SIMFOREST: AN EFFICIENT PLUG-IN TO BOOST FEW-SHOT LEARNING PERFORMANCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Due to a lack of labeled training data and the subsequent unreliability in empirical risk minimization, few-shot learning algorithms usually suffer from high variance and bias in their predictions. Ensemble, on the other hand, combines predictions from multiple predictors, thus alleviating the aforementioned unreliability. Essentially, we believe that ensemble is a simple yet effective solution to tackle the core problems in few-shot learning; therefore, we develop a plug-in (ensemble) method to boost the performance of trained few-shot models. To maximize the performance of ensemble, we use epoch-training to develop the feature representations used in the plug-in; in contrast, episodic training is used to obtain the feature representations of the original few-shot models. To minimize the extra computation cost induced by ensemble, we adopt a non-deep classifier (e.g: random forest) for the plug-in, which can complete its training within a few seconds. Our method achieves substantial improvements for the few-shot learning, consistently outperforming all the baseline methods.

1 INTRODUCTION

Deep neural networks (DNN) coupled with lots of labeled training data have empowered the salient progress in the image classification field (Kolesnikov et al., 2020). Nevertheless, in many real-world scenarios, large labeled datasets are usually not readily available (Ford, 2018; Zhou et al., 2018). Few-shot learning algorithms perform classifications based on a different set of classes compared to those of the test data set. With few-shot training, models can predict classes of images they are never trained on previously (Wang et al., 2020). This would bring much potential benefit to various real-world scenarios, as the deficiency of labeled data is a common problem in the real world.

So far, few-shot learning remains a challenging problem. In deep learning, reliable empirical risk minimization is generally achieved by having a large number of training examples (Wang et al., 2020; Kolesnikov et al., 2020). In contrast, in few-shot learning, the testing samples are of different classes compared to those used in training; the empirical risk minimization with few labeled test samples thus easily overfits, approximations of the few-shot predictors becoming unreliable with high potential bias and variance (Wang et al., 2020; Dvornik et al., 2019).

In this work, we develop an ensemble to boost the performance of trained few-shot learning models. In particular, we design a lightweight and computationally efficient few-shot classifier (plug-in) using an epoch-trained encoder, which is then ensembled with a model trained through an episodic set-up. The plug-in is named "SimForest" in our work. More specifically, to implement SimForest, we first pretrain a feature encoder using a standard epoch training approach (Krizhevsky et al., 2017), as opposed to a few-shot episodic approach. Then, we use the pretrained feature encoders to produce image features, which are paired up and fed as the input features to random forest classifiers (Breiman, 2001). To ensemble SimForest with other trained models, we simply average the output scores of the former and the latter, taking the averaged result as the final output.

We argue that our ensemble approach is simple yet effective in tackling several major challenges in few-shot learning. In particular, statistical problems arise when a machine learning algorithm is searching a space of hypotheses that is too large for the amount of training data (Dietterich et al., 2002). Secondly, computational problems arise when a learning algorithm tends to get stuck in a local minimums (Dietterich et al., 2002). Both of these are particularly serious challenges to few-shot

learning(Wang et al., 2020). To tackle the pain points, having multiple predictors can control the unreliability in each few-shot predictor by averaging out its individual bias. Besides, a representational problem arises when the algorithm has difficulty to produce representations that are complex enough to resemble those of the target function (Dietterich et al., 2002; Kittler, 2001). A single few-shot predictor falls prey to such a representational problem. To explain, few-shot models generally have to stay simple with relatively few parameters to avoid overfitting, again due to the lack of labeled data (Wang et al., 2020). For example, based on our preliminary experiments, when we try a resnet-18 (He et al., 2016) encoder for few-shot episodic classification, the model overfits and renders poor results. Therefore, in few-shot learning, less complicated encoder architectures such as resnet-12 (He et al., 2016; Ye et al., 2020) are used instead within the few-shot research community. The forced simplicity of model architecture exacerbates the aforementioned representational problem. Fortunately, based on findings in other works (Chen et al., 2019) and our own experiments, unlike episodic training, epoch training is reasonably compatible with complex model architectures. While resnet-18 is infeasible as an encoder for episodic training, it can produce reasonable classification results under epoch training using the same set of training data. Additionally, the different feature representations (epoch training features vs episodic training features) can complement each other to form more complex hypotheses that resemble the target function better, when compared to those formed by a single predictor (Dietterich et al., 2002; Kittler, 2001). With these, the representational problem in few-shot learning can be alleviated.

Our proposed approach has a range of advantages. Above all else, our approach can ease the statistical, computational and representational problems in few-shot learning. In addition, to minimize the extra computation cost of ensemble, we choose a non-deep classifier for SimForest. With pretrained features, the random forest classifier in SimForest can complete the classifier training within a few seconds. As a universally adaptable plug-in for an episodically trained few-shot model, the SimForest classifier can comfortably and consistently achieve a salient performance boost for the latter. We also conduct extensive experiments to demonstrate the obvious improvement in the few-shot learning performance achieved through our method. For the *mini*Imagenet dataset, we can achieve at least 3.79% improvement in the test accuracy scores across different classification tasks. Furthermore, we are able to achieve state-of-the-art result via the proposed approach.

2 Related Works

Few-Shot Learning There are several major types of gradient-based approaches to few-shot learning, including data-based approaches, model-based approaches and algorithm-based approaches (Wang et al., 2020). More relevant to our work, model-based approaches include metric-based approaches which measure the degree of similarity between queried data points and reference data points (Sung et al., 2018; Dvornik et al., 2019). Algorithm-based approaches include meta-learning (Ye et al., 2020; Snell et al., 2017; Ribeiro et al., 2016).

