Low-Shot Learning with Imprinted Weights

Hang Qi* UCLA
Matthew Brown Google
David G. Lowe Google

Abstract

Human vision is able to immediately recognize novel visual categories after seeing just one or a few training examples. We describe how to add a similar capability to ConvNet classifiers by directly setting the final layer weights from novel training examples during low-shot learning. We call this process weight imprinting *as it directly sets weights for a new category based on an appropriately scaled copy of the embedding layer activations for that training example. The imprinting process provides a valuable complement to training with stochastic gradient descent, as it provides immediate good classification performance and an initialization for any further fine-tuning in the future. We show how this imprinting process is related to proxy-based embeddings. However, it differs in that only a single imprinted weight vector is learned for each novel category, rather than relying on a nearest-neighbor distance to training instances as typically used with embedding methods. Our experiments show that using averaging of imprinted weights provides better generalization than using nearest-neighbor instance embeddings.*

1. Introduction

Human vision can immediately recognize new categories after a person is shown just one or a few examples [10, 8]. For instance, humans can recognize a new face from a photo of an unknown person and new objects or fine-grained categories from a few examples by implicitly drawing connections from previously acquired knowledge. Although deep neural networks trained on millions of images have in some cases exceeded human performance in large-scale image recognition [15], under an open-world setting with emerging new categories it remains a challenging problem how to continuously expand the capability of an intelligent agent from limited new samples, also known as *low-shot learning*.

Embedding methods [25] have a natural representation for low-shot learning, as new categories can be added simply by pushing data examples through the network and performing a nearest neighbor algorithm on the result [16]. It has long been realized in the semantic embedding literature that the activations of the penultimate layer of a ConvNet classifier can also be thought of as an embedding vector, which is a connection we further develop in this paper. ConvNets are the preferred solution for achieving the highest classification performance, and the softmax cross-entropy loss is faster to train than the objectives typically used in embedding methods, such as triplet loss.

In this paper, we attempt to combine the best properties of ConvNet classifiers[1] with embedding approaches for solving the low-shot learning problem. Inspired by the use of embeddings as proxies [11] or agents [24] for individual object classes, we argue that embedding vectors can be effectively compared to weights in the last linear layer of ConvNet classifiers. Our approach, called *imprinting*, is to compute these activations from a training image for a new object category and use an appropriately scaled version of these activation values as the final layer weights for the new category while leaving the weights of existing categories unchanged. This is extended to multiple training examples by incrementally averaging the activation vectors computed from the new training images, which our experiments find to outperform nearest-neighbor classification as used with embedding approaches.

We consider a low-shot learning scenario where a learner initially trained on *base classes* with abundant samples is then exposed to previously unseen *novel classes* with a limited amount of training data for each category [5]. The goal is to have a learner that performs well on the combined set of classes. This setup aligns with human recognition which continuously learns new concepts during a lifetime.

Existing approaches exhibit characteristics that render them infeasible for resource-limited environments such as mobile devices and robots. For example, training a deep ConvNet classifier with stochastic gradient descent requires an extensive fine-tuning process that cycles through all prior training data together with examples from additional categories [5]. Alternatively, semantic embedding methods

*The majority of the work was done while interning at Google.

[1] In this paper we use the term "ConvNet classifiers" to refer to convolutional neural networks trained with the softmax cross-entropy loss for classification tasks.

such as [12, 16, 18, 20] can immediately remember new examples and use them for recognition without retraining. However, semantic embeddings are difficult to train due to the computationally expensive hard-negative mining step and these methods require storing all the embedding vectors of encountered examples at test time for nearest neighbor retrieval or classification.

We demonstrate that the imprinted weights enable instant learning in low-shot object recognition with a single new example. Moreover, since the resulting model after imprinting remains in the same parametric form as ConvNets, fine-tuning via backpropagation can be applied when more training samples are available and when iterative optimization is affordable. Experiments show that the imprinted weights provide a better starting point than the usual random initialization for fine-tuning all network weights and result in better final classification results for low-shot categories. Our imprinting method provides a potential model for immediate recognition in biological vision as well as a useful approach for on-line updates for novel training data, as in a mobile device or robot.

The remainder of the paper is organized as follows. In Section 2, we discuss related work. Section 3 discusses the connections between embedding training and classification. Section 4 describes our approach. Then we provide implementation details and evaluate our approach with experiments in Sections 5 and 6. Section 7 concludes the paper.

2. Related Work

Metric Learning. Metric learning has been successfully used to recognize faces of new identities [2, 16] and fine-grained objects [11, 12, 14, 19, 20]. The idea is to learn a mapping from inputs to vectors in an embedding space where the inputs of the same identity or category are closer than those of different identities or categories. Once the mapping is learned, at test time a nearest neighbors method can be used for retrieval and classification for new categories that are unseen during training.

