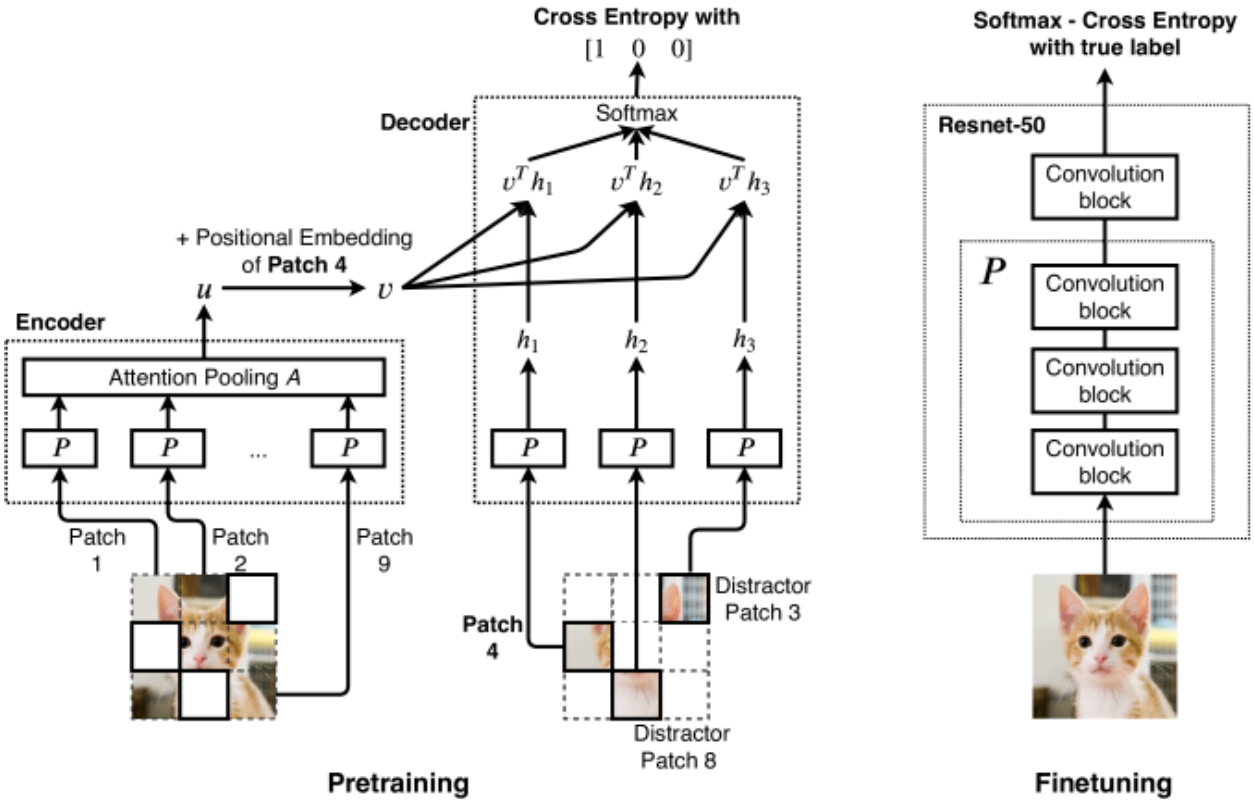
Selfie
Self-supervised Pretraining for Image Embedding



Introduction

A weakness of neural networks is that they often require a large amount of labeled data to perform well. Most practical neural network systems today are trained with supervised learning. Making use of unlabeled data through unsupervised representation learning to improve data-efficiency of neural networks remains an open challenge for the field.

In this paper, A pretraining method called Selfie, which stands for SELF-supervised Image Embedding is proposed. In Selfie, we propose to use classification loss because it is less sensitive to small changes in the image (such as translation of an edge) compared to regression loss which is more sensitive to small perturbations.

Method

During pretraining, this method makes use of an encoder-decoder architecture: the encoder takes in a set of square patches from the input image while the decoder takes in a different set. The encoder builds a single vector u that represents all of its input patches using a patch processing network P followed by an attention pooling network A. The decoder then takes **u** to predict its own input patches from their positions. Instead of predicting the actual pixel content, the decoder classifies the correct patch from negative examples (distractors) with a cross-entropy loss.

