consistently effective at reducing sensitivity to test-time shot variation.

2.3 Cosine Models

Cosine alternatives to Euclidean-based metric approaches have been known since the earliest formulations of the few-shot image recognition problem [26, 29]. Cosine models with Euclidean counterparts have also been proposed [9, 5]. This duality has been explored in more recent approaches [35]. To date, all comparisons have been based on accuracy, and are performed in the standard episodic setting, where train and test shot match. Differences are generally small, and the model with slightly better results is reported [26, 35, 9, 10]. While our results do not contradict these findings, we discover vast yet previously overlooked differences in behavior when train and test shot are not the same. We also extend the cosine distance beyond simple latent point-to-point comparisons, deriving a novel cosine alternative to the recently proposed FRN [34]. The following section provides further detail.

3 Method

We focus on three popular and/or recent metric techniques: ProtoNet [26], FEAT [35], and FRN [34]. We describe each model and our corresponding cosine alternatives in the following sections.

3.1 ProtoNet

ProtoNet is a popular and straightforward few-shot classification method. Given a set of \( nk \) labelled support images representing \( n \) classes of interest, the ProtoNet takes the latent embeddings for the \( k \) images in each class and computes an average class centroid, or prototype. Class probability logits are assigned to query images based on the negative squared Euclidean distance between query image embeddings and class centroids. We multiply the logits by a learned temperature scaling factor before the softmax layer, as prior work has shown this to have a large positive impact on performance [9, 5, 35, 34]. Since squared Euclidean distance grows linearly with latent dimensionality, we also normalize logits by the network width to ensure consistent performance across architectures. Thus if \( \mu_c \) is the class centroid for class \( c \), \( C \) is the set of support classes, \( \sigma \) is the learned temperature
scaling term, and \(d\) is the network width, then ProtoNet predictions are given by:

\[
p(y|x) = \frac{\exp\left(-\frac{d}{2}||x - \mu_y||^2\right)}{\sum_{c \in C} \exp\left(-\frac{d}{2}||x - \mu_c||^2\right)}
\]

(1)

The cosine alternative to this model is straightforward. We remove the \(1/d\) normalization since cosine distance does not grow with dimensionality, but keep the learned temperature scale, as cosine distance on its own is bounded by \(\pm 1\). Cosine ProtoNet predictions are then given by:

\[
p(y|x) = \frac{\exp(\sigma \bar{x}^T \mu_y)}{\sum_{c \in C} \exp(\sigma \bar{x}^T \mu_c)}
\]

(2)

where \(\bar{v}\) represents L2 normalization on the vector \(v\).

As a preliminary experiment, we verify our intuition from Fig. 1 that cosine distance stabilizes decision boundaries and unifies behavior across training shot regimes. Given a 5-way 1-shot episode, we observe the class predictions on query points from a simple 4-layer ProtoNet. We then resample just the support images, make a new set of predictions, and count how many change. If decision boundaries are stable and resistant to sample noise, then few predictions should change. After 1000 trials, agreement rates are compared to the same model but with cosine predictions (Eq. 2).

Table 1: 5-way/1-shot ProtoNet prediction consistency: when support images change, what proportion of predictions are preserved? Rows represent test-time prediction mechanism; columns represent training scheme. Cosine prediction is more robust to sample noise, and cosine training shows reduced sensitivity to shot value.

| Prediction | Training | Euclidean | Cosine |
|------------|----------|-----------|--------|
|            | 4-shot   | 32-shot   | 4-shot | 32-shot |
| Euclidean  | 48%      | 32%       | 37%    | 32%     |
| Cosine     | 53%      | 40%       | 55%    | 51%     |

To avoid model bias, we repeat the comparison across ProtoNets under four training regimes: Euclidean vs cosine, and 4- vs 32-shot. Models are trained on meta-iNat (details in Sec. 4.1) and results appear in Table 1. Our intuition is correct: cosine predictions are more consistent than Euclidean across all models, regardless of training (bottom row vs top row). We also see that cosine training stabilizes model behavior across different shots. Agreement rates for 4- vs 32-shot Euclidean models vary widely (col 1 vs 2), while agreement rates for cosine models are much closer (col 3 vs 4). Motivated by these findings, we now derive cosine alternative to more sophisticated few-shot learners.

### 3.2 FEAT

FEAT [35], or Few-shot Embedding Adaptation with Transformer, is a more recent and powerful approach based on ProtoNet. FEAT task-conditions the prototypes by passing them through a learned attention layer. Predictions are then made using Eq. 1 but with the refined centroids. The original paper performs a comparison to the cosine alternative using Eq. 2 and finds only small differences in performance. This is not surprising, as we find that the implementation of FEAT’s attentional transformer largely trivializes this distinction. The transformer head refines the centroids and then uses LayerNorm before classification; this operation recenters and L2-normalizes the input. To restore the distinction between Euclidean and cosine FEAT, therefore, we remove the LayerNorm from the FEAT attention head. This change did not greatly impact performance - we found that LayerNorm scores generally interpolate between cosine and Euclidean scores.

### 3.3 FRN

Feature map Reconstruction Networks (FRN) are a more recent state-of-the-art approach recasting the classification problem as reconstruction in feature space [34]. Rather than compare individual feature vectors, FRN aggregates feature maps into pools of features for each class, and attempts to reconstruct query features as weighted sums over features in the individual support pools. The support pool with the best reconstruction is the most likely class candidate. Reconstruction quality is measured via mean squared error from the target query features. FRN is chosen as a high performer, but also illustrates that cosine methods can generalize broadly beyond simple point to point comparisons.

FRN formulates the reconstruction problem as linear regression: given a matrix of class \(c\) support features \(S_c\), and a matrix of query features \(Q\), find \(\bar{W}\) such that \(\bar{W}S_c \approx Q\). For practical reasons we replace the original FRN regularizer, so our latent reconstruction problem is formulated as follows:
\[ W = \arg \min_W ||Q - WS_c||_2^2 + (\lambda||S_c^T S_c||_2)||W||^2_2 \]

(3)

where \( || \cdot ||_2 \) is the Frobenius norm and \( \lambda \) a learned weight. Discussion is provided in appendix A.3.

Deriving a cosine FRN is not straightforward. Calculating reconstruction quality with cosine distance leads to poor convergence: the reconstruction problem in Eq. (3) is optimized for mean squared error. Instead, we reformulate FRN as a comparison between support and query covariance matrices. Let \( \Sigma_S = S_c^T S_c \) and \( \Sigma_Q = Q^T Q \), and let the regularizer matrix \( \Lambda = (\lambda||S_c^T S_c||_2)I \). \([34]\) uses the Normal Equation to efficiently solve the reconstruction in closed form. Plugging in Eq. (3) yields the following FRN logit formula, where \( \otimes \) is elementwise multiplication (derivation in appendix A.3):

\[
\bar{W} = Q(\Sigma_S + \Lambda)^{-1} S_c^T
\]

\[
z(Q; S_c) = -||\bar{W} S_c - Q||^2 = 2||\Sigma_S + \Lambda||^{-1} \Sigma_S \otimes \Sigma_Q||_1 - tr((\Sigma_S + \Lambda)^{-1} \Sigma_S \Sigma_Q \Sigma_S (\Sigma_S + \Lambda)^{-1})
\]

(4)

Most of the terms in the last line can be discarded. The second term \( tr(\Sigma_Q) \) is constant across all classes and so vanishes in the softmax layer, and in practice the third term is almost exactly proportional to the first. For example, in a 4-layer FRN model trained with shot 4 on meta-iNat, 1000 5-way 5-shot predictions yielded a term-1 to term-2 ratio of 2.026 with standard deviation only .059. Comparing class logits for each query, the standard deviation shrinks further to .009. We therefore use only the first term, with no impact on performance (discussion in appendix A.3.3):

\[
z(Q; S_c) = ||\Sigma_S + \Lambda||^{-1} \Sigma_S \otimes \Sigma_Q||_1
\]

(5)

This formula is a dot product of flattened normalized covariance matrices. The cosine alternative is now clear: replace the FRN normalization with L2 normalization (or Frobenius norm, for matrices):

\[
z(Q; S_c) = ||\Sigma_S||_2 \otimes ||\Sigma_Q||_2^{-1} ||\Sigma_S \otimes \Sigma_Q||_1
\]

(6)

Like other cosine models, the novel Cosine FRN provides consistent improvement in shot robustness while maintaining competitive accuracy. These results are detailed in the following section.