Contrastive loss [2] minimizes the distances between inputs with the same label while keeping the distances between inputs with different labels far apart. Rather than minimizing absolute distances, recent approaches formulate objectives focusing on relative distances. FaceNet [16] optimizes a triplet loss and develops an online negative mining strategy to form triplets within a mini-batch. Instead of penalizing violating instance-based triplets independently, alternative loss functions regulate the global structure of the embedding space. Magnet loss [14] optimizes the distribution of different classes by clustering the examples using k-means and representing classes with centroids. Lifted structured loss [12] incorporates all pair-wise relations within a mini-batch instead of forming triplets. The N-pair loss [18] requires each batch to have examples from N categories for improved computational efficiency. All these methods require some online or offline batch generation step to form informative batches to speed up training. Structured clustering loss [19] optimizes a clustering quality metric globally in the embeddings space.

The Proxy-NCA loss [11] demonstrates faster convergence without requiring batch generation by assigning trainable proxies to each category, which we will describe in more detail in Section 3. NormFace [24] explores a similar idea with all feature vectors normalized. The embedding can generalize to unseen categories, however the nearest neighbor model needs to store the embeddings of all reference points during testing. In our work, we retain the parametric form of ConvNet models and demonstrate that semantic embeddings can be used to imprint weights in the final layer. As a result, our approach has the same convergence advantages as [11] and [24] during training, yet does not require storing embeddings for each training example or using nearest-neighbor search during inference.

One-shot and Low-shot Learning. One-shot or low-shot learning aims at training models with only one or a few training examples. The siamese network [7] uses two network streams to extract features from a pair of images and regress the inputs to a similarity score between two feature vectors. Matching networks [22] learn a neural network that maps a small support set of images from unseen categories and an unlabeled example to its label. Prototypical networks [17] use the mean embeddings of new examples as prototypes, but the embedding space is local with respect to the support classes due to the episodic scheme. These works formulate the low-shot learning problem as classifying an image among a number of unseen classes characterized by the support images; a query image and a support set must be provided together every time at inference. However, this evaluation setup does not align with human vision and many real-world applications where a learner grows its capability as it encounters more categories and training samples. In contrast, we consider an alternative setup similar to [5] which focuses on the overall performance of the learner on a combined set of categories including base classes represented by abundant examples together with novel low-shot classes. Hariharan and Girshick [5] train a multi-layer perceptron to generate additional feature vectors from a single example by drawing an analogy with seen examples. Their method retrains the last linear classifier at the low-shot training stage, whereas our approach allows instant performance gain on novel classes without retraining. More similar to our work is [13], which trains parameter predictors for novel categories from activations. However, our method directly imprints weights from activations, which is made possible by architecture modifications that introduce a normalization layer.

3. Metric Learning and Softmax Classifiers

In this section, we discuss the connection between a proxy-based objective used in embedding training and softmax cross-entropy loss. Based on these observations, we then describe our method for extending ConvNet classifiers to new classes in the next section.

3.1. Proxy-based Embedding Training

Recent work has blurred the divide between triplet-based embedding training and softmax classification. For example, Neighborhood Components Analysis [4] learns a distance metric with a softmax-like loss,

$$\mathcal{L}_{\text{NCA}}(x, y, Z) = -\log \frac{\exp(-d(x, y))}{\sum_{z \in Z} \exp(-d(x, z))} \quad (1)$$

which makes points x, y with the same label closer than examples z with different labels under the squared Euclidean distance $d(x, y) = ||x - y||_2^2$. Movshovitz-Attias *et al.* [11] reformulated the loss by assigning proxies $p(\cdot)$ to training examples according to the class labels

$$\begin{aligned}\mathcal{L}_{\text{proxy}}(x) &\triangleq \mathcal{L}_{\text{NCA}}(x, p(x), p(Z)) \\ &= -\log \frac{\exp(-d(x, p(x)))}{\sum_{p(z) \in p(Z)} \exp(-d(x, p(z)))}, \quad (2)\end{aligned}$$

where $p(Z)$ is a set of all negative proxies. This formulation allows sampling anchor points x, rather than triplets, for each mini-batch and results in faster convergence than other objectives.

3.2. Connections to Softmax Classifiers

We will now discuss the connections between metric learning and softmax classifiers. We consider the case that each class has exactly one proxy and the proxy of a data point is determined statically according to its label. Concretely, let C be the set of category labels and $P = \{p_1, p_2, \ldots, p_{|C|}\}$ be the set of trainable proxies, then the proxy of every point x is $p(x) = p_{c(x)}$ where $c(x) \in C$ is the class label of x. We argue that the proxies p_c are comparable to weights w_c in softmax classifiers.

