Revisiting Self-Supervised Visual Representation Learning: Supplementary Material

Alexander Kolesnikov*, Xiaohua Zhai*, Lucas Beyer*
Google Brain
Zürich, Switzerland
{akolesnikov,xzhai,lbeyer}@google.com

1. Self-supervised model details

For training all self-supervised models we use stochastic gradient descent (SGD) with momentum. The initial learning rate is set to 0.1 and the momentum coefficient is set to 0.9. We train for 35 epochs in total and decay the learning rate by a factor of 10 after 15 and 25 epochs. As we use large mini-batch sizes B during training, we leverage two recommendations from [2]: (1) a learning rate scaling, where the learning rate is multiplied by $\frac{B}{256}$ and (2) a linear learning rate warm-up during the initial 5 epochs.

In the following we give additional details that are specific to the choice of self-supervised learning technique.

Rotation: During training we use the data augmentation mechanism from [9]. We use mini-batches of $B = 1024$ images, where each image is repeated 4 times: once for every rotation. The model is trained on 128 TPU [5] cores.

Exemplar: In order to generate image examples, we use the data augmentation mechanism from [9]. During training, we use mini-batches of size $B = 512$, and for each image in a mini-batch we randomly generate 8 examples. We use an implementation[1] of the triplet loss [8] from the `tensorflow` package [1]. The margin parameter of the triplet loss is set to 0.5. We use 32 TPU cores for training.

Jigsaw: Similar to [7], we preprocess the input images by: (1) resizing the input image to 292×292 and randomly cropping it to 255×255; (2) converting the image to grayscale with probability ⅔ by averaging the color channels; (3) splitting the image into a 3×3 regular grid of cells (size 85×85 each) and randomly cropping 64×64-sized patches inside every cell; (4) standardize every patch individually such that its pixel intensities have zero mean and unit variance. We use SGD with batch size $B = 1024$. For each image individually, we randomly select 16 out of the 100 pre-defined permutations and apply all of them. The model is trained on 32 TPU cores.

Relative Patch Location: We use the same patch prepossessing, representation extraction and training setup as in the Jigsaw model. The only difference is the loss function as discussed in the main text.

2. Downstream training details

Training linear models with SGD: For training linear models with SGD, we use a standard data augmentation technique in the *Rotation* and *Exemplar* cases: (1) resize the image, preserving its aspect ratio such that its smallest side is 256. (2) apply a random crop of size 224×224. For the patch-based methods, we extract representations by averaging the representations of all nine, colorful, standardized patches of an image. At final evaluation-time, fixed patches are obtained by scaling the image to 255×255, cropping the central 192×192 patch and taking the 3×3 grid of 64×64-sized patches from it.

We use a batch-size of 2048 for evaluation of representations from *Rotation* and *Exemplar* models and of 1024 for *Jigsaw* and *Relative Patch Location* models. As we use large mini-batch sizes, we perform learning-rate scaling, as suggested in [2].

Training linear models with L-BFGS: We use a publicly available implementation of the L-BFGS algorithm [6] from the *scipy* [4] package with the default parameters and set the maximum number of updates to 800. For training all our models we apply l_2 penalty $\lambda||W||_2^2$, where $W \in \mathbb{R}^{M \times C}$ is the matrix of model parameters, M is the size of the representation, and C is the number of classes. We set $\lambda = \frac{100.0}{MC}$.

Training MLP models with SGD: In the MLP evaluation scenario, we use a single hidden layer with 1000 channels. At training time, we apply dropout [3] to the hidden layer with a drop rate of 50%. The l_2 regularization scheme is the same as in the *L-BFGS* setting. We optimize the MLP model using stochastic gradient descent with mo-

*equal contribution

[1] https://www.tensorflow.org/api_docs/python/tf/contrib/losses/metric_learning/triplet_semihard_loss