As outlined in (Simon et al., 2020), few-shot learning algorithms generally consist of a feature encoder and a classifier. A few research uses a pretraining strategy to initialize the feature encoder model parameters for few-shot training. By pretraining, we refer to the standard image classification epoch training (Krizhevsky et al., 2017) as opposed to episodic training. While the pretraining strategy is adopted, it is rather overlooked and only used as an auxiliary approach in these works (Ye et al., 2020; Rusu et al., 2018; Qiao et al., 2018). Some works propose that the mainstream meta-learning algorithms in few-shot learning are sub-optimal (Gidaris & Komodakis, 2018; Oreshkin et al., 2018; Qiao et al., 2018), and they demonstrate that a fine-tuned pretrained encoder coupled with even a simple classifier can achieve equivalent or even more superior performance as complex meta-learning models (Chen et al., 2019; Gidaris & Komodakis, 2018; Qiao et al., 2018). (Chen et al., 2019) proposes a baseline for few-shot learning, called *baseline++*. Similarly, authors in (Dhillon et al., 2019) pretrain models via standard image classification approach, and perform transductive fine-tuning by adding in another non-linear layer to the pretrained model for few-shot classification. While the transductive fine-tuning approach is able to achieve good results on few-shot classification, it assumes a certain degree of knowledge about the test dataset, which is usually unknown in real life.

Model Ensemble Ensemble can be achieved using different approaches, including independently constructed ensembles and coordinated constructed ensembles (Dietterich et al., 2002; Kittler,

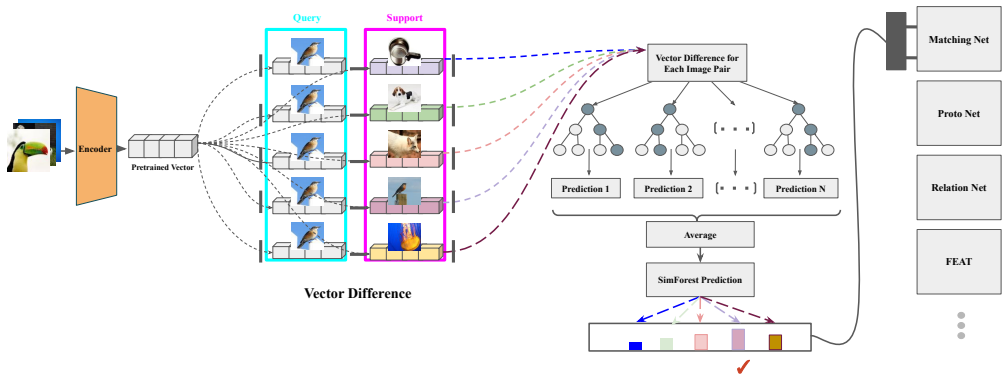

Figure 1: Pipeline of SimForest. Firstly, a pretrained encoder (orange) is obtained from a standard image classification training. Secondly, the pretrained encoder is used to prepare for SimForest training data. After training SimForest with the prepared data, SimForest will be able to make few-shot predictions. Finally, the few-shot predictions will be ensembled with other trained few-shot learning models. The figure is best viewed in colors.

2001). For an independently constructed ensemble, boosting ensemble method combines weak learners to produce a strong learner; random forest is also a type of ensemble that combines the predictions of decision trees through majority voting (Breiman, 2001). The collection of tree structured classifiers can be decision trees (Brijain et al.; Breiman et al., 2017). For coordinated constructed ensembles, Adaboost (Dietterich et al., 2002; Freund et al., 1999) algorithm construct new hypotheses incrementally.

In the intersection field of few-shot learning and model ensemble, (Dvornik et al., 2019) proposes to train multiple few-shot models with a loss function that encourages diversity via KL-divergence and cooperation via cosine similarity. Similar to our work, (Liu et al., 2020) designs a plug-in to be ensembled with other few-shot learning models to achieve enhanced performance. The approach called E^3BM learns and combines an ensemble of epoch-wise Bayes models. Our work fundamentally differs from these ensemble works in few-shot learning in different aspects. Most importantly, we propose a novel and justified scheme that combines epoch training features with episodic training features, while the previous works do not consider such a combination. Based on our experimental results, our work renders arguably more conspicuous improvement for few-shot classification tasks via ensemble.

3 METHOD

We propose the similarity-measuring random forest classifier, or SimForest, which combines DNN-based image embeddings and random forest classifiers for few-shot learning problems. SimForest takes in pairs of image feature vectors and produces their similarity scores as output. The overall architecture of SimForest classifier can be illustrated as figure 1. After training on a small dataset, SimForest is ensembled with other trained few-shot learning models through scores-averaging. A preliminary on few-shot learning problem formulation, along with more details on SimForest and the ensemble are given as follows.

3.1 PRELIMINARY

Few-shot learning employs an episodic strategy to train a few-shot model on a set of classes, and tests the model on a different set of classes. During the training and evaluation, episodic data loaders sub-sample a dataset to form each episode. An episode consists of a support set and a query. For each support set, there are n classes and k data points for each class. Such a support set configuration leads to "n-way-k-shot" classification tasks. Let $\mathcal{T}$ denote such a support set. A query data point x_q is sampled based on $\mathcal{T}$. The query x_q should belong to one of the n classes in the support set. Few-shot learning algorithms that adopt the episodic training and testing strategy need to identify

which class the query x_q belongs to, by comparing it with the samples from each class of the support set (Arnold et al., 2021).

3.2 SIMFOREST

SimForest is able to measure the pairwise similarity score between two input images. To build a SimForest, an image encoder is first prepared through standard deep-learning approaches. Secondly, pairs of images consisting of a support image and a query image are converted to vectors via the pretrained encoder. Their absolute vector differences are used as input for SimForest training, and training labels are whether the image pairs are of the same class.

For stage one, pretraining an encoder, a standard image classification (epoch-training) approach is adopted. For example, few-shot training is usually done on the train split of the *mini*Imagenet dataset; we use the same train split for the pretraining. During the pretraining, the train split images are fed to a DNN-based encoder and a standard classifier, which outputs scores for the image belonging to a particular class. There are 64 training classes in *mini*Imagenet train split, so the standard classifier will produce 64 scores. After the pretraining, only the image feature encoder is retained for further usage.