To see this, we assume point vectors and proxy vectors are normalized to the same length. It follows that minimizing the squared Euclidean distance between a point x and its proxy $p(x)$ is equivalent to maximizing the inner-product, or equivalently cosine similarity, of the corresponding unit vectors

$$\min d(x, p(x)) \triangleq \min ||x - p(x)||_2^2 = \max x^\top p(x), \quad (3)$$

since $||u - v||_2^2 = 2 - 2u^\top v$ for unit vectors $u, v \in \mathbb{R}^D$. Substituting the squared Euclidean distance with inner product in Eq. 2, the resulting loss can be written as

$$\mathcal{L}(x, c(x)) = -\log \frac{\exp(x^\top p_{c(x)})}{\sum_{c \in C} \exp(x^\top p_c)}, \quad (4)$$

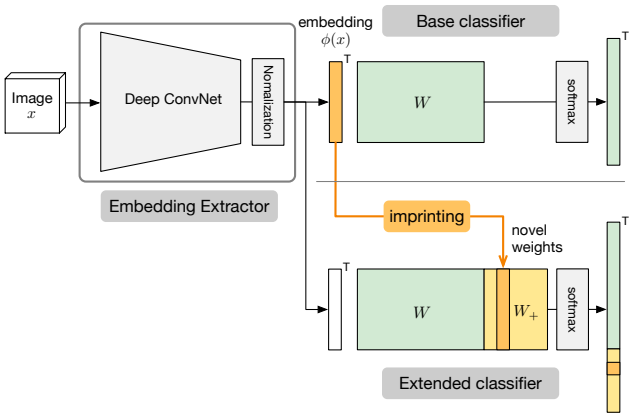


Figure 1. The overall architecture of imprinting. After a base classifier is trained, the embedding vectors of new low-shot examples are used to imprint weights for new classes in the extended classifier.

which is comparable to the softmax cross-entropy loss used for training classifiers

$$\mathcal{L}_{\text{softmax}}(x, c(x)) = -\log \frac{\exp(x^\top w_{c(x)} + b_{c(x)})}{\sum_{c \in C} \exp(x^\top w_c + b_c)}, \quad (5)$$

with bias terms $b_c = 0$ for all $c \in C$.

4. Imprinting

Given the conceptual similarity of normalized embedding vectors and final layer weights as discussed above, it seems natural that we should be able to set network weights for a novel class immediately from a single exemplar. In the following, we outline our proposed method to do this, which we call *imprinting*. In essence, imprinting exploits the symmetry between normalized inputs and weights in a fully connected layer, copying the embedding activations for a novel exemplar into a new set of network weights.

To demonstrate this method, we focus on a two-stage low-shot classification problem where a learner is trained on a set of base classes with abundant training samples in the first stage and then grows its capability to additional novel classes for which only one or a few examples are available in the second stage.

4.1. Model Architecture

Our model consists of two parts. First, an embedding extractor $\phi : \mathbb{R}^N \to \mathbb{R}^D$, parameterized by a convolutional neural network, maps an input image $x \in \mathbb{R}^N$ to a D-dimensional embedding vector $\phi(x)$. Different from standard ConvNet classifier architectures, we add an L_2 normalization layer at the end of the embedding extractor so that the output embedding has unit length, *i.e.* $||\phi(x)||_2 = 1$. Second, a softmax classifier $f(\phi(x))$ maps the embedding

into unnormalized logit scores followed by a softmax activation that produces a probability distribution across all categories

$$f_i(\phi(x)) = \frac{\exp(w_i^\top \phi(x))}{\sum_c \exp(w_c^\top \phi(x))}, \tag{6}$$

where w_i is the i-th column of the weight matrix normalized to unit length. No bias term is used in this layer.

We view each column of the weight matrix as a template of the corresponding category. Unlike in [11] where only the embedding extractor part is used during test time with the auxiliary proxies thrown away, we keep the entirety of the network. In the forward pass, the last layer in our model computes the inner product between the embedding of the input image $\phi(x)$ and all the templates w_i. With embeddings and templates normalized to unit lengths, the resulting prediction is equivalent to finding the nearest template in the embedding space in terms of squared Euclidean distance

$$\hat{y} = \arg\max_{c \in C} w_c^\top \phi(x) = \arg\min_{c \in C} d(\phi(x), w_c). \tag{7}$$

Compared with non-parametric nearest neighbor models, however, our classifier only contains one template per class rather than storing a large set of reference data points.

Normalization. Normalizing embeddings and columns of the weight matrix in the last layer to unit lengths is an important architectural design in our model. Geometrically, normalized embeddings and weights lie on a high-dimensional sphere. In contrast, existing deep neural networks normally encourage activations to have zero mean and unit variance within mini-batches [6] or layers [1] for optimization reasons while they do not address the scale differences between neuron activations and weights. In our model, as a result of normalizing embeddings and columns of the weight matrix, the magnitude differences do not affect the prediction as long as the angle between the normalized vectors remains the same, since the inner product $w_i^\top \phi(x) \in [-1, 1]$ now measures cosine similarity. Recent work in cosine normalization [9] discusses a similar idea of replacing the inner product with a cosine similarity for bounded activations and stable training, while we arrive at this design from a different direction. In particular, this establishes a symmetric relationship between normalized embeddings and weights, which enables us to treat them interchangeably.