Only the first three blocks of ResNet-50 (equivalent to ResNet-36) are used in P to save computation and memory load while Transformer layers have been used for A. During fine tuning, ResNet-50 is applied on the full image. Its first three blocks are initialized from the pretrained **P**, and the network is finetuned end-to-end.

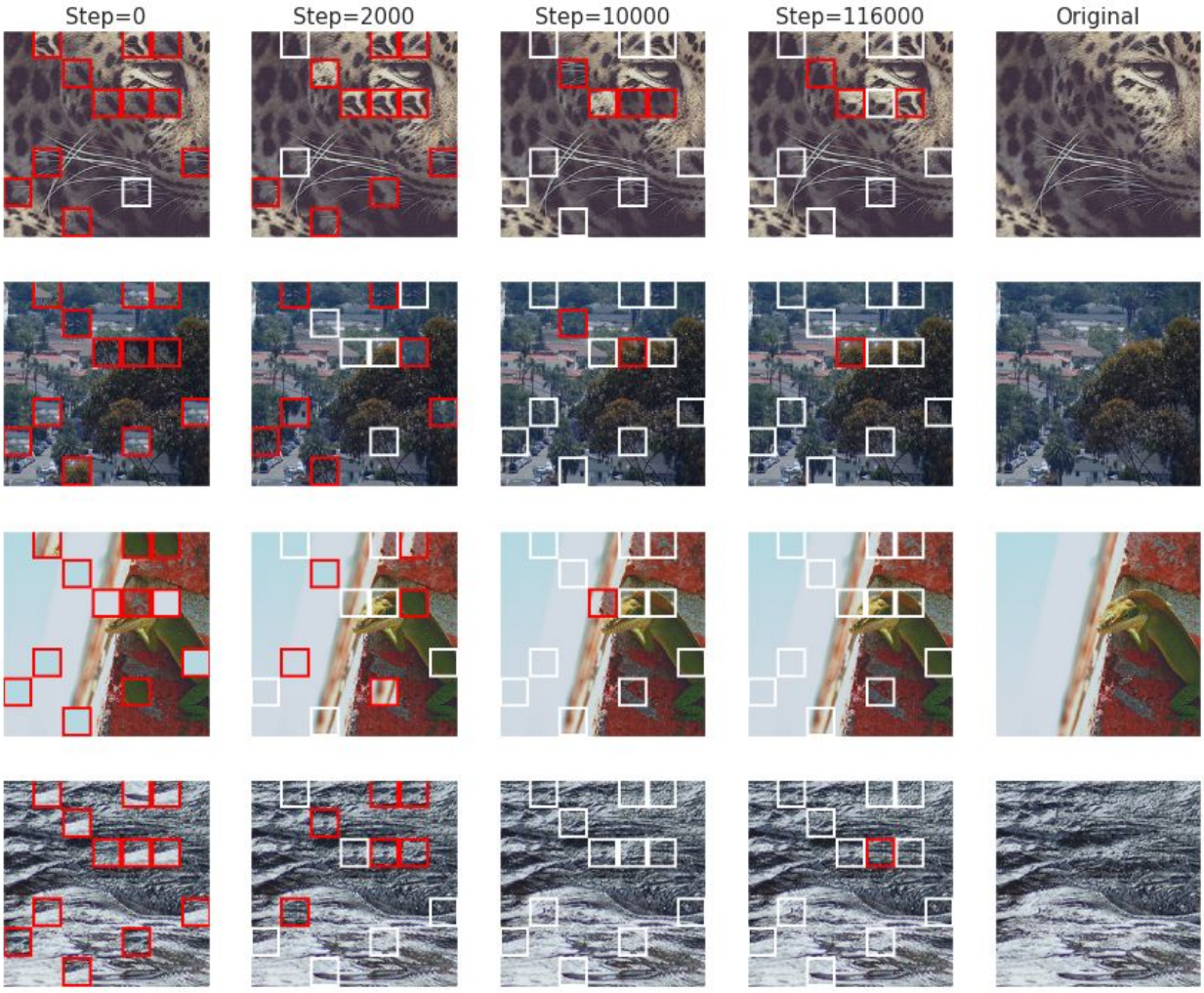
In the implementation, to make sure the distracting patches are hard, we sample them from the same input image and also mask them out in the input image.