For stage two, training the SimForest classifier, one needs to first prepare the dataset for SimForest training, before the actual training of SimForest. To prepare the training dataset, pairs of images are converted to vectors using the pretrained encoder. Their absolute vector differences are calculated and fed to the random forest classifier as input. Each absolute vector difference has a label, which is 1 if the image pair is of the same class, and 0 otherwise. Then, one can train the SimForest classifier using the prepared dataset.

To explain the rationale behind the overall design, each value of the feature vectors represents the visual strength of a particular feature in the original picture. Through the set-up, SimForest tends to identify the greatest important discrepancies between pairs of image feature vectors via the absolute vector difference. If the discrepancies are strong enough, the image pair tends to be different-class.

SimForest is formalized as follows. Let f_{θ_1} be a feature encoder (e.g: feature encoders in resnet-18 (He et al., 2016) etc., parametrized by network parameters θ_1), and f_{θ_2} be a standard classifier. Let $D_{train} = \{(x_i, y_i)\}_{i=1}^{N_{train}}$ denote the training data used for both pretraining of encoders and few-shot learning training. Let $(x, y) \in D_{train}$, a prediction result is rendered by $\hat{y} = f_{\theta_2} \circ f_{\theta_1}(x)$. Let θ be a concatenation of θ_1 and θ_2. Minimizing the empirical risk through cross-entropy loss, we can get the optimal parameters θ^* for θ, which is represented by :

$$\theta^* = \arg\min_{\theta} \frac{1}{N_{train}} \sum_{(x,y)\in D_{train}} -\log p_\theta(y|x), \tag{1}$$

Having obtained the feature encoder $f_{\theta_1^*}$, we proceed with SimForest training data preparation. We randomly sample training instances from D_{train}, which are denoted as $S = \{(x_i, y_i)|(x_i, y_i) \in D_{train}\}$. Then, we construct a set of training data for SimForest, $D_{SimForest}$ based on the following criteria:

$$D_{SimForest} = \{(|f_{\theta_1^*}(x_i) - f_{\theta_1^*}(x_j)|, \mathbb{1}(y_i = y_j))|(x_i, y_i), (x_j, y_j) \in S\} \tag{2}$$

We perform training on a random forest classifier with the prepared input $D_{SimForest}$. The random forest classifier performs classification based on votings of multiple decision trees. Each decision tree partitions the input features recursively, based on some threshold, such that samples with the same labels or target values are grouped together. This procedure is also known as "impurity minimization". Let data at node m of the decision tree be denoted as Q_m with N_m samples. Let $\overline{y}_m = \frac{1}{N_m} \sum_{y\in Q_m} y$. The "impurity" $H(\cdot)$ is defined as the following mean squared error function (Buitinck et al., 2013; Brijain et al.; Breiman et al., 2017):

$$H(Q_m) = \frac{1}{N_m} \sum_{y\in Q_m} (y - \overline{y}_m)^2 \tag{3}$$

For each decision tree in the random forest classifier, let ϕ denote the model parameters. Let t_m be each candidate's split threshold value. Let threshold t_m partition a node into two subsets, which are respectively denoted as $Q_m^{left}(\phi) = \{(x, y)|x < t_m, (x, y) \in D_{SimForest}\}$ and $Q_m^{right}(\phi) = Q_m \backslash Q_m^{left}(\phi)$. The training procedure of each decision tree is tantamount to the following equation (Buitinck et al., 2013; Brijain et al.; Breiman et al., 2017):

$$G(Q_m, \phi) = \frac{N_m^{\text{left}}}{N_m} H(Q_m^{\text{left}}(\phi)) + \frac{N_m^{\text{right}}}{N_m} H(Q_m^{\text{right}}(\phi)) \tag{4}$$

$$\phi^* = \underset{\phi}{\operatorname{argmin}}\, G(Q_m, \phi) \tag{5}$$

Let S_{dt} be the set of decision trees in the random forest classifier, where each decision tree model is represented by d_{ϕ^*}. A prediction for input x' is made based on (Breiman, 2001):

$$\hat{y}' = \operatorname{argmax} \frac{1}{|S_{dt}|} \sum_{d_{\phi^*} \in S_{dt}} d_{\phi^*}(x') \tag{6}$$

3.3 SIMFOREST ENSEMBLE

SimForest can be universally ensembled with different types of episodic few-shot algorithms. After preparing the SimForest classifier, one can ensemble the SimForest output scores with the output scores from other trained few-shot learning algorithms. To obtain the output scores, one simply needs to feed the same episodes of a support set and a query to SimForest and another trained few-shot learning model respectively. For multiple-shot scenarios, the average embeddings of the support set is considered during the comparison. For example, for 5-shot tasks, the 5 support images first have their embeddings averaged to become the support embedding for one particular class. Finally, ensembled predictions are obtained as the average scores from SimForest and the other few-shot learning algorithm. This procedure only occurs during the testing phase, and is very resource-efficient. Moreover, through the simple method of the ensemble, we can ensure the convenience and universal adaptability of SimForest ensemble.

SimForest is formalized as follows. For each episode, we have an n-way support set, $\mathcal{T}_{\mathcal{S}_n} = \{(x_{si}, y_{si})\}_{i=1}^n$ and a query sample x_q where x_{si} denotes the prototype of class y_{si}. With trained few-shot learning model γ, the output relation scores are $p_\gamma(x_q|\mathcal{T}_{\mathcal{S}_n}) \in \mathbb{R}^n$. Similarly, we can get the output relation score of SimForest $p_\theta(x_q|\mathcal{T}_{\mathcal{S}_n})$. Our final output score is defined as:

$$\hat{y}_q^{\text{ensemble}} = \frac{1}{2} \left(p_\gamma(x_q|\mathcal{T}_{\mathcal{S}_n}) + p_\theta(x_q|\mathcal{T}_{\mathcal{S}_n}) \right) \tag{7}$$

The entire procedure of SimForest training and ensemble can be described as Alg 1 in appendix.