Scale factor. The cosine similarity $w_i^\top \phi(x) \in [-1, 1]$ can prevent the normalized probability of the correct class from reaching close to 1 when applying softmax activation. For example, consider for an input x the inner product producing 1 for the correct category and producing the minimum possible value -1 for the incorrect categories, the normalized probability is $p(y_i|x) = e^1/[e^1 + (|C| - 1)e^{-1}] = 0.069$, assuming a total of $|C| = 100$ categories. In consequence, it fails to produce a distribution close to the one-hot encoding of the ground truth label and therefore imposes a lower bound on the cross-entropy loss. This effect becomes more severe as the number of categories increases. To alleviate this problem, we adapt a scaling factor in our model as discussed by Wang *et al*. [24]. Concretely, we modify Eq. 6 by adding a trainable scalar s shared across all classes to scale the inner product

$$f_i(\phi(x)) = \frac{\exp(s w_i^\top \phi(x))}{\sum_c \exp(s w_c^\top \phi(x))}. \tag{8}$$

We also experimented with the option of using an adaptive scale factor per class, but we did not observe significant effects on classification accuracy compared to our use of a single global scale factor.

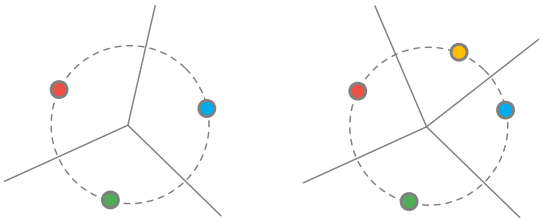
In summary, our model architecture is similar to standard ConvNet classifiers except for two differences. The normalized embeddings and weights introduce a symmetric relationship that allows us to treat them interchangeably. The scaled inner product at the final layer enables training the entire model with the cross-entropy loss in the same way that standard ConvNet classifiers are trained. Next, we discuss how to extend such a classifier to novel categories by leveraging the symmetry between embeddings and weights.

4.2. Weight Imprinting

Inspired by the effectiveness of embeddings in retrieving and recognizing objects from unseen classes in metric learning, our proposed imprinting method is to directly set the final layer weights for new classes from the embeddings of training exemplars. Consider a single training sample x_+ from a novel class, our method computes the embedding $\phi(x_+)$ and uses it to set a new column in the weight matrix for the new class, *i.e.* $w_+ = \phi(x_+)$. Figure 1 illustrates this idea of extending the final layer weight matrix of a trained classifier by imprinting additional columns for new categories.

Intuitively, one can think of the imprinting operation as remembering the semantic embeddings of low-shot examples as the templates for new classes. Figure 2 illustrates the change of the decision boundaries after a new weight column is imprinted. The underlying assumption is that test examples from new classes are closer to the corresponding training examples, even if only one or a few are observed, than to instances of other classes in the embedding space. Notably, this desired property coincides with metric learning objectives such as triplet loss. The proxy-based loss, from which we have derived our method, upper bounds the instance-based triplet loss [11].

Average embedding. If $n > 1$ examples $\{x_+^{(i)}\}_{i=1}^n$ are available for a new class, we compute new weights by averaging the normalized embeddings $\tilde{w}_+ = \frac{1}{n} \sum_{i=1}^n \phi(x_+^{(i)})$



(a) (b)

Figure 2. Illustration of imprinting in the normalized embedding space. (a) Before imprinting, the decision boundaries are determined by the trained weights. (b) With imprinting, the embedding of an example (the yellow point) from a novel class defines a new region.

and re-normalizing the resulting vector to unit length $w_+ = \tilde{w}_+/||\tilde{w}_+||$. In practice, the averaging operation can also be applied to the embeddings computed from the randomly augmented versions of the original low-shot training examples.

Fine-tuning. Since our model architecture has the same differentiable form as ordinary ConvNet classifiers, a fine-tuning step can be applied after new weights are imprinted. The average embedding strategy assumes that examples from each novel class have a unimodal distribution in the embedding space. This may not hold for every novel class since the learned embedding space could be biased towards features that are salient and discriminative among base classes. However, fine-tuning (using backpropagation to further optimize the network weights) should move the embedding space towards having unimodal distribution for the new class.

5. Implementation Details

The implementation details are comparable to [11] and [12]. For training, all the convolutional layers are initialized from ConvNet classifiers pre-trained on the ImageNet dataset [15]. InceptionV1 [21] is used in our experiments. The parameters of the fully-connected layers producing the embedding and unnormalized logit scores are initialized randomly. L_2 normalization is used for embedding vectors and weights in the last layer along the embedding dimension. Input images are resized to 256×256 and cropped to 224×224. Intensity is scaled to $[-1, 1]$. During training, we augment inputs with random cropping and random horizontal flipping. The learning rate is 0.0001 for pre-trained layers; a 10× multiplier is used for randomly initialized layers. We apply exponential decay every four epochs with decay rate 0.94. The RMSProp optimizer is used with momentum 0.9. During testing, input patches are cropped from the center.

6. Experiments

We empirically evaluate the classifiers containing imprinted weights. We first describe the overall protocols, then we present results on the CUB-200-2011 dataset.