4 Results

We undertake a systematic empirical study across benchmarks, network backbones, learning approaches, and training shot values. Classifiers perform as expected, with FRN outperforming FEAT, and ProtoNet trailing behind (see appendix A.4.2). Cosine alternatives match or reduce shot-sensitivity relative to their Euclidean counterparts in all settings, while providing a notable bump in very-low-shot accuracy (though frequently trading off high-shot accuracy to a lesser degree).

4.1 Experimental Setup

We perform our comparisons on two standard network backbones (Conv-4 and ResNet-12), and four popular few-shot benchmarks (CUB [30, 27], meta-iNat [13, 33], mini-ImageNet [29] and tiered-ImageNet [21]). The backbones match [17, 35, 34] and our hyperparameters largely follow [34]. Dataset, implementation and training details can be found in appendix A.2.

4.2 Measuring Shot Sensitivity

For each dataset and classifier, we train four networks with different shot numbers (4, 8, 16, 32). Each model is then evaluated on a range of test shots (1, 2, 4, 8, 16, 32) to measure sensitivity to test shot variation. All test-time evaluation is 5-way and results are averaged over 10K trials. We do not report 95% confidence intervals for each result, as they are uniformly small (about .20, .17, .14, .12, .11, .10 for 1, 2, 4, 8, 16, 32 test shot respectively, regardless of model or setting). We employ batch folding [33] during training, to ensure that batch sizes do not explode as shot increases. For practical reasons we hold batch size constant at 256, so as shot increases, way decreases. This ensures that all models receive the same amount of data and have the same maximum number of updates.
Figure 2: The process for calculating our shot sensitivity scores and heatmaps. The raw scores (left) are dominated by test shot bias: as shot increases, accuracy increases. The model collection’s mean accuracy is therefore subtracted from each row to produce accuracy offsets, with subtracted averages displayed to the side (middle). Now behavior is dominated by model bias: certain members of the group are always better than others. The model bias from each column is therefore subtracted to produce a corrected heatmap of shot-sensitive behavior (right). Subtracted model biases are displayed above. Weak overfitting to shot (XOR patterning) is now visible in the heatmap.

Decomposing accuracy and isolating shot-sensitive behavior: The above training/evaluation scheme produces a table of raw accuracy scores organized by cross-referenced train and test shot values (Fig. 2 left). Unfortunately, this table of raw scores is dominated by expected and uninteresting behavior. We know that accuracy will increase with testing shot; this alone is not evidence of shot sensitivity. Similarly, we do not expect all models to perform equally well in expectation: some hyperparameter combinations are simply more or less effective across the board. A very low way, high shot training regime may saturate performance too quickly to provide a useful loss signal, while very high way, low shot training may be too difficult for proper convergence. This behavior is also not interesting, or evidence of shot sensitivity. We therefore decompose the raw accuracy scores:

\[
\text{Accuracy} = \text{bias}_{\text{test shot}} + \text{bias}_{\text{model}} + \text{shot sensitive behavior}
\]  

By marginalizing out the first two terms, we can isolate and extract meaningful metrics and representations for shot-sensitivity.

We first marginalize out \(\text{bias}_{\text{test shot}}\) by subtracting the average accuracy over the collection of networks for each test shot (Fig. 2 step 1). This produces a table of model offsets relative to the behavior of the group in expectation (Fig. 2 middle). For a learner entirely insensitive to test shot, we would expect each column of offsets to be a repeated constant: model bias is still present, but networks are equally insensitive to test shot changes. We then marginalize out \(\text{bias}_{\text{model}}\) to isolate shot-sensitive behavior specifically, by subtracting the average difference between each model’s accuracy and the whole collection’s accuracy (Fig. 2 step 2). This produces a final heatmap of isolated shot-sensitive behavior (Fig. 2 right), with all behavior linked to train shot in isolation, or test shot in isolation, removed. Shot overfitting manifests in these heatmaps as XOR-style patterns: increasing positive values on the diagonal indicate good performance when shots match, and decreasing negative values on the anti-diagonal indicate poor performance when shots differ (Fig. 3 contains more examples). The full set of heatmaps is displayed in appendix A.4.3.

Figure 3: An illustrative example of overfitting to shot in a Euclidean (left) and cosine (right) Conv-4 ProtoNet (metainat). Average group accuracy is plotted to the right of each heatmap; model performance relative to its group is plotted above. Both maps display typical XOR-style shot-overfitting, though the cosine model greatly reduces it (shot sensitivity of 14.86 vs 2.13).
Sensitivity score: Shot-overfitting produces large positive diagonal values and negative off-diagonal values, while a perfectly shot-robust approach produces entirely zeros. A straightforward sensitivity metric is then the difference between the largest and smallest offsets in the corrected heatmap. This score provides a rough bound on how much better or worse a given approach is expected to behave as train and test shot vary: a score of 10 indicates that for the given problem setup, a model will always be within 10 points of its best- or worst-case performance.

Note that shot sensitivity is orthogonal to accuracy. This is by design: a few-shot classifier may gain 2 points of accuracy, but if sensitivity also goes up by 2, shot mismatch may wipe those gains entirely. Ideally we would like both high accuracy and low sensitivity; we now investigate how to attain this.

4.3 Reducing Shot Sensitivity with Cosine

Sensitivity scores are compared across settings in Table 2. We make four observations. First, ProtoNet is sensitive to shot, especially when using Conv-4. For these models, scores fall within 7 to 14 points of best- or worst-case behavior, after model bias correction. This is not a promising guarantee, especially on such well-studied benchmarks where state-of-the-art advancements may be two points or smaller. This sensitivity stems from shot overfitting; Fig. 3 provides an example.

Second, the overfitting problem is not unique to ProtoNet. FEAT and FRN display similar shot sensitivity (controlling for network architecture), though Conv-4 ProtoNet is the worst offender. In line with [10], shot sensitive behavior is broadly inherent to metric few-shot learners.

Third, shot-overfitting is generally more dramatic for Conv-4 models than for ResNet-12. Overall patterns of behavior are the same, but the larger ResNet-12 feature extractor, with over 100x the parameters of Conv-4, possesses a similarly large degree of built-in insensitivity to test shot. This makes any improvement on ResNet-12 models particularly noteworthy.

Fourth, and most crucially, cosine models consistently and reliably reduce shot sensitivity (as shown in Fig. 3) compared to Euclidean counterparts, with only a single exception in Table 2, where sensitivity is unchanged. This is regardless of benchmark, architecture, or few-shot learner. In fact, for Conv-4 architectures, the worst cosine model is still more robust to shot than the best Euclidean model (2.43 vs 2.70). On average, cosine models reduce the spread of accuracy scores due to shot by a factor of 4.5 compared to their Euclidean counterparts.