4 EXPERIMENTS

4.1 EXPERIMENT SET-UP

Datasets: We use two datasets, *mini*Imagenet and cifar-fs (Sun et al., 2019). The *mini*Imagenet dataset is sampled from the imagenet dataset (Russakovsky et al., 2015), consisting of colored images with a 84x84 resolution. There are 100 classes, with 64 of them for training, 16 for validation and 20 for testing. Each of the class contains 600 images. Cifar-fs dataset comprises colored images (of 32x32 resolution) sampled from the cifar-100 dataset (Krizhevsky et al., 2009). We follow the split specified in (Bertinetto et al., 2019), which also has 64 training classes, 16 validation classes and 20 testing classes with 600 samples for each class.

Evaluation Protocols: For evaluation of SimForest, we test the set-up on standard 5-way-1-shot and 5-way-5-shot few-shot classification tasks. We use the Res12 and the Conv4 (Sung et al., 2018; Ye et al., 2020) backbones for the original few-shot learning algorithms, which are also our baselines. In particular, we choose the prototypical network (Snell et al., 2017; Ye et al., 2020), the relation

Table 1: Accuracy for Baselines and SimForest Ensembles (%)

Method	**Backbone**	***mini*Imagenet**		**cifar-fs**	
		1-shot	5-shot	1-shot	5-shot
SimForest(Ours)	Res18†	42.01	60.40	41.66	60.02
ProtoNet (Snell et al., 2017)	Conv4	$50.02_{\pm 0.20}$	$66.89_{\pm 0.17}$	-	-
ProtoNet+*SimForest*	Conv4	$\mathbf{55.78}_{\pm 0.20}$	$\mathbf{74.66}_{\pm 0.16}$	-	-
ProtoNet	Res12	$60.12_{\pm 0.21}$	$73.94_{\pm 0.16}$	$66.73_{\pm 0.18}$	$77.07_{\pm 0.29}$
ProtoNet+*SimForest*	Res12	$\mathbf{65.69}_{\pm 0.20}$	$\mathbf{82.99}_{\pm 0.13}$	$\mathbf{71.34}_{\pm 0.14}$	$\mathbf{84.83}_{\pm 0.11}$
RelationNet (Sung et al., 2018)	Conv4	$49.63_{\pm 0.64}$	$65.16_{\pm 0.65}$	$58.50_{\pm 0.63}$	$74.37_{\pm 0.50}$
RelationNet+*SimForest*	Conv4	$\mathbf{54.18}_{\pm 0.32}$	$\mathbf{71.85}_{\pm 0.44}$	$\mathbf{64.50}_{\pm 0.42}$	$\mathbf{80.67}_{\pm 0.39}$
MatchingNet (Vinyals et al., 2016)	Conv4	$44.00_{\pm 0.74}$	-	$55.87_{\pm 0.66}$	-
MatchingNet+*SimForest*	Conv4	$\mathbf{56.74}_{\pm 0.62}$	-	$\mathbf{67.60}_{\pm 0.63}$	-
FEAT (Ye et al., 2020)	Conv4	$52.21_{\pm 0.19}$	$67.90_{\pm 0.17}$	-	-
FEAT+*SimForest*	Conv4	$\mathbf{57.08}_{\pm 0.19}$	$\mathbf{75.39}_{\pm 0.15}$	-	-
FEAT	Res12	$62.17_{\pm 0.25}$	$78.76_{\pm 0.34}$	$71.76_{\pm 0.27}$	$85.14_{\pm 0.21}$
FEAT+*SimForest*	Res12	$\mathbf{67.14}_{\pm 0.20}$	$\mathbf{84.71}_{\pm 0.13}$	$\mathbf{74.46}_{\pm 0.14}$	$\mathbf{89.23}_{\pm 0.19}$
DeepSet (Ye et al., 2020)	Res12	$60.34_{\pm 0.17}$	$74.82_{\pm 0.33}$	$68.04_{\pm 0.18}$	$77.26_{\pm 0.16}$
DeepSet+*SimForest*	Res12	$\mathbf{66.98}_{\pm 0.30}$	$\mathbf{84.00}_{\pm 0.29}$	$\mathbf{71.35}_{\pm 0.19}$	$\mathbf{82.91}_{\pm 0.11}$
InfoPatch (Liu et al., 2021)	Res12	$67.50_{\pm 0.47}$	$82.10_{\pm 0.31}$	$79.16_{\pm 0.38}$	$89.29_{\pm 0.29}$
InfoPatch+*SimForest*	Res12	$\mathbf{71.29}_{\pm 0.47}$	$\mathbf{88.26}_{\pm 0.33}$	$\mathbf{81.81}_{\pm 0.32}$	$\mathbf{93.16}_{\pm 0.47}$

† The ResNet-18 backbone is frozen and pretrained using epoch-training.

network (Sung et al., 2018), the matching network (Vinyals et al., 2016), FEAT and its variant, DeepSet (Ye et al., 2020) and infoPatch (Liu et al., 2021).

For evaluation of the ensemble performance, We compare the performance of baseline models before and after the ensemble. We report the mean accuracy as well as the 95% confidence interval for both ensemble results and baseline results. For SimForest performance, we only report the mean accuracy.

Implementation Details: For implementation of the SimForest, we first pretrain a resnet-18 encoder (He et al., 2016) on the train split of the *mini*Imagenet dataset, which consists of 64 classes with 600 images for each class. Training hyperparameters are given in the appendix.

After obtaining the image feature encoder, we combine pairs of image feature vectors obtained via the pretrained encoder, and merge two 512-vectors into one 512-vector by computing their absolute vector difference. Each merged 512-vector corresponds to either a positive label representing same-class pairs, and a negative label representing different-class pairs. We sample only 12,800 pairs of positive and negative pairs in total. Among the 12,800 pairs, half are positive and half are negative. Finally, we train a random forest classifier implemented in the scikit-learn library (Buitinck et al., 2013), using the prepared dataset. We set the number of estimators of random forest to 200, the maximum features to 4 and the random seed to 0, 42 or 222.