6.1. Data Splits

The CUB-200-2011 dataset [23] contains 200 fine-grained categories of birds with 11,788 images. We use the train/test split provided by the dataset. In addition, we treat the first 100 classes as base classes where all the training examples (about 30 images per class on average) from these categories are used to train a base classifier. The remaining 100 classes are treated as novel classes where only n examples from the training split are used for low-shot learning. We experiment with a range of sizes $n = 1, 2, 5, 10, 20$ of novel exemplars for the low-shot training split. During testing, the original test split that includes both base and novel classes is used. We measure the top-1 classification accuracy of the final classifier on all categories. To show the effect of weight imprinting for low-shot categories, we also report the performance on the test examples from the novel classes only.

6.2. Models and Configuration Variants

Imprinting. To obtain imprinted models, we compute embeddings of novel examples and set novel weights in the final layer directly. When more than one novel example is available for a class, the mean of the normalized embeddings is used. The basic configuration (*Imprinting*) uses only the novel examples in their original forms. Alternatively, we experiment with random augmentation (*Imprinting+Aug*). Five augmented versions are generated for each novel example by random cropping and horizontal flipping, followed by averaging the embedded vectors. Both variants require only forward-pass computation of a trained embedding extractor without any iterative optimization. We compare these imprinting variants against a model initialization consisting of random novel weights without fine-tuning (*Rand-noFT*), which also involves zero backpropagation. Random weights are generated with a Xavier uniform initializer [3].

Fine-tuning. To demonstrate that imprinted weights can be used as better initializations than random values, we apply fine-tuning to the imprinting model (*Imprinting+FT*) and to the model with random novel weights (*Rand+FT*), respectively. In both cases, we fine-tune the entire network end-to-end. We use only low-shot examples from novel classes in addition to all training examples from base classes. When the distribution across all classes is imbalanced, we oversample the novel classes such that all the classes are sampled uniformly for each mini-batch. Random data augmentation is also applied.

		$n=$	1	2	5	10	20
w/o FT	Rand-noFT[2]		0.17	0.17	0.17	0.17	0.17
	Imprinting		21.26	28.69	**39.52**	45.77	49.32
	Imprinting + Aug		**21.40**	**30.03**	39.35	**46.35**	**49.80**
w/ FT	Rand + FT		5.25	13.41	34.95	54.33	65.60
	Imprinting + FT		**18.67**	**30.17**	**46.08**	**59.39**	**68.77**
	AllClassJoint		3.89	10.82	33.00	50.24	64.88
	Generator + Classifier [5]		18.56	19.07	20.00	20.27	20.88
	Matching Networks [22]		13.45	14.75	16.65	18.81	25.77

Table 1. 200-way top-1 accuracy for novel-class examples in CUB-200-2011. Imprinting provides good immediate performance without fine tuning. Adding data augmentation (Imprinting+Aug) does not give significant further benefit. The second block of 3 rows shows the results of fine tuning, for which the imprinting initialization retains an advantage. This remains true even when compared to training all classes from scratch (AllClassJoint). The final 2 rows provide comparisons with previous methods.

		$n=$	1	2	5	10	20
w/o FT	Rand-noFT		37.36	37.36	37.36	37.36	37.36
	Imprinting		**44.75**	48.21	**52.95**	55.99	57.47
	Imprinting + Aug		44.60	**48.48**	52.78	**56.51**	**57.84**
w/ FT	Rand + FT		39.26	43.36	53.69	63.17	**68.75**
	Imprinting + FT		**45.81**	**50.41**	**59.15**	**64.65**	68.73
	AllClassJoint		38.02	41.89	52.24	61.11	68.31
	Generator + Classifier [5]		45.42	46.56	47.79	47.88	48.22
	Matching Networks [22]		41.71	43.15	44.46	45.65	48.63

Table 2. 200-way top-1 accuracy measured across examples in all classes (100 base plus 100 novel classes) of CUB-200-2011. Imprinting retains similar advantages for rapid learning and initialization of fine-tuning as seen in Table 1.

Jointly-trained ConvNet classifier. For comparison, we train a ConvNet classifier for base and novel classes jointly without a separate low-shot learning phase (*AllClassJoint*). The same data splits and preprocessing pipeline are used as in the fine-tuning cases. This model does not normalize embeddings or weights.

Other low-shot methods. We also apply the feature generator [5] and matching networks [22] to our normalized embeddings trained with the softmax loss for comparison.

6.3. Results

Tables 1 and 2 show the top-1 accuracy of 200-way classification for novel examples and all examples in CUB-200-2011, respectively. Without any backpropagation, the imprinted weights computed from one-shot examples instantly provide good classification performance: 21.26% on novel classes and 44.75% on all classes. Imprinting using the average of multiple augmented exemplars (Imprinting+Aug), using the same random flips and crops as for base class training, does not give a significant improvement in performance. We conjecture this is because the embedding extractor has been trained on the base classes to be invariant to the applied transformations.

When fine-tuning the network weights with backpropagation, models initialized with imprinted weights (Imprint-

[2]Rand-noFT is listed for easy comparison. Strictly, the header $n = 1, \ldots, 20$ does not apply, since low-shot examples are not used.

