Fine-tuning Momentum Contrast on Image Classification

Lewis Arnsten **Isaac Restrick**

1 Project Objective

For our project, we chose to focus on momentum contrast for unsupervised visual representation learning. Unsupervised learning allows for training neural networks on unlabeled data. These networks are first trained on a proxy task—a task in which we are not genuinely interested, but will teach the network good data representation. They are then fine-tuned for downstream applications. Specifically, we were interested in fine-tuning our model for the purpose of classification. We accomplished this task by pretraining and then fine-tuning on the CIFAR-10 dataset.

2 Method

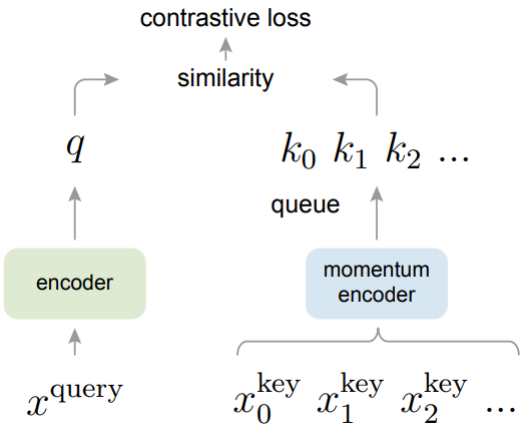

Figure 1: Visual intuition for MoCo, image from original paper

We used Momentum Contrast (MoCo) for unsupervised visual representation learning to generate feature representations of images. MoCo trains an encoder that converts images to feature representations. It does this by optimizing contrastive loss between an encoded input image and a set of encoded keys from a dynamically updated dictionary of sample images.

Contrastive loss is a modified cross entropy loss in which cosine similarity between pairs of images (divided by a constant) is used in the exponent terms. It is calculated on pairs consisting of (input image, key) and is low if the pair is a positive pair and high if it is a negative pair, where the definition of a positive pair is that the two images are data-augmented versions of the same image. An example of one of these data-augmented positive pairs would be the two images of the same cat in figure 2 (augmented through a horizontal flip), while a negative pair would be the cat with any of the dog, elephant, or bird. Intuitively, these positive pairs would have high cosine similarity, and thus lower cross entropy loss (and vice versa for the negative pairs).

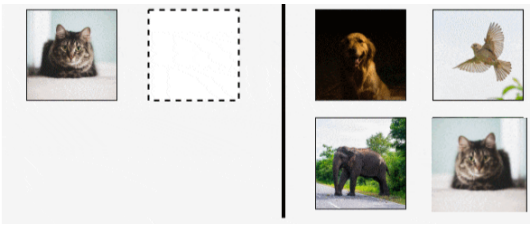

Figure 2: Intuition for contrastive loss

Using the contrastive loss, the encoder for the input images is updated through backpropagation. However, the encoder for the dictionary keys is not updated through backpropagation, but rather through a momentum update dependent on the gradient from the input encoder. In practice a higher momentum value is used and the dictionary encoder updates more slowly than the input encoder, this keeps the dictionary consistent.

Following unsupervised training, we fine-tuned our model for a linear classification task. To do so, we froze the encoder and trained the supervised linear evaluation head with a cross entropy loss. After some experimentation, we chose to train

the linear evaluation head for 100 epochs on the CIFAR-10 dataset. Unlike many implementations of MoCo (which train end-to-end), to underscore our focus on classification as a downstream application of our learned proxy task, our code completes pretraining and fine-tuning for linear classification using two distinct training loops. Thus, our implementation of MoCo saves the model at two checkpoints. The first checkpoint saves the pretrained encoder that has learned the target proxy task. The second checkpoint saves the trained linear classifier. This implementation allowed us to experiment with hyperparameters for unsupervised learning and supervised classification separately. As such, we have submitted separate output logs for training and fine-tuning.

3 Experimental Procedure

While we attempted to complete this project in Google colab, collaboration notebooks cannot execute as background scripts. Thus, we found prolonged training extremely difficult as our connection to the remote GPU would continually be interrupted. To solve this issue we completed this assignment on our own local GPU. We used ResNet-18 as our encoder for the project. For data augmentation, we took a random 32x32 crop from a resized image, randomly applied horizontal flip, applied color jitter, and randomly applied grayscale transformation.