4.4 Comparison to Prior State of the Art

In addition to straightforward Cosine/Euclidean comparisons, we also evaluate two state-of-the-art shot robustness baselines: EST, a projection layer for maximal clustering, and a baseline PCA projection to the same number of dimensions [3]. Interestingly, EST does not always improve upon the shot sensitivity of PCA. This does not directly contradict the findings of [3]: while EST and PCA do have similar shot sensitivity scores, EST has generally higher accuracy (for detailed analysis of PCA/EST/cosine accuracy, see appendix A.4.1), which [3] does not decorrelate from shot robustness. Regardless, while both techniques reduce ProtoNet shot sensitivity, cosine is still superior.

Finally, we note that this comparison is only relevant for ProtoNet, the overall worst performer of our few-shot learners. For FEAT and FRN, cosine models win by default, as the EST down-projection layer is based on class prototypes, and it is not clear what the corresponding layer would be for more sophisticated techniques that a) already exist in a smaller feature space, as feature maps are pooled not flattened, and b) do not use class prototypes directly for classification.
Figure 4: Model group behavior by shot. Each line represents a model group from one dataset, architecture, and choice of few-shot learner. Left: cosine accuracy gains over Euclidean alternatives by test shot. Right: accuracy offsets from group performance due to training shot.

Figure 5: Euclidean models using cosine prediction, and cosine models using Euclidean prediction, tend to mimic their training counterparts rather than their testing counterparts.

5 Analysis and Discussion

Impact on accuracy: Accuracy and shot-robustness are orthogonal, so we might expect a model that provides consistent shot robustness to trade off accuracy to achieve this. Interestingly, this is not the case: compared to Euclidean models, cosine instead trades off high-shot performance for low-shot performance, usually at a beneficial rate. This is made clear in Fig. 4 (left), where accuracy gains as a function of test shot are plotted for cosine models across all settings. 1-shot performance improves for cosine by 3.2 points on average, while sacrificed high-shot performance saturates to 1 point on average (for more detailed comparisons, see appendix A.4.2). This behavior likely stems from the stabilized decision boundaries discussed in Sec. 3.1. Anchoring decision boundaries to the origin is useful in low-shot regimes with high sample noise. When support images are plentiful, it becomes less a useful prior and more an arbitrary constraint.

One issue with accuracy gains as reported in Fig. 4 (left) is that they are aggregate scores from collections of networks. As such, these averages are not realized by any single model. To better examine individual model behavior within collections, we plot model accuracy offsets (model biases displayed above the heatmaps in Fig. 3) in Fig. 4 (right). Average trendlines represent the general impact of training shot on Euclidean and Cosine models. Hyperparameter sensitivity is similar, but where Euclidean models do not consistently prefer high or low shot training, cosine models gather closer around their trend line. The more distinct behavior includes a preference for higher-shot training, as no 4-shot cosine model exceeds the mean accuracy of its group (offsets are negative or zero). This could explain why our 1-shot gains go unobserved in prior work: in 1- or 5-shot settings, gains from cosine are cancelled out by an aversion to low-shot training. Removing these underperforming 4-shot models boosts cosine performance by 1.47 points on average.

Is training with cosine distance necessary? Our findings in Sec. 3.1 and Table 1 suggest that cosine distance stabilizes decision boundaries regardless of training mechanism, and so could possibly robustify Euclidean models at test time without any specialized training. We therefore re-evaluate eight of our Conv-4 Mini-ImageNet models with predictors swapped at test time: Euclidean models evaluate using cosine, and vice versa. Fig. 5 compares the results to the same models evaluated with the correct prediction types. Swapped models mostly track the accuracy of the model with the corresponding training predictor, not the test predictor. This suggests that training with cosine distance is the crucial aspect, and shot overfitting is a learned response to sample noise at train time, as predicted in prior work [3, 10].
Limitations: In our study we expand the scope of evaluation for few-shot classification to more realistic settings. This includes the obvious scenario where train and test shot do not match, but also a higher range of test shot (1-32) compared to most prior work, which typically caps at 5 (even [3] uses only 1/5/10). However, there are orthogonal directions that our benchmark (as well as most others for few-shot classification) does not capture, such as class imbalance. Because we study the impact of shot mismatch, we must assume for practicality that train and test shot always exist, and are set to constant, comparable values. In the real world, however, class distributions are frequently long-tailed [33][11]. The degree to which shot-robust cosine classifiers can handle such problems is an interesting direction for future work.

Second, while cosine models confer impressive robustness to test shot, they are not reliably any more or less robust to the training shot hyperparameter, as shown in Fig. 4(right). In the cases where our cosine models trades off training shot sensitivity for test shot robustness, we view it as a worthwhile trade, as the practitioner has control over the training shot hyperparameter but no control over shot at test time. Therefore overcoming any increased sensitivity is a simple matter of careful validation.

While our analysis primarily concerns shot mismatch, which does not feature heavily in existing few-shot literature, we do not view this as a limitation of scope. Rather, shot mismatch is an inherent feature of real-world problems, and as such, we view our work as broadening the practical scope of deployment for few-shot classifiers.

Societal impact: By facilitating few-shot learning in more realistic settings, our work carries the same potential negative impacts as few-shot learning overall. This includes inadvertently exacerbating problematic trends in data collection, such as overfitting to spurious factors and lack of diversity or minority representation, by allowing practitioners to “settle” for smaller training datasets. However, these are issues inherent to few-shot and deep learning generally, and not specific to our work.

6 Conclusion

We conduct a systematic empirical study of test-time shot variation across datasets, neural architectures, and few-shot learning techniques. A simple, general, and powerfully effective solution to shot overfitting is proposed: replacing Euclidean distance logits with cosine distance reliably stabilizes test time performance without sacrificing accuracy, even on ResNet-12 where starting methods are already quite robust. These cosine models make beneficial tradeoffs between high- and low-shot performance, achieving up to double-digit gains in 1-shot accuracy while sacrificing less than 1 point of high-shot accuracy on average. By addressing a critical weakness in metric few-shot learning, we hope that these cosine methods will serve as general and valuable tools for more realistic settings.

Acknowledgements: This work was funded by NSF IIS-2144117 and the DARPA Learning with Less Labels program (HR001118S0044).

References

[1] A. Banerjee, S. Merugu, I. S. Dhillon, J. Ghosh, and J. Lafferty. Clustering with bregman divergences. *Journal of machine learning research*, 6(10), 2005.

[2] L. Bertinetto, J. F. Henriques, P. H. S. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[3] T. Cao, M. T. Law, and S. Fidler. A theoretical analysis of the number of shots in few-shot learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[4] W. Chen, Y. Liu, Z. Kira, Y. F. Wang, and J. Huang. A closer look at few-shot classification. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[5] Y. Chen, Z. Liu, H. Xu, T. Darrell, and X. Wang. Meta-baseline: Exploring simple meta-learning for few-shot learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9062–9071, 2021.

[6] Y. Chen, X. Wang, Z. Liu, H. Xu, and T. Darrell. A new meta-baseline for few-shot learning, 2020.
[26] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4077–4087, 2017.

[27] L. Tang, D. Wertheimer, and B. Hariharan. Revisiting pose-normalization for fine-grained few-shot recognition. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 14340–14349, 2020.

[28] Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola. Rethinking few-shot image classification: a good embedding is all you need? In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[29] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 3630–3638, 2016.

[30] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[31] Y. Wang, W.-L. Chao, K. Q. Weinberger, and L. van der Maaten. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *arXiv preprint arXiv:1911.04623*, 2019.

[32] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3):1–34, 2020.

[33] D. Wertheimer and B. Hariharan. Few-shot learning with localization in realistic settings. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 6558–6567, 2019.

[34] D. Wertheimer, L. Tang, and B. Hariharan. Few-shot classification with feature map reconstruction networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8012–8021, June 2021.