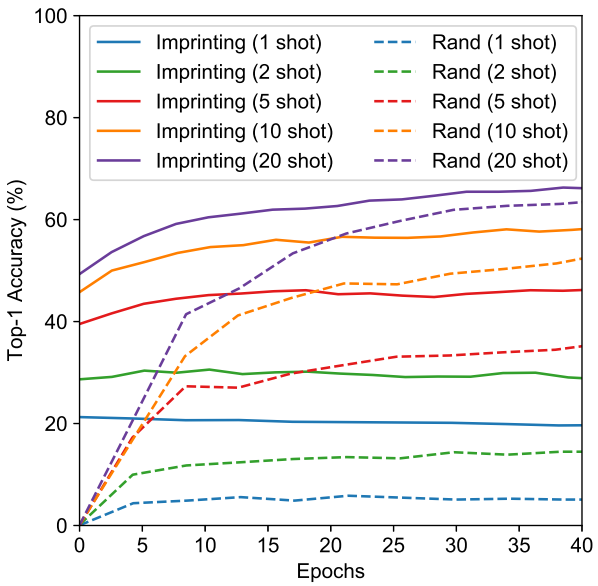


Figure 3. Accuracy of fine-tuned models on novel classes for the first 40 epochs of training. Table 1 lists results after 112 epochs.

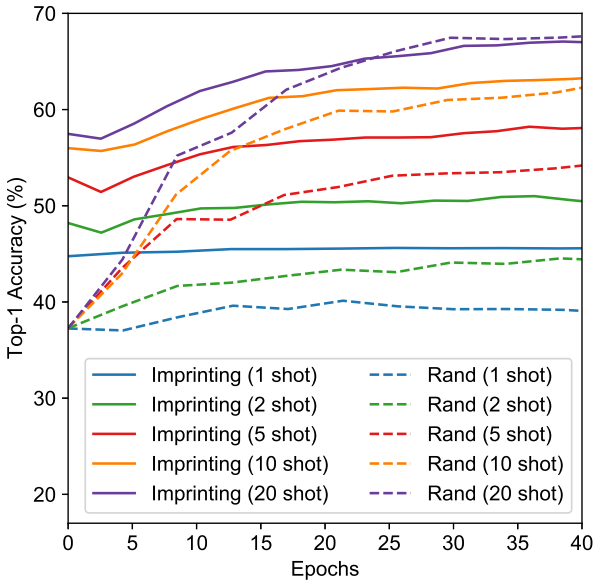


Figure 4. Accuracy of fine-tuned models measured over all classes (100 base plus 100 novel classes) for the first 40 epochs of training. Table 2 lists results after 112 epochs.

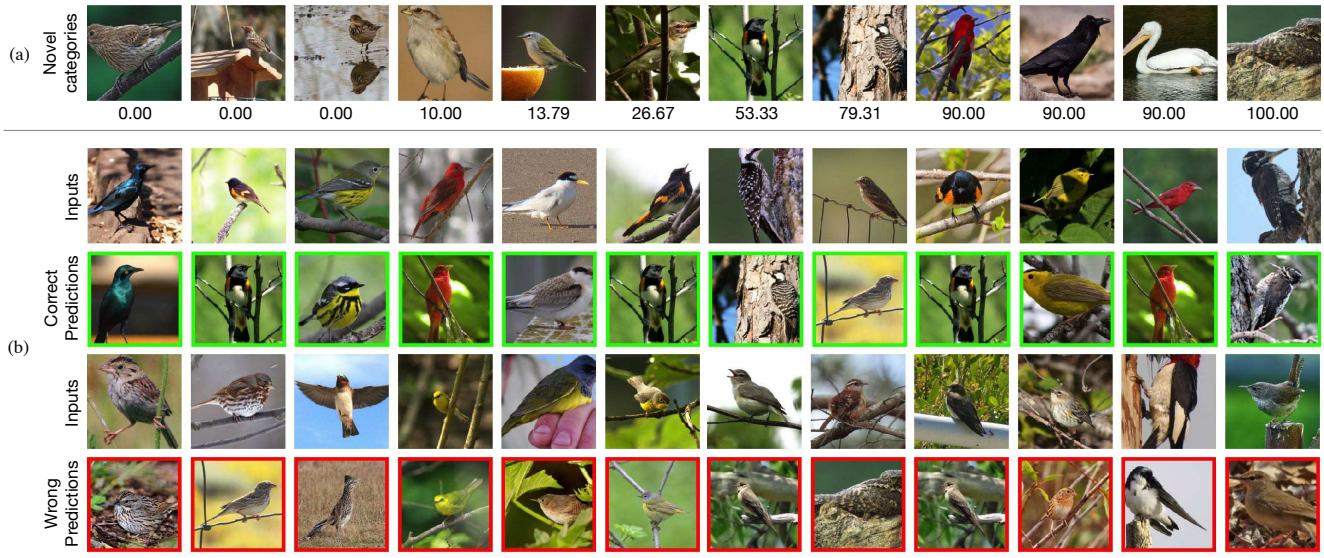


Figure 5. (a) A subset of exemplars used for 1-shot training of novel classes sorted by their recall@1 scores as shown below each exemplar. High-performing categories tend to exhibit more distinctive colors, shapes, and/or textures. (b) Randomly selected success and failure cases predicted by a 1-shot imprinted model on CUB-200-2011. Test images and the 1-shot exemplar whose embedding was used to imprint the predicted class are shown in separate rows. Correct and wrong predictions are marked with green and red borders, respectively.