For implementation of the ensemble, we feed the same testing episodes, each consisting of a support set and a query set, to SimForest and the other trained few-shot model. Both SimForest and the other trained few-shot model produce a relation score for each pairwise combination of the query and support samples. For each episode, we simply combine the relation scores of a SimForest and the relation scores of the other trained few-shot model through averaging. We take the maximum average score as the final prediction. This ensemble is done only during the testing phase.

4.2 Main Results

The first row of table 1 summarizes the performance of SimForest respectively on 5-way-1-shot and 5-way-5-shot tasks, tested on the *mini*Imagenet dataset. The accuracy scores are generally limited. The remaining rows summarize the main results we obtain before and after the ensemble. All the reported accuracy scores of the baseline methods are based on our reproduced experimental results. From the table, we observe that SimForest ensemble can effectively boost the performance

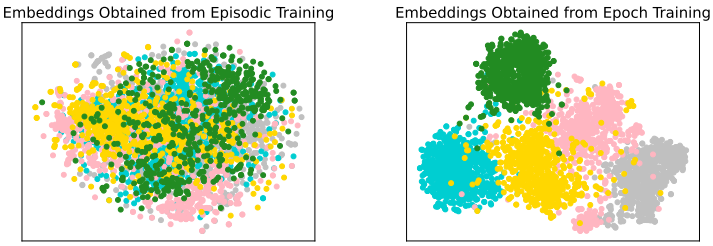

Figure 2: We use TSNE (van der Maaten & Hinton, 2008) projection to visualize feature embeddings based on encoders trained through either a few-shot episodic set-up or standard epoch training set-up. Based on the visualization, features from the few-shot encoder on the left tend to mess together, while the pretrained encoder on the right forms better and more well-separated clusters.

of various few-shot learning algorithms, despite the limited performance of SimForest itself. For example, while SimForest only achieves around 42.01% accuracy score for 5-way-1-shot tasks on *mini*Imagenet, and the original matching network only achieves a 44% accuracy score for the same task, the ensembled model can achieve a 56.74% accuracy score, which is a significant improvement of over 12 percent. Lastly, when ensembled with state-of-the-art (SOTA) models like FEAT or infoPatch, we can achieve new SOTA result. More SOTA comparison is given in table 6 in appendix.

5 Analysis and Ablation Studies

This section summarizes the evaluation and analysis of our approach, including its ablative variants.

5.1 Visualization of Feature Representations

A dissection of SimForest improvement can be found in appendix. We visualize the feature representations of a few-shot encoder and the pretrained encoder. Figure 2 are TSNE projections (van der Maaten & Hinton, 2008) done on the pretrained encoder and the few-shot encoder respectively. In the projections, the same 5 classes are randomly sampled from 64 training classes of the *mini*Imagenet dataset. The few-shot encoder is a randomly initialized Conv4 encoder trained via the relation network algorithms on 5-way-1-shot tasks, until convergence. Based on the visualization, the pretrained encoder forms better clusters, when compared to those formed by the episodic training. We also visualize the attention map for the two different encoders, which is illustrated by figure 3. The attention maps illustrate that the pretrained encoders have much more precise attention than the encoders of the relation network. The attention maps here are different from traditional attention maps, and is named *mutual attention* map in this work. Instead of highlighting regions of interest in a traditional classification setting, the *mutual attention* map highlights regions of interest during pairwise images comparisons. In other words, the attention map highlights the regions the classifiers focus on, in order to tell whether the two image belong to the same class. A standard image classification model produces a set of confidence scores for a single input image, based on which the weights activation mapping is calculated (Selvaraju et al., 2019). In contrast, a few-shot model can take in a pair of images, producing only one similarity score for each image pair. We calculate the weights activation mapping for a pair of images based on the the single similarity output score. Please see our code for more details on the implementation of "*mutual attention* map", which is modified based on (Gildenblat & contributors, 2021; Selvaraju et al., 2019).

Both TSNE and *mutual attention* map visualizations demonstrate that our pretrained encoder can produce more complex feature representations which are visually disparate from the few-shot encoder. This may be explained by the much more complex model architecture of the pretrained encoder when compared to the few-shot encoder, in addition to the difference in training approach. As we mention in the introduction, episodic training cannot afford to have complex model configurations with excessive model parameters due to its susceptiblity to model overfitting. However, preparing the complex encoder based on epoch training and then ensembling it with a few shot encoder can bypass the barrier.

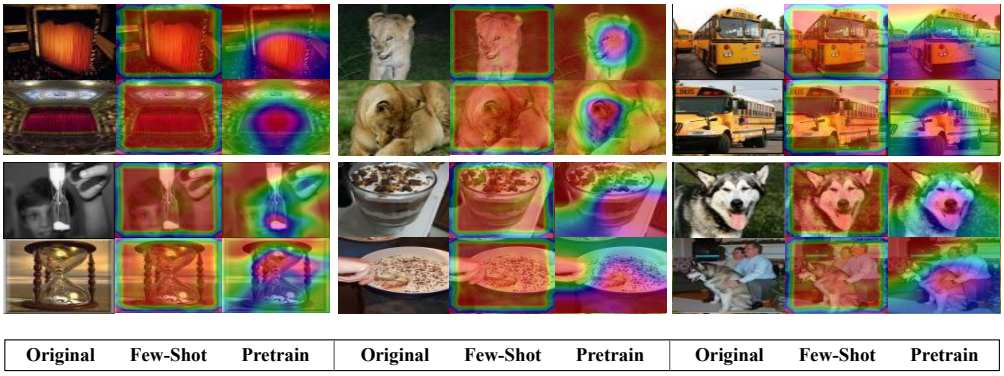

Figure 3: We use *mutual attention* maps to visualize the *mini*Imagenet test set image pairs. There are six groups of *mutual attention* maps. Each group comprises, from left to right: (i) the original image pair; (ii) *mutual attention* maps based on relation network encoders and (iii) *mutual attention* maps based on SimForest encoders, or the pretrained encoders. Please see the main text for more details on the term "*mutual attention*".