[35] H. Ye, H. Hu, D. Zhan, and F. Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 8805–8814, 2020.

[36] C. Zhang, Y. Cai, G. Lin, and C. Shen. Deepemd: Few-shot image classification with differentiable earth mover’s distance and structured classifiers. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 12200–12210, 2020.
A Supplementary Materials

A.1 Datasets

A summary of dataset statistics is given in Table 3, with details on each dataset below.

**meta-iNat** [33, 13] is a benchmark of animal species in the wild. This benchmark is particularly difficult, as classes are unbalanced, distinctions are fine-grained, and images are not cropped or centered, and may contain multiple animal instances. We follow the class split proposed by [33]: of 1135 classes with between 50 and 1000 images, one fifth (227) are assigned to evaluation and the rest to training. While the original meta-iNat involves all-way, imbalanced shot classification at test time [33], we instead employ an episodic evaluation scheme as in [34].

**CUB** [30] consists of 11,788 images from 200 bird classes. Following [4, 35, 27, 34], we randomly split categories into 100 classes for training, 50 for validation and 50 for evaluation. Our split is identical to [27, 34]. During training, we use raw images as input and do not crop to the provided bounding boxes.

**mini-ImageNet** [29] is a subset of ImageNet containing 100 classes in total, with 600 examples per class. Following [20], we split categories into 64 classes for training, 16 for validation and 20 for test.

**tiered-ImageNet** [21] is a larger subset of ImageNet with 351-97-160 categories for training-validation-testing, respectively. Tiered-ImageNet ensures larger domain differences between training and evaluation compared to mini-ImageNet. Most works [31, 28, 6, 34] use images from [21] or [17] which have 84×84 resolution, and we do the same.

| Dataset                     | #Images | #Classes | Train/Val/Test |
|-----------------------------|---------|----------|----------------|
| CUB [30, 27]                | 11,788  | 200      | 100/50/50      |
| meta-iNat [13, 33]          | 243,986 | 1135     | 908/0/227      |
| mini-ImageNet (mIN)         | 60,000  | 100      | 64/16/20       |
| tiered-ImageNet (tIN)       | 779,165 | 608      | 351/97/160     |

A.2 Training Details

Our training setup and hyperparameters for the most part match [34] (available under MIT License), on which we base our code. Hyperparameters particular to each method are kept the same as in the original paper, so for example we keep the same attention dropout rates for FEAT as in [35], and reduce to 60 dimensions for all PCA/EST models as in [3]. For practical reasons, we reduce the way (and therefore batch size) by 25% for all ResNet-12 ProtoNet models trained on CUB, including ProtoNet, Cosine ProtoNet, PCA and EST, as well as all ResNet-12 FEAT models. We use the pretrained feature extractor from [34] for mini-ImageNet and tiered-ImageNet ResNet-12 FRNs (standard and cosine). Following [35], we pre-train our ResNet-12 FEAT models linearly before episodic fine-tuning. For mini-ImageNet and tiered-ImageNet we use the pretrained models provided by [35]; for CUB we pre-train our own feature extractor using the FRN pre-training hyperparameters. All other models are trained from scratch.

For Conv-4 FEAT models that do not train with Adam (CUB and mini-ImageNet), we found that the standard hyperparameters from [34] produced divergent behavior during training. We reduced the initial learning rate for these FEAT models by a factor of 10 to recover stable behavior.

Likewise, for ResNet-12 FEAT models, we found that the default hyperparameters from [34] do not work well for fine-tuning. We instead adopt the hyperparameters from [35], but found that this caused convergence to be very slow. We therefore increased the learning rate: train with SGD and momentum 0.9 for 200 epochs (15 for the much larger tiered-ImageNet), with initial learning rate 1e-3 for the feature extractor and 1e-2 for the attention head, dividing learning rate by 10 halfway through.

[3] https://github.com/renmengye/few-shot-ssl-public
[4] https://github.com/kjunlee/MetaOptNet
All models were trained on-premise on a range of NVIDIA GPU types. Each run used a single GPU and finished within hours to days, depending on dataset size, backbone architecture, and training procedure. We will release all the detailed training hyperparameters along with code upon acceptance, such that the audience will be able to reproduce our results.

A.3 Cosine FRN Derivation

A.3.1 Motivating the new FRN regularizer term

The original FRN formulates the reconstruction problem as linear regression: given a matrix of class $c$ support features $S_c$, and a matrix of query features $Q$, find $\bar{W}$ such that $\bar{W}S_c \approx Q$. This is formalized as:

$$\bar{W} = \arg \min_{\bar{W}} \|Q - \bar{W}S_c\|^2 + \frac{\lambda n}{d} ||\bar{W}||^2$$  \hspace{1cm} (8)

where $n$ is the number of support features, $d$ is the latent dimensionality, and $\lambda$ is a learned scaling term. \cite{34} justifies this choice of regularizer parameterization on the grounds that when $n/d$ is high, support features begin to span the feature space and the reconstruction becomes easy, so higher regularization is required. The intuition is sensible, and the formulation comes with the additional benefit of being invariant to shot: expanding the support pool by repeating the existing entries has no effect on the final reconstruction.

Unfortunately, this parameterization leads to other, unintuitive behavior - for example, reconstruction quality is not equivariant to, or even monotonic with, feature scale or dimensionality. Multiplying both the support and query features by a constant factor $\alpha$ should yield the original reconstruction concatenated to itself. When these behaviors are observed, shifting to different feature scales or network widths may require manual hyperparameter adjustment to compensate for the unpredictable shift in reconstruction behavior. Indeed, the authors of \cite{34} found it necessary to manually normalize latent values when implementing FRN on wider networks to prevent divergence during training. Instead, we reformulate the reconstruction problem as:

$$\bar{W} = \arg \min_{\bar{W}} \|Q - \bar{W}S_c\|^2 + (\lambda||S^TS_c||_2)\|\bar{W}\|^2$$  \hspace{1cm} (9)

where $\|\cdot\|_2$ is the Frobenius norm. Eq. (9) provides all of the above desirable invariances and equivariances. Demonstrating this is relatively straightforward.

**Invariance to shot:** expanding the support pool by repeating the individual features should not change the reconstruction. Given the original $n \times d$ support feature matrix $S_c$, let $S'_c$ be the $2n \times d$ feature matrix where every feature in $S_c$ occurs twice. It must then be the case that $S'_cS'_c = 2S^TS_c$. Therefore the closed-form reconstruction from $S'_c$ is given via the Normal Equation and Woodbury Identity by:

$$\bar{W}S'_c = Q(S'^TS'_c + \lambda||S'^TS'_c||_2)^{-1}S'^TS'_c$$

$$= Q(2S^TS_c + 2\lambda||S^TS_c||_2)^{-1} \ast 2S^TS_c$$

$$= Q(S^TS_c + \lambda||S^TS_c||_2)^{-1}S^TS_c$$  \hspace{1cm} (10)

which is the original reconstruction. We refer the reader to \cite{34} for background on the Woodbury normal formulation.

**Equivariance to scale:** scaling all features by $\alpha$ should yield the original reconstruction times $\alpha$. This follows an analogous argument to shot invariance. Let $S'_c = \alpha S_c$ and $Q' = \alpha Q$. It must then be the case that $S'_cS'_c = \alpha S^TS_c$. Therefore the closed-form reconstruction from $S'_c$ is given by:

$$\bar{W}S'_c = Q'(S'^TS'_c + \lambda||S'^TS'_c||_2)^{-1}S'^TS'_c$$

$$= \alpha Q(\alpha^2 S^TS_c + \alpha^2 \lambda||S^TS_c||_2)^{-1}\alpha^2 S^TS_c$$

$$= \alpha Q(S^TS_c + \lambda||S^TS_c||_2)^{-1}S^TS_c$$  \hspace{1cm} (11)

which is the original reconstruction times $\alpha$.