ing+FT) take less time to converge and achieve better final accuracies than randomly initialized models, especially when limited low-shot examples are used. Figures 3 and 4 plot evaluation accuracy of the fine-tunned models in the first 40 epochs on novel classes and all classes, respectively. Accuracies in Tables 1 and 2 are recorded after around 112 epochs. For cases $n = 1, 2$, the performance of imprinted weights is close to saturation and fine-tuning for more epochs can lead to degraded evaluation accuracies on novel classes, which we conjecture is due to overfitting on the 1 or 2 examples. The results show that the imprinted initialization can lead to better results for low-shot categories even when training from scratch on the entire dataset, as with AllClassJoint.

The classifier using generated features [5] has a similar performance to imprinting for $n = 1$. While the matching network outperforms the feature generator as n increases, we observe a performance gap when compared with imprinting. For our tests we modified the matching network to perform 200-way classification instead of 5-way [22].

Figure 5 shows some sampled results following training of novel categories from the 1-shot imprinted model on CUB-200-2011. The top row shows randomly selected novel categories sorted by their classification accuracy as given below each exemplar. As might be expected, the highest-performing categories tend to exhibit more distinctive features of color, texture, and/or shape. In Figure 5(b) we show randomly selected success and failure cases predicted by the 1-shot imprinted model. The learned embeddings demonstrate an ability to generalize to different viewpoints, poses, and backgrounds from the single training example for each new category.

Transfer Learning with Imprinted Weights. We show that imprinting benefits transfer learning in general. To transfer a trained classifier to a new set of classes, we substitute the final layer parameters with the mean embeddings of examples from new classes. The only difference between our approach and standard transfer learning approaches is that we initialize the new weights by imprinting rather than with random values. Note that the imprinting process requires little cost in terms of computation. Table 3 shows the top-1 classification accuracy of the imprinted model on the new classes. Random initialization yields an accuracy of 0.85% while the models using imprinted weights have accuracies from 26.76% up to 52.25% as the number of training examples increases. Applying random augmentation (Imprinting+Aug) does not impact the performance significantly. Additional fine-tuning improves the performance. When novel training data is scarce ($n = 1, 2, 5$), starting from the imprinted weights (Imprinting+FT) outperforms fine-tuning from random weights (Rand+FT) by a large margin. With more training examples, fine-tuning from imprinted weights converges to similar accuracy as when starting from random weights.

Comparison with Nearest Neighbors. As discussed in Section 2, the usual approach used in metric learning has been to store all exemplar embeddings and use the nearest neighbor algorithm for classification. Therefore, we com-

	$n =$	1	2	5	10	20
w/o FT	Rand-noFT	0.85	0.85	0.85	0.85	0.85
	Imprinting	**26.76**	33.11	43.00	48.74	52.25
	Imprinting + Aug	26.08	**34.13**	**43.34**	**48.91**	**52.94**
w/ FT	Rand+FT	15.90	28.84	46.21	61.37	**71.57**
	Imprinting + FT	**26.59**	**34.33**	**49.39**	**61.65**	70.07

Table 3. Top-1 accuracy for transfer learning on CUB-200-2011 using 1–20 examples for computing imprinted weights. The imprinted weights provide good immediate performance while also providing better final classification accuracy for 1 to 5 shot learning following fine tuning.

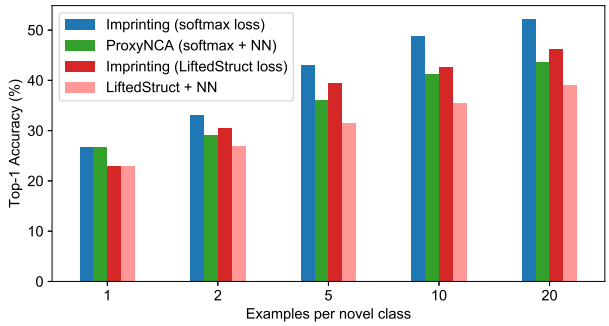


Figure 6. Top-1 accuracy of 100-way classification on novel classes of CUB-200-2011. Imprinting averaged embeddings with a softmax loss (blue bars) outperforms storing all individual embeddings with a nearest-neighbor classifier (green). By comparison, embeddings trained with the lifted structured loss do not perform as well as with the softmax loss (red and pink).

pare our approach of using averaged embeddings with using a nearest-neighbor classifier where the embeddings of n low-shot training examples from each novel class form the population set. When there is only one training example per class, $n = 1$, the imprinted classifier is equivalent to the nearest neighbor classifier. When $n > 1$, the size of the imprinted classifier remains constant, whereas the size of the nearest neighbor classifier grows linearly as n increases. Note that storing all embeddings trained with the softmax loss in a nearest-neighbor classifier is equivalent to a special case of Proxy-NCA [11] using one proxy per class.