As described, our training procedure was unique in that we created distinct training loops for pretraining and fine-tuning. We used SGD as our optimizer. Our MoCo momentum value was 0.99–in accordance with the original paper. The SGD weight decay was 0.000001 and the fine-tune weight decay was 0. Our fine-tuning learning rate was initialized to 10, while our learning rate for contrastive training was initialized to 0.015. For our pertaining loop we trained for 100 epochs, which took several hours for training ResNet-18. For our fine-tuning loop, we also trained for 100 epochs, which finished considerably faster than pretraining. We trained with a batch size of 128 on one GPU.

4 Results

We can see from our accuracy graphs of top1 and top5 that our accuracy is increasing over the course of the epochs–while we are confident in our model (and data), we lack the time and comput-

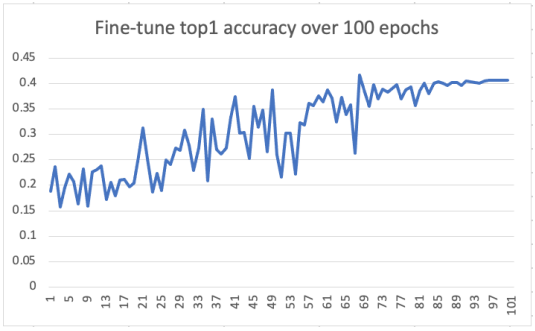

Figure 3: Top1 accuracy on validation dataset across epochs

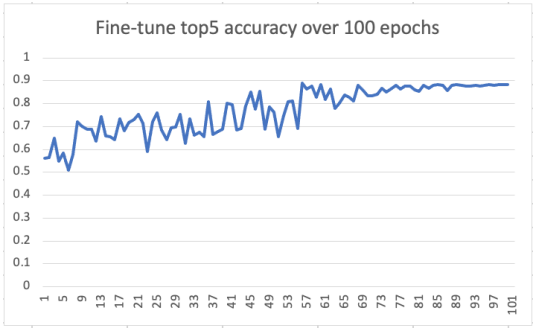

Figure 4: Top5 accuracy on validation dataset across epochs

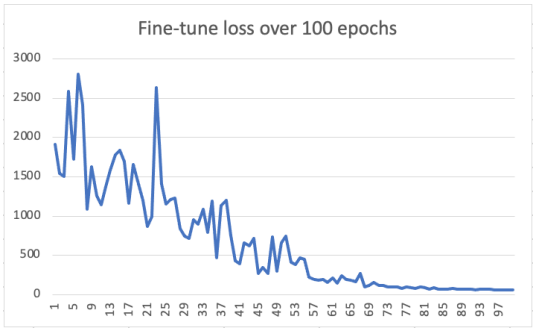

Figure 5: Loss on validation dataset across epochs

Final Loss	Final top1 accuracy	Final top5 accuracy
67.3899	0.4099	0.8740

Figure 6: The linear classification model outputs a vector of ranked classes. The top1 accuracy indicates that the most likely predicted class is the correct class ∼41% of the time. The top5 accuracy indicates that the correct class is within the five most likely predicted classes ∼87% of the time.

ing power necessary to realize its full capability. As expected, the loss appears to greatly decrease over the course of the first $\sim$60 epochs with diminishing returns (but further decreases still) over the subsequent $\sim$40 epochs.

Our fully trained model had a top5 accuracy of 87.4% on linear classification of our validation data. In addition, our model had a top1 accuracy of 40.99%, while the model used in the original MoCo paper had an average top1 accuracy of 60.8%. We attribute this difference in performance to model depth, computational power, and time. We used the ResNet-18 model for our encoder(s), while the original MoCo paper used the ResNet-50 model for its encoder(s). Furthermore, while the authors of the original paper pretrained their model for 200 epochs over 53 hours using 8 nvidia gpus, we pretrained for 100 epochs on 1 underpowered gpu. We followed this pretraining with 100 epochs of fine-tuning for linear classification.

In summary, we are satisfied with the results we were able to attain given our constraints on model depth and available computational power. Although our encoder had less depth than the encoder of the original MoCo paper, we were able to use our encoder to generate features that were good enough to help attain non-trivial top1 (40.99%) and top5 accuracy (87.4%) on linear classification after fine-tuning.

5 References

K. He, et. al Momentum Contrast for Unsupervised Visual Representation Learning

X. Chen, et. al Improved Baselines with Momentum Contrastive Learning

facebookresearch MoCo Code GitHub

HobbitLong CMC GitHub

T. Chen, et. al SimCLR Paper

AidenDurrant Unofficial Pytorch Implementation of MocCoV2 GitHub