**Equivariance to dimensionality:** expanding the dimensionality by repeating all existing features should yield the original reconstruction, also with repeated features. Let $S'_c = S_c \oplus S_c$ and $Q' = Q \oplus Q$ where $\oplus$ represents concatenation. It must then be the case that $S'_cS'^T = 2S_cS^T$ and $Q'S'^T = 2QS^T$.
Therefore the closed-form reconstruction from $S'_c$ is given by (discarding the Woodbury Identity in this case):

$$W S'_c = Q S'_c^T (S'_c S'_c^T + \lambda ||S'_c S'_c^T||_2)^{-1} S'_c$$

$$= 2Q S'_c^T (2 S_c S_c^T + 2 \lambda ||S'_c S_c^T||_2)^{-1} (S_c \oplus S_c)$$

$$= Q S_c^T (S_c S_c^T + \lambda ||S'_c S_c^T||_2)^{-1} (S_c \oplus S_c)$$

(12)

Distributing over the concatenation in the last line yields the original reconstruction, concatenated with itself. The second line uses the fact that $||X^T X||_2 = ||XX^T||_2$ for all $X$.

Equipped with this more intuitive and stable behavior, our updated FRN regularizer works out of the box in all settings, without requiring any manual hyperparameter tuning based on neural architecture. We also found it to be more robust to pretraining: unlike the original, the updated regularizer is amenable to vanilla linear pretraining (though we do not report these results in the main paper).

### A.3.2 Deriving the expanded FRN prediction formula

We use the same notation as above. Let $\Sigma_S = S'_c^T S_c$ and $\Sigma_Q = Q^T Q$, and let the regularizer matrix $\Lambda = (\lambda ||S'_c S_c^T||_2) I$. The prediction logit for a given class is then:

$$\bar{W} = Q(\Sigma_S + \Lambda)^{-1} S'_c$$

$$z(Q, S_c) = -||\bar{W} S_c - Q||^2$$

$$= -tr((\bar{W} S_c - Q)^T (\bar{W} S_c - Q))$$

$$= 2tr(Q^T \bar{W} S_c) - tr(\bar{W}^T \bar{W} S_c)$$

$$= 2tr(\Sigma_Q (\Sigma_S + \Lambda)^{-1} S_S) - tr(\Sigma_Q) - tr((\Sigma_S + \Lambda)^{-1} \Sigma_S \Sigma_Q S_S (\Sigma_S + \Lambda)^{-1})$$

$$= 2(||(\Sigma_S + \Lambda)^{-1} \Sigma_S \otimes \Sigma_Q||_1 - tr(\Sigma_Q) - tr((\Sigma_S + \Lambda)^{-1} \Sigma_S \Sigma_Q S_S (\Sigma_S + \Lambda)^{-1})$$

$$\approx ||(\Sigma_S + \Lambda)^{-1} \Sigma_S \otimes \Sigma_Q||_1$$

(13)

where $\otimes$ represents elementwise multiplication, which matches the result in Eq. 5 of the main paper. The second-to-last line uses the fact that $tr(AB) = \langle A, B \rangle_F$, the Frobenius inner product. Motivation for removing the extra terms in the final line is discussed in Sec. 3.3. The FRN models in our experiments use the updated regularizer, and train using the last line of the above formula.

### A.3.3 Comparing the new and old FRN models

Table 4 displays example accuracy scores from two Conv-4 FRN models trained on meta-iNat with the old (left) and new (right) prediction and regularizer formulations. Behavior is very similar, with the updated and simplified FRN formula actually showing slight improvements overall.

### A.4 Full Results

#### A.4.1 Detailed EST/PCA/Cosine Comparison

For the ProtoNet setting, we compare our cosine classifiers to both a Euclidean ProtoNet and state-of-the-art shot robustness baselines: EST, a projection layer for maximal clustering, and a baseline PCA projection to the same number of dimensions [3]. Detailed comparisons of accuracy and shot-robustness between these methods are given in rows 2-4 and 10-12 of Tables 4, 5, and 6. Cosine models not only provide better robustness than EST and PCA, they also generally produce higher accuracy scores across the range of test shots. This is demonstrated in Fig. 6 where all EST, PCA, and cosine accuracy gains, averaged over test shots, are plotted against shot sensitivity. We wish to attain low sensitivity and high accuracy (bottom right corner), and cosine models form a clear frontier in this direction.

#### A.4.2 Additional Accuracy Gain Tables

Sensitivity scores and average group accuracy gains for all models on CUB, mini-ImageNet, and meta-iNat/tiered-ImageNet are displayed in Tables 5, 6, and 7 respectively.
Table 4: Accuracy comparison for the old (left) and new (right) FRN formulations on meta-iNat. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

| meta-iNat Conv-4 | FRN (old) | Train Shot | Test Shot |
|-----------------|-----------|------------|-----------|
|                 |           | 4          | 8         | 16        | 32 | Mean |
| 1               | 64.25     | 63.55      | 60.58     | 57.67     | 61.51 |
| 2               | 73.17     | 73.32      | 71.34     | 68.64     | 71.62 |
| 4               | 78.73     | 79.81      | 79.10     | 77.36     | 78.75 |
| 8               | 81.87     | 83.82      | 84.01     | 83.13     | 83.21 |
| 16              | 83.81     | 86.16      | 86.79     | 86.35     | 85.78 |
| 32              | 84.60     | 87.12      | 88.33     | 88.33     | 87.10 |
| Offset          | -0.26     | 0.97       | 0.36      | -1.08     | 2.60 |

| meta-iNat Conv-4 | FRN (new) | Train Shot | Test Shot |
|------------------|-----------|------------|-----------|
|                 |           | 4          | 8         | 16        | 32 | Mean |
| 1               | 65.30     | 63.77      | 61.03     | 58.56     | 62.17 |
| 2               | 74.01     | 73.52      | 71.78     | 69.58     | 72.22 |
| 4               | 79.37     | 80.15      | 79.42     | 77.95     | 79.22 |
| 8               | 82.78     | 84.21      | 83.98     | 83.10     | 83.52 |
| 16              | 84.36     | 86.05      | 86.55     | 86.28     | 85.81 |
| 32              | 85.37     | 87.29      | 87.97     | 88.07     | 87.18 |
| Offset          | 0.18      | 0.81       | 0.10      | -1.10     | 2.30 |

Figure 6: Accuracy/sensitivity tradeoffs for shot-robust methods based on ProtoNet. Each dot corresponds to a single benchmark and neural architecture. Values lower and to the right are desirable (low sensitivity, high accuracy).