Perhaps surprisingly, the averaged embeddings perform better than storing all individual embeddings (Figure 6). We conjecture that the averaging operation reduces potentially noisy dimensions in the embedding to focus on those that are more consistent for that category. Although the averaging may not seem to be the optimal choice in cases where the distribution of novel class examples has multiple modalities in the embedding space, we do not observe this in our experiments. When the embedding space is first trained on the base classes, lower layers of the network will have been trained to bring multiple modalities together for feature inputs to the final linear layer. Moreover, keeping a single embedding for each class in the imprinted classifier has additional benefits since this standard form allows fine-tuning the embedding space with backpropagation and reduces test time computation and memory requirements.

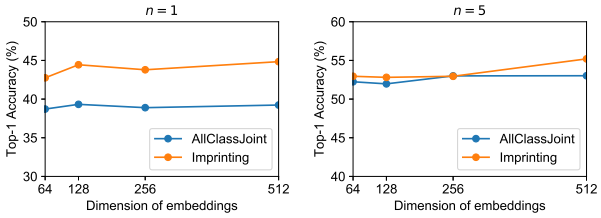


Figure 7. Classification accuracies for Imprinting and AllClassJoint with different embedding dimensionalities under 1-shot and 5-shot settings, respectively.

Comparison with Lifted Structured Loss. Imprinted weights and Proxy-NCA are both trained with the softmax cross-entropy loss. Alternatively, we compare with embeddings trained with the lifted structured loss [12], which is a generalization of the widely used triplet loss. Figure 6 shows that the softmax loss performs better in our experiments than the lifted structured loss. However, the lifted structured loss can also benefit from imprinting averaged embeddings rather than a nearest-neighbor classifier.

Embedding Dimensionality. We use 64-dimensional embeddings in all the experiments above. Empirically we experimented with various settings $D = 64, 128, 256, 512$ for the imprinting model and the jointly-trained ConvNet Classifier (Figure 7). Increasing the dimensionality does not appear to have significant effects on the results.

7. Conclusions

This paper has presented a new method, *weight imprinting*, that directly sets the final layer weights of a ConvNet classifier for novel low-shot categories. This is a valuable complement to stochastic gradient descent, as it provides instant good classification performance on novel categories while allowing for further fine tuning when time permits. The key change that is made to the ConvNet architecture is a normalization layer with a scaling factor that allows activations computed for novel training examples to be directly copied (imprinted) as final layer weights. When multiple low-shot examples are presented, the computed activations for additional examples are averaged with the existing weights, which our experiments show to perform better than the nearest-neighbor approach typically used with embedding methods. An area for future research is whether the imprinting approach can also be used for more rapid training of other network layers, such as when encountering novel lower-level features.

References

[1] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 4

[2] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 539–546, 2005. 2

[3] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010. 5

[4] J. Goldberger, G. E. Hinton, S. T. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, pages 513–520, 2005. 3

[5] B. Hariharan and R. Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017. 1, 2, 6, 7

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 4

[7] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015. 2

[8] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 1

[9] C. Luo, J. Zhan, L. Wang, and Q. Yang. Cosine normalization: Using cosine similarity instead of dot product in neural networks. *arXiv preprint arXiv:1702.05870*, 2017. 4

[10] E. G. Miller, N. E. Matsakis, and P. A. Viola. Learning from one example through shared densities on transforms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 464–471, 2000. 1

[11] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh. No fuss distance metric learning using proxies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 360–368, 2017. 1, 2, 3, 4, 5, 8

[12] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4004–4012, 2016. 2, 5, 8

[13] S. Qiao, C. Liu, W. Shen, and A. Yuille. Few-shot image recognition by predicting parameters from activations. *arXiv preprint arXiv:1706.03466*, 2017. 2

[14] O. Rippel, M. Paluri, P. Dollar, and L. Bourdev. Metric learning with adaptive density discrimination. In *Proceedings of International Conference on Learning Representations*, 2016. 2

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1, 5

[16] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015. 1, 2

[17] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4080–4090, 2017. 2

[18] K. Sohn. Improved deep metric learning with multi-class N-pair loss objective. In *Advances in Neural Information Processing Systems*, pages 1857–1865, 2016. 2

[19] H. O. Song, S. Jegelka, V. Rathod, and K. Murphy. Learnable structured clustering framework for deep metric learning. *arXiv preprint arXiv:1612.01213*, 2016. 2

[20] Y. Song, Y. Li, B. Wu, C.-Y. Chen, X. Zhang, and H. Adam. Learning unified embedding for apparel recognition. In *The IEEE International Conference on Computer Vision Workshops*, Oct 2017. 2

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 5

[22] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016. 2, 6, 7

[23] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD birds-200-2011 dataset. 2011. 5

[24] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. NormFace: L2 hypersphere embedding for face verification. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 1041–1049, 2017. 1, 2, 4

[25] K. Q. Weinberger and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009. 1