5.2 Model Interpetability

We use Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016) to qualitatively understand the inner mechanisms of the model. LIME allows us to understand the effects of superpixels in test images on the prediction of class similarity. We choose LIME as it can be applied to SimForest, which is a mixture of gradient and non-gradient based models. Furthermore, recent studies find theoretical guarantees and connections between LIME and integrated gradient methods (Garreau & Mardaoui, 2021). We randomly sample two classes (class A and B) and select two images from one class (images A and B), and one image from another class (image C). We then create 2 image pairs, (i) image A and B, (ii) image A and C. We designate the randomly-selected image A as the input image for LIME's perturbations. Using a trained SimForest on *mini*Imagenet, we run LIME on the model while feeding the model with pairs (i) and (ii). We find qualitative results that validate our hypothesis that SimForest is able to identify image features that contribute to similarity scoring. Figure 4 in the appendix shows samples of our qualitative results.

5.3 Computation Efficiency

In our experiment, we use 16-bit floating precision representation for SimForest training, as opposed to 32-bit full precision representation. We use the lower-precision as we observe that the performance of SimForest is similar for the two representations. The 32-bit SimForest results can be found in appendix 4.

The time complexity of random forest algorithms is $O(N \times log(N) \times k)$, where N is the size of the dataset and k is the number of features for each data point (Breiman, 2001). The overall training time for SimForest is summarized in table 2. Our CPU model is Intel(R) Xeon(R) Silver 4310 CPU @ 2.10GHz. Overall, the training time of SimForest is almost negligible compared to other few-shot algorithm training.

5.4 What if the random forest classifier is replaced?

In our work, we also explore a fully-DNN approach. By replacing random forest classifier with a 3-layer fully-connected DNN classifier, the "DNN-based SimForest" can achieve even better performance as suggested in table 2 and table 4 in appendix. We also try replacing the absolute vector difference set-up with a concatenation set-up. However, these methods take much longer and more data to train. Therefore, we decide not to focus on the design in this work.

Table 2: Accuracy for SimForest and its Variants

SimForest Classifier (Δ)	**1-shot Accuracy(%)**	**5-shot Accuracy(%)**	**Training Time(s)**
Random Forest (Breiman, 2001)	42.01	60.40	7.62
Extra Trees (Geurts et al., 2006)	41.89	60.87	3.86
Ada-Boosting (Friedman, 2001)	42.99	59.29	2.98

The key component of SimForest which leads to the ensemble performance improvement is the feature encoder as opposed to the classifier. Other non-deep classifiers can also lead to similar performance. These classifiers can be gradient boosting classifier (Friedman, 2002), extra tree classifier (Geurts et al., 2006), ada-boosting classifier (Friedman, 2001), k-nearest-neighbours classifier (Goldberger et al., 2004), support vector machine classifier (Wu et al., 2003) and so on. We only report random forest classifier, extra tree classifier and ada-boosting classifier results in our work, due to their simplicity of hyperparameters involved.

6 Conclusions

To conclude, this paper proposes an algorithm called SimForest, which can be either used on its own, or conveniently ensembled with various trained few-shot learning models during the testing phase. SimForest makes use of pretrained image features, strengthening the original few-shot learning prediction scores with alternative predictions, easing the statistical, computational and representational problems in a single few-shot predictor. Aside from achieving substantial boost for few-shot learning performance, the SimForest module can train itself within a few seconds on standard CPUs, thus emerging as an efficient and practical "plug-in" for almost all types of episodically trained few-shot learning models.

References

Sébastien Arnold, Guneet Dhillon, Avinash Ravichandran, and Stefano Soatto. Uniform sampling over episode difficulty. *Advances in Neural Information Processing Systems*, 34, 2021.

Luca Bertinetto, Joao F. Henriques, Philip Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=HyxnZh0ct7`.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

Mr. Brijain, R Patel, Mr. Kushik, and K Rana. A survey on decision tree algorithm for classification.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. *arXiv preprint arXiv:1909.02729*, 2019.

Thomas G Dietterich et al. Ensemble learning. *The handbook of brain theory and neural networks*, 2(1):110–125, 2002.

Nikita Dvornik, Cordelia Schmid, and Julien Mairal. Diversity with cooperation: Ensemble methods for few-shot classification. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 3723–3731, 2019.

Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004.

Martin Ford. *Architects of Intelligence: The truth about AI from the people building it*. Packt Publishing Ltd, 2018.

Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4): 367–378, 2002.

Damien Garreau and Dina Mardaoui. What does lime really see in images? *International Conference on Machine Learning*, 2021.

Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4367–4375, 2018.

Jacob Gildenblat and contributors. Pytorch library for cam methods. `https://github.com/jacobgil/pytorch-grad-cam`, 2021.

Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. *Advances in neural information processing systems*, 17, 2004.

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

Josef Kittler. *Multiple Classifier Systems: Second International Workshop, MCS 2001 Cambridge, UK, July 2-4, 2001 Proceedings*, volume 2. Springer Science & Business Media, 2001.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pp. 491–507. Springer, 2020.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10657–10665, 2019.

Chen Liu, Yanwei Fu, Chengming Xu, Siqian Yang, Jilin Li, Chengjie Wang, and Li Zhang. Learning a few-shot embedding model with contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8635–8643, 2021.

Yaoyao Liu, Bernt Schiele, and Qianru Sun. An ensemble of epoch-wise empirical bayes for few-shot learning. In *European Conference on Computer Vision*, pp. 404–421. Springer, 2020.

Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. *Advances in neural information processing systems*, 31, 2018.

Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7229–7238, 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, oct 2019. doi: 10.1007/s11263-019-01228-7. URL https://doi.org/10.1007%2Fs11263-019-01228-7.

Christian Simon, Piotr Koniusz, Richard Nock, and Mehrtash Harandi. Adaptive subspaces for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4136–4145, 2020.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30, 2017.

Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 403–412, 2019.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision*, pp. 266–282. Springer, 2020.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning, 2016. URL https://arxiv.org/abs/1606.04080.

Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020.

Ting-Fan Wu, Chih-Jen Lin, and Ruby Weng. Probability estimates for multi-class classification by pairwise coupling. *Advances in Neural Information Processing Systems*, 16, 2003.

Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8808–8817, 2020.

Sung Whan Yoon, Jun Seo, and Jaekyun Moon. Tapnet: Neural network augmented with task-adaptive projection for few-shot learning. In *International Conference on Machine Learning*, pp. 7115–7123. PMLR, 2019.

Chi Zhang, Yujun Cai, Guosheng Lin, and Chunhua Shen. Deepemd: Few-shot image classification with differentiable earth mover's distance and structured classifiers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12203–12213, 2020.

Donghao Zhou, Zheng Yan, Yulong Fu, and Zhen Yao. A survey on network data collection. *Journal of Network and Computer Applications*, 116:9–23, 2018.

A Appendix

A.1 Dissect SimForest Improvement

The significant performance boost achieved through the ensemble may be explained by a decomposition of the correct/wrong predictions made by SimForest and other algorithms respectively. Intuitively, there is much "orthogonality" between SimForest and the other trained few-shot model, whose predictions complement each other's. Table 3 summarizes such a decomposition. The 4th column represents that both SimForest and original method have correct predictions; 5th represents that only SimForest has correct predictions; 6th represents that both SimForest and original method have wrong predictions; 7th represents that only the original method has correct predictions; 8th represents predictions corrected by SimForest ensemble, and 9th represents predictions wrongly changed by SimForest ensemble. According to the decomposition, the number of predictions corrected by SimForest ensemble are consistently higher than those wrongly corrected by SimForest, thus explaining the boost in performance.

Table 3: Ensemble Statistics (%)

Method	**Backbone**	**Task**	**SimForest ✓ Original ✓**	**SimForest ✓ Original ×**	**SimForest × Original ×**	**SimForest × Original✓**	**SimForest Correction**	**SimForest Mistake**
FEAT (Ye et al., 2020)	Conv4	1-shot	21.86	20.31	27.65	30.18	7.35	2.39
	Conv4	5-shot	41.33	18.91	12.6	27.16	8.68	1.14
	Res12	1-shot	26.92	15.23	20.88	36.98	5.56	1.93
	Res12	5-shot	46.96	12.98	8.61	31.45	6.80	0.91
ProtoNet (Snell et al., 2017)	Conv4	1-shot	23.40	19.52	26.93	30.25	8.99	1.68
	Conv4	5-shot	40.50	19.48	13.25	26.78	9.08	1.23
	Res12	1-shot	25.01	17.12	22.39	35.48	8.98	3.67
	Res12	5-shot	44.63	15.29	10.59	29.49	11.02	2.22
RelationNet (Sung et al., 2018)	Conv4	1-shot	24.62	17.03	23.47	34.87	4.25	1.23
	Conv4	5-shot	38.08	21.65	14.69	25.57	6.75	0.95

A.2 Why not include SimForest in the training phase?

Based on our experiment, including SimForest does not help improve the performance of the ensemble. As suggested by table 5 in appendix, if we are to ensemble SimForest with the original few-shot learning algorithm during the training phase, the performance degrades by around 2%, which is suggested by the *train+test* column in the table. Furthermore, the computation cost also increases. The experiments are conducted on 5-way-1-shot *mini*Imagenet classification tasks.

A.3 Hyperparameters for Pretraining of Encoders

For pretraining of the feature encoders, we use a randomly initialized resnet-18 (He et al., 2016) model, whose last fully-connected layer is changed to produce 64 scores as opposed to 1000 scores. For the input data, we only apply normalization without any augmentation. The optimizer we use is a stochastic gradient descent (SGD) optimizer. The momentum is 0.9. The weight decay rate is 0.0001. The initial learning rate is 0.1. We train the model for 100 epochs, and the learning rate is reduced by a factor of 0.1 during the 30th, 60th and 90th epoch.

We do not directly finetune a pretrained resnet-18 model because the model is pretrained on the ImageNet (Russakovsky et al., 2015) dataset. As the *mini*Imagenet, including its test split, is a subset of the ImageNet dataset, directly using a pretrained resnet-18 model will lead to an unfair assessment. Meanwhile, we also do not want to involve extra training data.

Algorithm 1: SimForest Emsemble

Data: D_{train}, D_{test}, $M_{trained}$
Result: $FinalPredictions$
Function *PrepareEncoder(D_{train})*
    $Enc \leftarrow RandomlyInitializedEncoder$;
    $Class \leftarrow RandomlyInitializedClassifier$;
    train Enc and $Class$ *together, via gradient descent, on* D_{train};
    return Enc;
end
Function *PrepareData(D_{train}, Enc)*
    $TrainForestX \leftarrow emptyList$;
    $TrainForestY \leftarrow emptyList$;
    $iterationSize \leftarrow constant$;
    $sampleSize \leftarrow constant$;
    for $i \leftarrow 1$ **to** $iterationSize$ **do**
        for $classIndex \leftarrow 1$ **to** *GetClassSize(D_{train})* **do**
            $PostivePairs \leftarrow SamplePositivePairs(classIndex)$;
            $Difference =$
             $Enc(PostivePairs.get(image1)) - Enc(PostivePairs.get(image2))$;
            $TrainForestX.append(Difference.absolute())$;
            $TrainForesY.append(1.repeat(sampleSize))$; /* Sample positive pairs */
            $NegativePairs \leftarrow SampleNegativePairs(classIndex)$;
            $Difference =$
             $Enc(NegativePairs.get(image1)) - Enc(NegativePairs.get(image2))$;
            $TrainForestX.append(Difference.absolute())$;
            $TrainForesY.append(0.repeat(sampleSize))$; /* Sample negative pairs */
        end
    end
    return $concat(TrainForestX, TrainForestY)$;
end
Function *Ensemble($D_{test}, TrainedSimForest, M_{trained}$)*
    $SimForestPreds \leftarrow Test(D_{test}, TrainedSimForest)$;
    $OtherPreds \leftarrow Test(M_{trained}, TrainedSimForest)$; /* $M_{trained}$ represents a trained few-shot learning model (e.g: Matching network, FEAT...) */
    return $average(SimForestPreds, OtherPreds)$;
end
$Enc \leftarrow PreapreEncoder(D_{train})$;
$D_{train-forest} \leftarrow PrepareData(D_{train}, Enc)$;
$TrainedSimForest \leftarrow Train(D_{train-forest}, Enc)$;
$FinalPredictions \leftarrow Ensemble(D_{test}, TrainedSimForest, M_{trained})$;
Return $FinalPredictions$;