Table 5: Sensitivity scores and average group accuracy gains for all models on CUB, illustrating sensitivity and accuracy tradeoffs.

| Network | Model     | SS (↓) | 1 | 2 | 4 | 8 | 16 | 32 |
|---------|-----------|--------|---|---|---|---|----|----|
| Conv-4  | Proto [26]| 9.33   | 50.86 | 65.40 | 74.54 | 80.02 | 82.81 | 84.22 |
|         | PCA [3]   | 4.46   | 57.32 (+0.46) | 68.48 (+3.08) | 75.92 (+1.38) | 80.21 (+0.19) | 82.58 (-0.24) | 83.77 (-0.44) |
|         | EST [3]   | 4.07   | 59.94 (+0.08) | 70.23 (+4.84) | 76.99 (+2.44) | 80.78 (+0.76) | 82.85 (+0.03) | 83.37 (+0.35) |
|         | Cosine Proto | 1.60 | 62.85 (+11.99) | 70.79 (+5.39) | 76.78 (+2.23) | 80.62 (+0.59) | 82.86 (+0.05) | 84.09 (+0.13) |
|         | FEAT [35] | 0.70   | 59.85 | 70.39 | 77.24 | 81.28 | 83.50 | 84.65 |
|         | Cosine FEAT | 0.72 | 63.75 (+3.90) | 70.71 (+1.14) | 77.11 (+0.13) | 80.27 (-1.02) | 82.04 (-1.46) | 82.88 (-1.77) |
|         | FRN [34]  | 6.71   | 70.78 (+3.05) | 78.11 (+0.97) | 82.95 (-0.34) | 85.62 (-1.00) | 87.06 (-1.43) | 87.77 (-1.75) |
| ResNet-12| Proto [26]| 1.14   | 74.44 | 82.70 | 87.15 | 89.36 | 90.45 | 91.01 |
|         | PCA [3]   | 0.89   | 74.53 (+0.09) | 82.62 (+0.07) | 87.14 (-0.01) | 89.27 (+0.10) | 90.45 (-0.06) | 90.99 (-0.01) |
|         | EST [3]   | 0.84   | 74.59 (+1.16) | 82.69 (+0.01) | 87.18 (-0.03) | 89.34 (-0.02) | 90.45 (-0.01) | 90.96 (-0.05) |
|         | Cosine Proto | 0.76 | 76.67 (+2.24) | 83.50 (+0.80) | 87.51 (+0.35) | 89.51 (+0.13) | 90.52 (+0.07) | 91.05 (+0.04) |
|         | FEAT [35] | 0.10   | 72.55 | 82.71 | 87.97 | 90.58 | 91.87 | 92.54 |
|         | Cosine FEAT | 1.27 | 75.43 (+2.88) | 84.43 (+1.72) | 89.38 (+1.41) | 91.85 (+1.27) | 93.03 (+1.16) | 93.64 (+1.10) |
|         | FRN [34]  | 2.00   | 70.78 | 87.47 | 90.86 | 92.72 | 93.67 | 94.26 |
|         | Cosine FRN | 0.21 | 79.96 (-0.82) | 86.12 (-1.36) | 89.41 (-1.45) | 91.04 (-1.69) | 91.93 (-1.84) | 92.18 (-2.08) |
Table 6: Sensitivity scores and average group accuracy gains for all models on mini-ImageNet, illustrating sensitivity and accuracy tradeoffs.

| Network | Model          | SS (↓) | 1  | 2  | 4  | 8  | 16 | 32 |
|---------|----------------|--------|----|----|----|----|----|----|
| Conv-4  | Proto [26]     | 7.32   | 4.70 | 58.34 | 67.07 | 72.98 | 76.39 | 78.33 |
|         | PCA [3]        | 3.76   | 50.60 (+3.66) | 59.86 (+1.52) | 67.54 (+0.47) | 72.93 (+0.66) | 76.02 (+0.36) | 77.85 (+0.48) |
|         | EST [3]        | 3.11   | 51.60 (+4.61) | 60.99 (+2.56) | 68.20 (+1.13) | 73.17 (+0.49) | 76.18 (+0.22) | 77.73 (+0.56) |
|         | Cosine Proto   | 1.39   | 50.38 (+3.86) | 57.80 (+0.54) | 64.43 (+2.64) | 69.85 (+3.12) | 73.61 (+2.78) | 75.88 (+2.45) |
|         | FEAT [35]      | 3.77   | 50.37 | 59.58 | 67.02 | 72.25 | 75.32 | 77.11 |
|         | Cosine FEAT    | 1.31   | 50.33 (+0.03) | 57.52 (+0.07) | 64.43 (+2.64) | 69.04 (+3.21) | 72.20 (+1.12) | 74.05 (+3.06) |
|         | FRN [34]       | 5.09   | 52.11 | 61.16 | 68.75 | 74.20 | 77.57 | 79.53 |
|         | Cosine FRN     | 1.14   | 52.80 (+0.69) | 59.87 (+1.29) | 66.25 (+2.50) | 71.31 (+2.88) | 74.50 (+3.07) | 76.37 (+3.16) |
| ResNet-12| Proto [26]   | 1.21   | 58.69 | 68.49 | 75.30 | 79.64 | 82.09 | 83.48 |
|         | PCA [3]        | 1.43   | 58.75 (+0.06) | 68.41 (+0.08) | 75.17 (+0.14) | 79.54 (+0.10) | 81.97 (+0.12) | 83.32 (+0.16) |
|         | EST [3]        | 1.24   | 58.89 (+0.20) | 68.47 (+0.02) | 75.17 (+0.14) | 79.47 (+0.17) | 81.99 (+0.11) | 83.32 (+0.16) |
|         | Cosine Proto   | 1.09   | 61.65 (+2.96) | 69.14 (+0.65) | 75.22 (+0.09) | 79.60 (+0.04) | 82.20 (+0.11) | 83.67 (+0.19) |
|         | FEAT [35]      | 3.77   | 50.38 | 69.55 | 76.95 | 81.35 | 83.81 | 85.18 |
|         | Cosine FEAT    | 2.43   | 62.37 (+13.33) | 70.60 (+6.65) | 76.65 (+2.48) | 80.53 (+0.67) | 82.69 (+0.07) | 83.92 (+0.39) |
|         | FRN [34]       | 0.30   | 65.82 | 75.18 | 82.09 | 86.85 | 89.62 | 91.12 |
|         | Cosine FRN     | 0.30   | 66.73 (+0.91) | 75.29 (+0.11) | 81.82 (+0.49) | 86.19 (+0.66) | 88.53 (+1.09) | 89.75 (+1.37) |

A.4.3 Full Accuracy and Heatmap Results

Paired accuracy tables and heatmaps are displayed for all tested models below. Models are listed by dataset, then by model architecture, then by learning method. Precise figure and table locations are given in Table 8.

Table 8: Hyperlinked figure numbers for the accuracy tables and sensitivity heatmaps for each tested model configuration.

| Model          | Conv-4 | mIN | iNat | ResNet-12 | mIN | tIN |
|----------------|--------|-----|------|-----------|-----|-----|
| Proto [26]/Cosine Proto | fig. 11 | fig. 19 | fig. 7 | fig. 15 | fig. 23 | fig. 27 |
| PCA/EST [3]    | fig. 12 | fig. 20 | fig. 8 | fig. 16 | fig. 24 | fig. 28 |
| FEAT [35]/Cosine Protot  | fig. 13 | fig. 21 | fig. 9 | fig. 17 | fig. 25 | fig. 29 |
| FRN [34]/Cosine FEAT     | fig. 14 | fig. 22 | fig. 10 | fig. 18 | fig. 26 | fig. 30 |
Figure 7: ProtoNet vs Cosine ProtoNet on meta-iNat, both using a Conv-4 backbone.

| Test Shot | Train Shot | Proto | Mean |
|-----------|------------|-------|------|
| 4         | 8          | 16    | 32   |
| 1         | 57.47      | 53.48 | 45.69| 39.53| 49.04 |
| 2         | 67.52      | 67.47 | 63.20| 57.61| 63.95 |
| 4         | 73.93      | 76.13 | 74.77| 71.86| 74.17 |
| 8         | 77.64      | 80.91 | 81.16| 79.72| 79.86 |
| 16        | 79.53      | 83.36 | 84.34| 83.82| 82.76 |
| 32        | 80.81      | 84.68 | 85.86| 85.86| 84.30 |
| Offset    | 0.47       | 1.99  | 0.16 | -2.61| 14.86 |

| Test Shot | Train Shot | Cosine Proto | Mean |
|-----------|------------|--------------|------|
| 4         | 8          | 16           | 32   |
| 1         | 63.09      | 63.68        | 62.19| 60.53| 62.37 |
| 2         | 70.91      | 71.77        | 70.64| 69.06| 70.60 |
| 4         | 76.37      | 77.66        | 76.96| 75.61| 76.65 |
| 8         | 79.73      | 81.38        | 81.08| 79.93| 80.53 |
| 16        | 81.42      | 83.46        | 83.37| 82.51| 82.69 |
| 32        | 82.66      | 84.58        | 84.48| 83.94| 83.92 |
| Offset    | -0.43      | 0.96         | 0.33 | -0.86| 2.13  |