Pretraining

The main idea is to use a part of the input image to predict the rest of the image during this phase. To do so, we first sample different square patches from the input. On the encoder side, the output vectors produced by **P** are routed into the attention pooling network to summarize these representations into a single vector **u**. On the decoder side, **P** creates the output vectors $\mathbf{h}_i$ .The decoder then queries the encoder by adding to the output vector **u,** the location embedding of a patch, selected at random among the patches in the decoder to create a vector **v**.

The vector **v** is then used in a dot product to compute the similarity between **v** and each **h**. Having seen the dot products between them, the decoder has to decide which patch is most relevant to fill in the chosen location. The cross entropy loss is applied for this classification task, whereas the encoder and decoder are trained jointly with gradients back-propagated from this loss.

On small images of size 32×32, we use a patch size of 8, while on larger images of size 224×224, we use a patch size of 32×32. The patch size is intentionally selected to divide the image evenly, so that the image can be cut into a grid. For a more efficient use of computation, the decoder is implemented to predict multiple correct patches for multiple locations at the same time.



Attention Pooling

In the attention pooling network **A,** Transformer layers are used to perform pooling. Given a set of input vectors $\{\mathbf{h}_1,\mathbf{h}_2,...,\mathbf{h}_n\}$ produced by applying the patch processing network **P** on different patches, they are to be pooled into a single vector **u** to represent the entire image. There are multiple choices at this stage including max pooling or average pooling. Here, we consider these choices special cases of the attention operation (where the softmax has a temperature approaching zero or

infinity respectively) and let the network learn to pool by itself. To do this, we learn a vector $\mathbf{u_0}$ with the same dimension with **h**'s and feed them together through the Transformer layers. The output **u** corresponding to input $\mathbf{u_o}$ is the pooling result. We discard the output $\mathbf{h_i}$'s .

$$u, h_1^{output}, h_2^{output}, \ldots, h_n^{output} = \text{TransformerLayers}(u_o, h_1, h_2, .., h_n)$$

Every self-attention layer is followed with two fully connected layers that sequentially project the input vector to an intermediate size and back to the original hidden size. The only non-linearity used is GeLU and is applied at the intermediate layer. We perform dropout with rate 0.1 on the output, followed by a residual connection connecting from the block's input and finally layer normalization.

For images of size 32×32, we learn a positional embedding vector for each of the 16 patches of size 8×8. Images of size 224×224, on the other hand, are divided into a grid of 7×7 patches of size 32×32. Since there are significantly more positions in this case, we decompose each positional embedding into two different components: row and column embeddings. The resulting embedding is the sum of these two components. For example, instead of learning 49 positional embeddings, we only need to learn 7+7=14 positional embeddings. This greatly reduces the number of parameters and helps with regularizing the model.

Fine Tuning

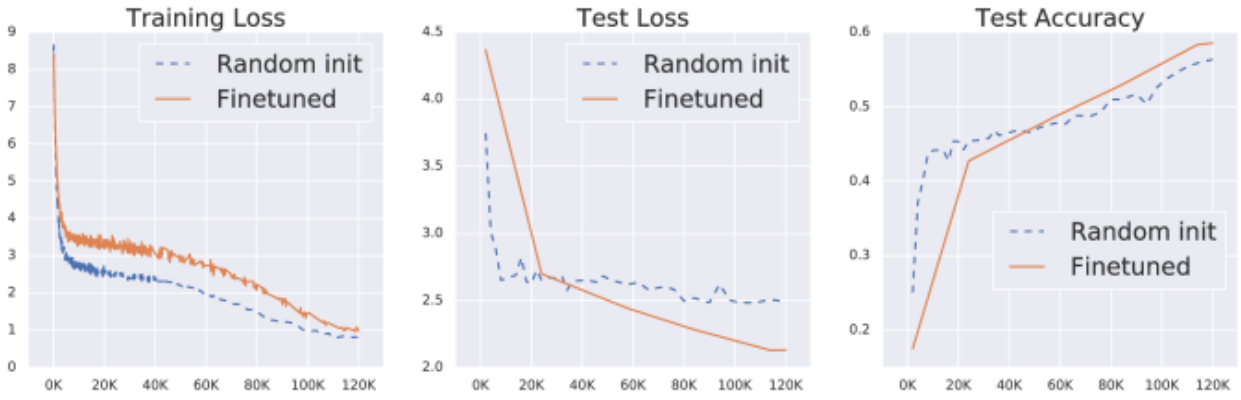
As mentioned above, in this phase, the first three convolution blocks of ResNet-50 are initialized from the pretrained patch processing network. The last convolution block of ResNet-50 is initialized by the standard initialization method. ResNet-50 is then applied on *the full image* and fine tuned end-to-end.