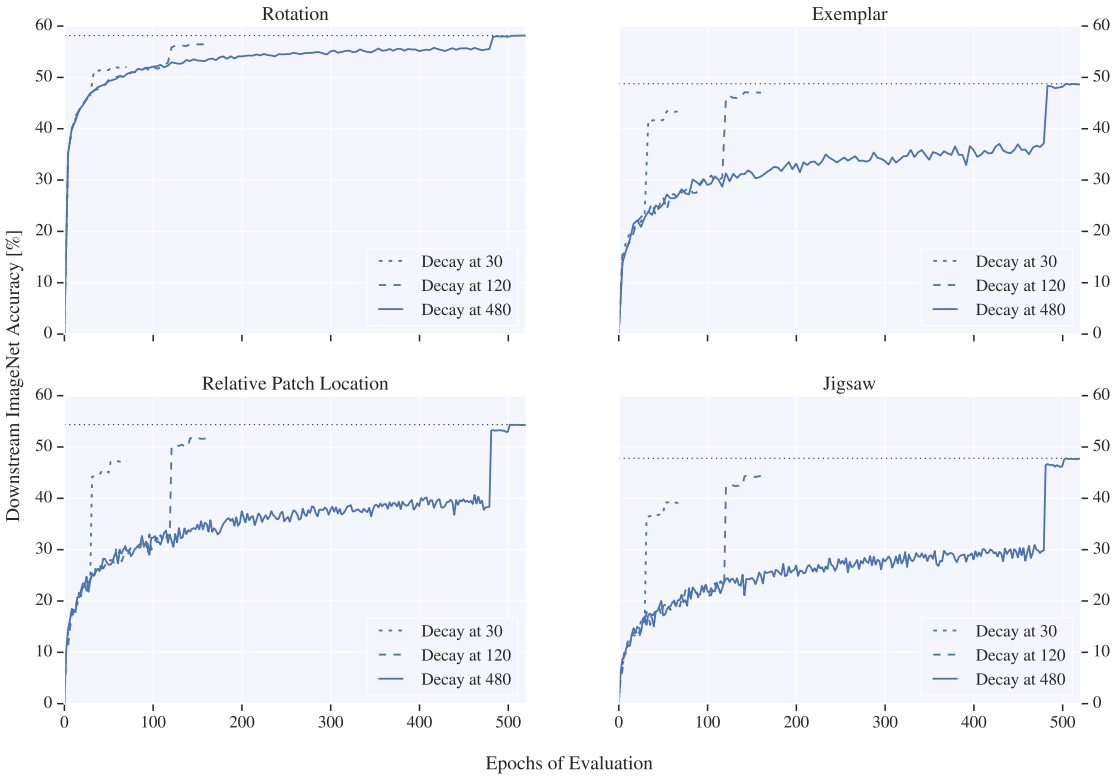

Figure 1. Downstream task accuracy curve of the linear evaluation model trained with SGD on representations learned by the four self-supervision pretext tasks.

Table 1. Skip-connections ablation study for *Rotation* and *Exemplar* techniques. We measure quality of representations learned in different layers by computing accuracy of a linear classifier trained on the corresponding activations and evaluated on the ILSVRC-2012 dataset.

		Block1	Block2	Block3	Block4	PreLogits
Rotation	No Skip	26.0	33.3	38.6	39.6	37.9
	Skip	26.0	36.0	41.2	41.5	44.1
	Diff.	+0.0	+2.7	+2.6	+1.9	+6.2
Exemplar	No Skip	14.4	21.5	26.9	30.7	31.4
	Skip	14.9	22.0	28.3	32.8	35.3
	Diff.	+0.5	+0.5	+1.4	+2.1	+3.9

mentum (the momentum coefficient is 0.9) for 180 epochs. The batch size is 512, initial learning rate is 0.01 and we decay it twice by a factor of 10: after 60 and 120 epochs.

3. Skip-connections ablation study

In the paper we hypothesize that skip-connections help to prevent deterioration of representations learned by self-supervised models. In order to confirm this hypothesis we train the ResNet model with and without skip-connections. Specifically, we use the *Rotation* and *Exemplar* techniques for learning representations to train ResNet18 network from [14] and a modified version of it, where we remove the skip-connections. Both networks achieve similar performance in the fully-supervised setting (69.6% and 68.8% top-1 accuracy), but the one with skip-connections results in significantly better representations (as measured by our main metric), see Table 1.