Table 9: Accuracy comparison for ProtoNet / Cosine ProtoNet on meta-iNat. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Figure 8: PCA vs EST on meta-iNat, both using a Conv-4 backbone.

| Test Shot | Train Shot | PCA | Mean |
|-----------|------------|-----|------|
| 4         | 8          | 16  | 32   |
| 1         | 59.71      | 60.61| 59.19| 53.11| 58.15 |
| 2         | 67.68      | 69.53| 68.36| 61.07| 66.66 |
| 4         | 73.22      | 75.64| 75.05| 66.78| 72.67 |
| 8         | 76.53      | 79.30| 78.94| 70.73| 76.38 |
| 16        | 78.52      | 81.65| 81.41| 73.04| 78.65 |
| 32        | 79.09      | 82.28| 82.23| 73.88| 79.37 |
| Offset    | 0.48       | 2.85 | 2.22 | -5.54 | 2.26  |

| Test Shot | Train Shot | EST | Mean |
|-----------|------------|-----|------|
| 4         | 8          | 16  | 32   |
| 1         | 60.08      | 61.07| 59.75| 58.23| 59.78 |
| 2         | 68.38      | 70.46| 69.90| 68.67| 69.35 |
| 4         | 73.87      | 76.88| 77.12| 76.14| 76.00 |
| 8         | 77.30      | 80.67| 81.06| 80.58| 79.90 |
| 16        | 79.33      | 83.02| 83.66| 83.42| 82.36 |
| 32        | 79.87      | 83.67| 84.52| 84.38| 83.11 |
| Offset    | -1.95      | 0.88 | 0.92 | 0.15 | 3.95  |

Table 10: Accuracy comparison for PCA and EST on meta-iNat. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Figure 9: FEAT vs Cosine FEAT on meta-iNat, both using a Conv-4 backbone.

| meta-iNat Conv-4 | Train Shot | Mean |
|------------------|-------------|------|
| FEAT             | 4           | 58.85|
|                  | 8           | 59.50|
|                  | 16          | 59.50|
|                  | 32          | 59.50|

| Test Shot | Mean |
|-----------|------|
| 1         | 61.13|
| 2         | 70.89|
| 4         | 80.72|
| 8         | 82.58|
| 16        | 83.87|
| 32        | 83.87|

| Offset | 0.14 |
|--------|------|
| 4.16   | 0.31 |

| Test Shot | Mean |
|-----------|------|
| 1         | 64.02|
| 2         | 71.93|
| 4         | 80.72|
| 8         | 82.16|
| 16        | 83.32|
| 32        | 84.48|

| Offset | -0.59 |
|--------|-------|
| 2.43   | 0.31  |

Table 11: Accuracy comparison for FEAT and Cosine FEAT on meta-iNat. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Figure 10: FRN vs Cosine FRN on meta-iNat, both using a Conv-4 backbone.

| meta-iNat Conv-4 | Train Shot | Mean |
|------------------|-------------|------|
| FRN              | 4           | 58.56|
|                  | 8           | 58.56|
|                  | 16          | 58.56|
|                  | 32          | 58.56|

| Test Shot | Mean |
|-----------|------|
| 1         | 65.30|
| 2         | 74.01|
| 4         | 79.37|
| 8         | 82.78|
| 16        | 84.36|
| 32        | 85.37|

| Offset | 0.18 |
|--------|------|
| 5.46   | 0.31 |

| Test Shot | Mean |
|-----------|------|
| 1         | 66.97|
| 2         | 74.28|
| 4         | 79.33|
| 8         | 82.11|
| 16        | 83.52|
| 32        | 84.46|

| Offset | -0.23 |
|--------|-------|
| 2.12   | 0.31  |

Table 12: Accuracy comparison for FRN and Cosine FRN on meta-iNat. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Table 13: Accuracy comparison for ProtoNet / Cosine ProtoNet on CUB. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Table 14: Accuracy comparison for PCA and EST on CUB. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Figure 13: FEAT vs Cosine FEAT on CUB, both using a Conv-4 backbone.

| CUB Conv-4 | FEAT | Train Shot | 4 | 8 | 16 | 32 | Mean |
|------------|------|------------|---|---|----|----|------|
| Test Shot  |      |            |   |   |    |    |      |
| 1          | 61.05| 61.46      | 60.51| 58.89| 59.83|     |
| 2          | 70.89| 72.16      | 71.26| 67.22| 70.38|     |
| 4          | 76.83| 79.17      | 78.46| 74.50| 77.24|     |
| 8          | 80.34| 82.87      | 82.84| 79.07| 81.28|     |
| 16         | 81.19| 85.03      | 85.14| 81.62| 83.50|     |
| 32         | 83.15| 85.96      | 86.57| 82.91| 84.65|     |
| Offset     | -0.41| 1.63       | 1.23| -2.45| 2.70|     |

Table 15: Accuracy comparison for FEAT and Cosine FEAT on CUB. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Figure 14: FRN vs Cosine FRN on CUB, both using a Conv-4 backbone.

| CUB Conv-4 | FRN | Train Shot | 4 | 8 | 16 | 32 | Mean |
|------------|-----|------------|---|---|----|----|------|
| Test Shot  |      |            |   |   |    |    |      |
| 1          | 70.36| 70.86      | 67.17| 62.56| 67.74|     |
| 2          | 77.89| 79.59      | 77.53| 73.57| 77.14|     |
| 4          | 82.43| 84.98      | 84.36| 81.40| 83.29|     |
| 8          | 84.98| 87.49      | 87.94| 86.06| 86.62|     |
| 16         | 86.34| 88.83      | 90.11| 88.67| 88.49|     |
| 32         | 86.98| 89.66      | 91.25| 90.18| 89.49|     |
| Offset     | -0.63| 1.42       | 0.93| -1.72| 6.71|     |

Table 16: Accuracy comparison for FRN and Cosine FRN on CUB. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Figure 15: ProtoNet vs Cosine ProtoNet on CUB, both using a ResNet-12 backbone.

Table 17: Accuracy comparison for ProtoNet / Cosine ProtoNet on CUB. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Figure 16: PCA vs EST on CUB, both using a ResNet-12 backbone.