Results

Experiments show that Selfie works well across many datasets, especially when the datasets have a small number of labeled examples. On CIFAR-10, ImageNet 32×32, and ImageNet 224×224, we observe consistent accuracy gains as we vary the amount of labeled data from 5% to 100% of the typical training sets. The gain tends to be highest when the labeled set is small. For example, on ImageNet 224×224 with only 60 labeled examples per class, pretraining using our method improves the mean accuracy of ResNet-50 by 11.1%, going from 35.6% to 46.7%. Additional analysis on ImageNet 224×224 provides evidence that the benefit of self-supervised pretraining significantly takes off when there is at least an order of magnitude (10X) more unlabeled data than labeled data.

In addition to improving the averaged accuracy, pretraining ResNet-50 on unlabeled data also stabilizes its training on the supervised task. We observe this by computing the standard deviation of the final test

accuracy across 5 different runs for all experiments. On CIFAR-10 with 400 examples per class, the standard deviation of the final accuracy reduces 3 times compared to training with the original initialization method. Similarly, on ImageNet rescaled to 32×32, our pretraining process gives an 8X reduction on the test accuracy variability when training on 5% of the full training set.



The sizes of all of our models are chosen such that each architecture has roughly 25M parameters and 50 layers, the same size and depth of a standard ResNet-50. For attention pooling, three attention blocks are added with a hidden size of 1024, intermediate size 640 and 32 attention heads on top of the patch processing network **P**.

Both pretraining and fine tuning tasks are trained using Momentum Optimizer with Nesterov coefficient of 0.9. We use a batch size of 512 for CIFAR-10 and 1024 for ImageNet. Learning rate is scheduled to decay in a cosine shape with a warm up phase of 100 steps. We do not use any extra regularization besides an L2 weight decay of magnitude 0.0001. The full training is done in 120,000 steps.

For each reported experiment, we first tune its hyper-parameters by using 10% of training data as validation set and train the neural net on the remaining 90%. Once we obtain the best hyper-parameter setting, the neural network is retrained on 100% training data 5 times with different random seeds. We report the mean and standard deviation values of these five runs.

		Labeled Data Percentage			
		5%	*8%*	*20%*	*100%*
CIFAR-10	**Supervised**	75.9 ± 0.7	79.3 ± 1.0	88.3 ± 0.3	95.5 ± 0.2
	***Selfie* Pretrained**	75.9 ± 0.4	80.3 ± 0.3	89.1 ± 0.5	95.7 ± 0.1
	Δ	0.0	**+1.0**	**+0.8**	**+0.2**
		5%	*10%*	*20%*	*100%*
ImageNet 32×32	**Supervised**	13.1 ± 0.8	25.9 ± 0.5	32.7 ± 0.4	55.7 ± 0.6
	***Selfie* Pretrained**	18.3 ± 0.1	30.2 ± 0.5	33.5 ± 0.2	56.4 ± 0.6
	Δ	**+5.2**	**+4.3**	**+0.8**	**+0.7**
ImageNet 224×224	**Supervised**	35.6 ± 0.7	59.6 ± 0.2	65.7 ± 0.2	76.9 ± 0.2
	***Selfie* Pretrained**	46.7 ± 0.4	61.9 ± 0.2	67.1 ± 0.2	77.0 ± 0.1
	Δ	**+11.1**	**+2.3**	**+1.4**	**+0.1**