4. Training linear models with SGD

In Figure 1 we demonstrate how accuracy on the validation data progresses during the course of SGD optimization. We observe that in all cases achieving top accuracy requires training for a very large number of epochs.

5. More Results on Places205 and ImageNet

For completeness, we present full result tables for various settings considered in the main paper. These include numbers for ImageNet evaluated on $10\,\%$ of the data (Table 2) as well as all results when evaluating on the Places205 dataset (Table 3) and a random subset of $5\,\%$ of the Places205 dataset (Table 4).

Table 2. Evaluation on ImageNet with 10 % of the data.

Model	Rotation				Exemplar			RelPatchLoc		Jigsaw	
	4×	8×	12×	16×	4×	8×	12×	4×	8×	4×	8×
RevNet50	31.3	34.6	37.9	38.4	27.1	30.0	31.1	24.6	27.8	25.0	24.2
ResNet50 v2	28.2	31.7	32.2	33.3	28.3	30.1	31.2	25.8	29.4	23.3	24.1
ResNet50 v1	26.8	27.2	27.4	27.8	28.7	30.8	31.7	30.2	33.2	26.4	28.3
RevNet50 (-)	30.2	32.3	33.3	33.4	25.7	26.3	26.4	21.6	25.0	24.1	24.9
ResNet50 v2 (-)	28.4	28.6	28.2	28.5	26.5	27.3	27.3	26.1	26.3	23.9	23.1
VGG19-BN	8.8	6.7	7.6	13.1	16.6	17.7	18.2	15.8	16.8	10.6	10.7

Table 3. Evaluation on Places205 with 100% of the data.

Model	Rotation				Exemplar			RelPatchLoc		Jigsaw	
	4×	8×	12×	16×	4×	8×	12×	4×	8×	4×	8×
RevNet50	41.8	45.3	47.4	47.9	39.4	43.1	44.5	37.5	41.9	37.1	40.7
ResNet50 v2	39.8	43.2	44.2	44.8	39.5	42.8	44.3	38.7	43.2	36.3	39.2
ResNet50 v1	38.1	40.0	41.3	42.0	39.3	43.1	44.5	42.3	46.2	39.4	42.9
RevNet50 (-)	39.5	44.3	46.3	47.5	35.8	39.3	40.7	32.5	39.7	34.5	38.5
ResNet50 v2 (-)	35.5	39.5	41.8	42.8	32.6	34.9	36.0	35.8	39.1	31.6	33.2
VGG19-BN	22.6	21.6	23.8	30.7	29.3	32.0	33.3	31.5	33.6	24.6	27.2

Table 4. Evaluation on Places205 with 5 % of the data.

Model	Rotation				Exemplar			RelPatchLoc		Jigsaw	
	4×	8×	12×	16×	4×	8×	12×	4×	8×	4×	8×
RevNet50	32.1	33.4	34.5	34.8	30.7	31.2	31.6	28.9	29.7	29.3	29.3
ResNet50 v2	30.6	31.8	31.8	32.0	32.1	31.8	32.2	29.8	31.1	29.4	28.9
ResNet50 v1	30.0	29.2	29.0	29.2	32.5	32.5	32.7	33.2	33.9	31.2	31.3
RevNet50 (-)	33.5	34.4	34.5	34.3	31.0	32.2	32.2	27.4	30.8	29.8	31.1
ResNet50 v2 (-)	31.6	33.2	33.6	33.6	30.0	31.4	31.9	30.9	31.9	28.4	28.9
VGG19-BN	16.8	13.9	15.3	20.2	23.5	23.4	23.7	23.8	24.0	19.3	18.7

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. 1

[2] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 1

[3] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 1

[4] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001. 1

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture (ISCA)*. IEEE, 2017. 1

[6] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989. 1

[7] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision (ECCV)*, 2016. 1

[8] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. 1

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1