Table 18: Accuracy comparison for PCA and EST on CUB. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Table 19: Accuracy comparison for FEAT and Cosine FEAT on CUB. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Table 20: Accuracy comparison for FRN and Cosine FRN on CUB. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Figure 19: Proto vs Cosine Proto on mini-ImageNet, both using a Conv-4 backbone.

| Test Shot | 4 | 8 | 16 | 32 | Mean | Offset |
|-----------|---|---|----|----|------|--------|
| 1         | 4.2 | 1.1 | -2.2 | -3.1 | 0.52 | -1.12  |
| 2         | 1.9 | 0.25 | -0.8 | -1.3 | 0.13 | 0.25   |
| 4         | -0.17 | -0.22 | 0.16 | 0.23 | 0.25 | 0.25   |
| 8         | -1.4 | -0.24 | 0.65 | 1   | 0.54 | 0.54   |
| 16        | -2  | -0.45 | 0.99 | 1.4 | 0.2  | -0.74  |
| 32        | -2.5 | -0.42 | 1.2 | 1.8 | 0.16 | -0.16  |

Table 21: Accuracy comparison for ProtoNet / Cosine ProtoNet on mini-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Figure 20: PCA vs EST on mini-ImageNet, both using a Conv-4 backbone.

| Test Shot | 4 | 8 | 16 | 32 | Mean | Offset |
|-----------|---|---|----|----|------|--------|
| 1         | 50.82 | 50.84 | 50.27 | 50.45 | 50.60 | -1.98  |
| 2         | 58.87 | 59.89 | 60.00 | 60.66 | 59.85 | -2.12  |
| 4         | 65.63 | 67.31 | 68.10 | 69.11 | 67.54 | -0.10  |
| 8         | 70.28 | 72.67 | 73.95 | 74.80 | 72.93 | 0.68   |
| 16        | 72.82 | 75.82 | 77.30 | 78.15 | 76.02 | 1.41   |
| 32        | 74.46 | 77.65 | 79.23 | 80.06 | 77.85 | 3.76   |

Table 22: Accuracy comparison for PCA and EST on mini-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Table 23: Accuracy comparison for FEAT and Cosine FEAT on mini-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Table 24: Accuracy comparison for FRN and Cosine FRN on mini-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Figure 23: ProtoNet vs Cosine ProtoNet on mini-ImageNet, both using a ResNet-12 backbone.

Table 25: Accuracy comparison for ProtoNet / Cosine ProtoNet on mini-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Table 26: Accuracy comparison for PCA and EST on mini-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Figure 25: FEAT vs Cosine FEAT on mini-ImageNet, both using a ResNet-12 backbone.

![Figure 25](image1.png)

Table 27: Accuracy comparison for FEAT and Cosine FEAT on mini-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

| mIN ResNet-12 | FEAT | Train Shot | 4  | 8  | 16 | 32 | Mean |
|---------------|------|------------|----|----|----|----|------|
| Test Shot     |      |            | 4  | 8  | 16 | 32 | Mean |
| 1             | 58.55| 58.04      | 56.88| 56.54| 56.34| 57.50| 57.00 |
| 2             | 69.47| 69.26      | 69.00| 68.84| 69.44| 69.94| 69.14 |
| 4             | 77.04| 76.84      | 77.21| 77.24| 77.08| 77.48| 77.08 |
| 8             | 85.52| 85.30      | 82.09| 82.68| 81.79| 82.39| 82.19 |
| 16            | 83.97| 84.11      | 84.99| 85.13| 84.55| 85.29| 85.29 |
| 32            | 85.40| 85.46      | 86.39| 86.63| 85.97| 86.87| 86.87 |
| Offset        | -0.02| -0.17      | 0.09| 0.10| 0.23| 2.13|      |

| mIN ResNet-12 | Cosine FEAT | Train Shot | 4  | 8  | 16 | 32 | Mean |
|---------------|-------------|------------|----|----|----|----|------|
| Test Shot     |             |            | 4  | 8  | 16 | 32 | Mean |
| 1             | 58.42       | 58.39      | 59.36| 59.39| 59.39| 59.39| 59.39 |
| 2             | 69.65       | 69.64      | 69.65| 69.54| 69.54| 69.62| 69.54 |
| 4             | 77.47       | 77.52      | 77.46| 77.35| 77.45| 77.35| 77.45 |
| 8             | 82.42       | 82.32      | 82.39| 82.33| 82.36| 82.36| 82.36 |
| 16            | 85.22       | 85.27      | 85.39| 85.27| 85.29| 85.29| 85.29 |
| 32            | 86.83       | 86.81      | 86.87| 86.95| 86.87| 86.87| 86.87 |
| Offset        | 0.01        | 0.00       | 0.02| 0.02| 0.19|      |      |

Figure 26: FRN vs Cosine FRN on mini-ImageNet, both using a ResNet-12 backbone.

![Figure 26](image2.png)

Table 28: Accuracy comparison for FRN and Cosine FRN on mini-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

| mIN ResNet-12 | FRN | Train Shot | 4  | 8  | 16 | 32 | Mean |
|---------------|-----|------------|----|----|----|----|------|
| Test Shot     |     |            | 4  | 8  | 16 | 32 | Mean |
| 1             | 66.35| 65.91      | 65.49| 65.53| 65.82| 65.82| 65.82 |
| 2             | 75.34| 75.22      | 75.16| 75.00| 75.18| 75.18| 75.18 |
| 4             | 82.17| 82.50      | 82.40| 82.14| 82.30| 82.30| 82.30 |
| 8             | 86.32| 86.93      | 87.09| 87.06| 86.85| 86.85| 86.85 |
| 16            | 88.95| 89.61      | 89.94| 89.69| 89.61| 89.61| 89.61 |
| 32            | 90.30| 91.04      | 91.57| 91.56| 91.12| 91.12| 91.12 |
| Offset        | -0.24| 0.05       | 0.13| 0.06| 1.15|      |      |

| mIN ResNet-12 | Cosine FRN | Train Shot | 4  | 8  | 16 | 32 | Mean |
|---------------|------------|------------|----|----|----|----|------|
| Test Shot     |            |            | 4  | 8  | 16 | 32 | Mean |
| 1             | 65.64       | 66.70      | 67.01| 67.56| 66.73| 66.73| 66.73 |
| 2             | 74.11       | 75.22      | 75.71| 76.10| 75.29| 75.29| 75.29 |
| 4             | 80.65       | 81.64      | 82.35| 82.63| 81.82| 81.82| 81.82 |
| 8             | 85.05       | 85.97      | 86.74| 87.00| 86.19| 86.19| 86.19 |
| 16            | 87.40       | 88.36      | 89.07| 89.27| 88.55| 88.55| 88.55 |
| 32            | 88.63       | 89.65      | 90.53| 90.41| 89.75| 89.75| 89.75 |
| Offset        | -1.14       | -0.13      | 0.49| 0.78| 0.30|      |      |
Figure 27: Proto vs Cosine Proto on tiered-ImageNet, both using a ResNet-12 backbone.

Table 29: Accuracy comparison for ProtoNet / Cosine ProtoNet on tiered-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Figure 28: PCA vs EST on tiered-ImageNet, both using a ResNet-12 backbone.

Table 30: Accuracy comparison for PCA and EST on tiered-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.
Figure 29: FEAT vs Cosine FEAT on tiered-ImageNet, both using a ResNet-12 backbone.

| tIN ResNet-12 | Train Shot | Test Shot |
|---------------|------------|-----------|
| FEAT          | Mean       | Offset    |
| 4             | 66.26      | -0.58     |
| 8             | 66.97      | 0.49      |
| 16            | 68.65      | -0.26     |
| 32            | 69.38      | 0.01      |
| Mean          | 66.90      | 0.63      |
| Offset        | -0.1-0.5    | -0.1-0.5  |

Table 31: Accuracy comparison for FEAT and Cosine FEAT on tiered-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.

Figure 30: FRN vs Cosine FRN on tiered-ImageNet, both using a ResNet-12 backbone.

| tIN ResNet-12 | Train Shot | Test Shot |
|---------------|------------|-----------|
| FRN           | Mean       | Offset    |
| 4             | 69.56      | -0.54     |
| 8             | 70.00      | 0.21      |
| 16            | 70.50      | 0.37      |
| 32            | 70.84      | -0.04     |
| Mean          | 70.07      | 1.07      |
| Offset        | -0.1-0.5    | -0.1-0.5  |

Table 32: Accuracy comparison for FRN and Cosine FRN on tiered-ImageNet. Average group accuracy is displayed in the rightmost column of each table, model offsets are given in the bottom row. Bottom right cell (intersection of Offset and Mean) shows the sensitivity score.