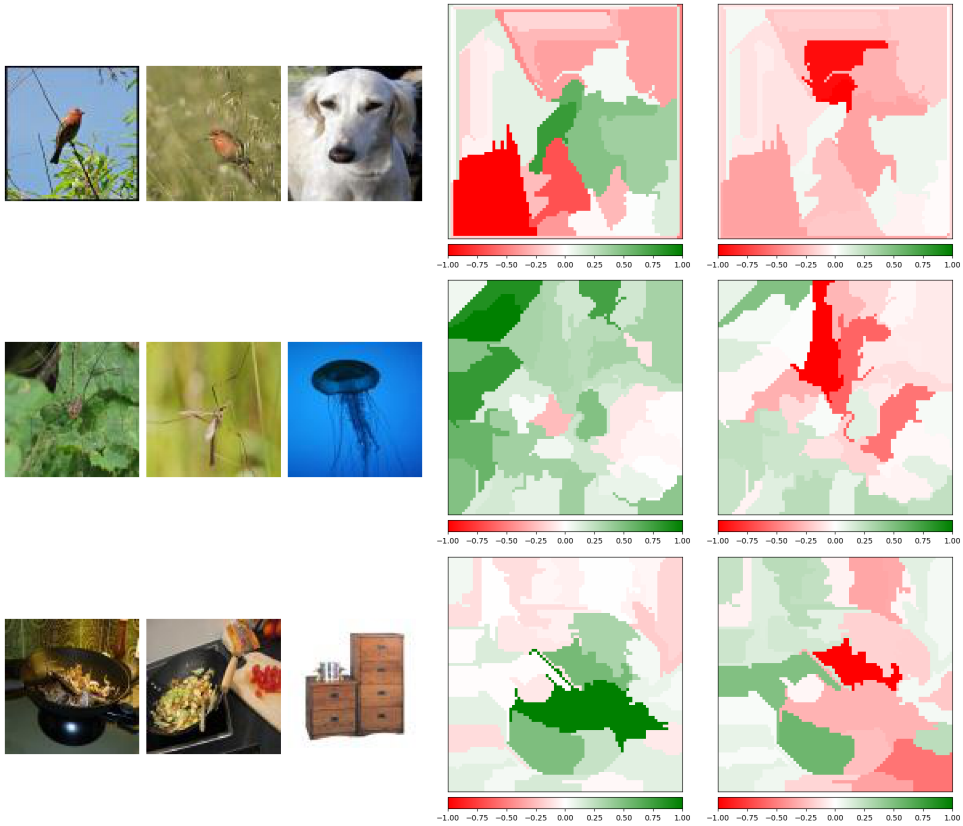

Figure 4: *LIME* interpretability maps for sampled *mini*Imagenet images. We choose image pairs from the same class at random, and image pairs from different classes at random. We use the image on the left most side as the input image to LIME, the middle image as the image from the same class, and the image on the right as the image from a different class. LIME scores are based on superpixels identified via unsupervised image segmentation using Felzenszwalb's graph-based image segmentation (Felzenszwalb & Huttenlocher, 2004). Green areas indicate regions that contribute to a prediction of same class, while red regions indicate regions that contribute to a prediction of different class. Classes shown here are (from top to bottom: [house finch, Saluki], [harvestman, jellyfish], [wok, file cabinet].

Table 4: Accuracy for SimForest and its Variants (full-precision) (%)

Simforest Classifier (Δ)	**1-shot Accuracy**	**5-shot Accuracy**
Random Forest (Breiman, 2001)	42.17	60.40
Extra Trees (Geurts et al., 2006)	41.89	61.49
Ada-Boosting (Friedman, 2001)	42.99	61.49
DNN	47.41	-
DNN-concat	49.52	-

Table 5: 5-way-1-shot Ensemble Accuracy(%)

Method	**Backbone**	**Train+Test**	**Test-Only**
Relation Net (Sung et al., 2018)	Conv4	52.43	54.18
Matching Net (Vinyals et al., 2016)	Conv4	55.20	56.74
FEAT (Ye et al., 2020)	Res12	65.63	67.14
Deep Set (Ye et al., 2020)	Res12	64.62	66.98

Table 6: Reported Few-shot Learning SOTA Results (Inducitve setting with Res12 backbones) (%)

Method	**1-shot Accuracy**	**5-shot Accuracy**
TapNet (Yoon et al., 2019)	$61.65_{\pm 0.15}$	$76.36_{\pm 0.10}$
MetaOptNet (Lee et al., 2019)	$62.64_{\pm 0.61}$	$78.63_{\pm 0.46}$
FEAT (Ye et al., 2020)	$66.78_{\pm 0.20}$	$82.05_{\pm 0.14}$
DeepEMD (Zhang et al., 2020)	$65.91_{\pm 0.82}$	$82.41_{\pm 0.56}$
Rethink-Distill (Tian et al., 2020)	$64.82_{\pm 0.60}$	$82.14_{\pm 0.43}$
infoPatch (Liu et al., 2021)	$67.67_{\pm 0.45}$	$82.44_{\pm 0.31}$
RobustDistill (Dvornik et al., 2019)	$63.06_{\pm 0.61}$	$80.63_{\pm 0.42}$
MLT (Sun et al., 2019)	63.40	80.10
E^3BM + MLT (Liu et al., 2020)	64.30	81.00
FEAT+SimForest	$\mathbf{67.14}_{\pm 0.20}$	$\mathbf{84.71}_{\pm 0.13}$
infoPatch+SimForest	$\mathbf{71.29}_{\pm 0.47}$	$\mathbf{88.26}_{\pm 0.33}$