
Deep Clustering for Unsupervised Learning of Visual Features

News

We release paper and code for SwAV, our new self-supervised method. SwAV pushes self-supervised learning to only 1.2% away from supervised learning on ImageNet with a ResNet-50! It combines on-line clustering with a multi-crop data augmentation.

We also present DeepCluster-v2, which is an improved version of DeepCluster (ResNet-50, better data augmentation, cosine learning rate schedule, MLP projection head, use of centroids, ...). Check out DeepCluster-v2 code.

DeepCluster

This code implements the unsupervised training of convolutional neural networks, or convnets, as described in the paper Deep Clustering for Unsupervised Learning of Visual Features.

Moreover, we provide the evaluation protocol codes we used in the paper: * Pascal VOC classification
* Linear classification on activations * Instance-level image retrieval

Finally, this code also includes a visualisation module that allows to assess visually the quality of the learned features.

Requirements

- a Python installation version 2.7
- the SciPy and scikit-learn packages
- a PyTorch install version 0.1.8 (pytorch.org)
- CUDA 8.0
- a Faiss install (Faiss)
- The ImageNet dataset (which can be automatically downloaded by recent version of torchvision)

Pre-trained models

We provide pre-trained models with AlexNet and VGG-16 architectures, available for download. * The models in Caffe format expect BGR inputs that range in [0, 255]. You do not need to subtract the per-color-channel mean image since the preprocessing of the data is already included in our released

models. * The models in PyTorch format expect RGB inputs that range in [0, 1]. You should preprocess your data before passing them to the released models by normalizing them: `mean_rgb = [0.485, 0.456, 0.406]`; `std_rgb = [0.229, 0.224, 0.225]` Note that in all our released models, sobel filters are computed within the models as two convolutional layers (greyscale + sobel filters).

You can download all variants by running

```
1 $ ./download_model.sh
```

This will fetch the models into `${HOME}/deepcluster_models` by default. You can change that path in the environment variable. Direct download links are provided here: * AlexNet-PyTorch * AlexNet-prototxt + AlexNet-caffemodel * VGG16-PyTorch * VGG16-prototxt + VGG16-caffemodel

We also provide the last epoch cluster assignments for these models. After downloading, open the file with Python 2:

```
1 import pickle
2 with open("./alexnet_cluster_assignment.pickle", "rb") as f:
3     b = pickle.load(f)
```

If you're a Python 3 user, specify `encoding='latin1'` in the load function. Each file is a list of (image path, cluster_index) tuples. * AlexNet-clusters * VGG16-clusters

Finally, we release the features extracted with DeepCluster model for ImageNet dataset. These features are in dimension 4096 and correspond to a forward on the model up to the penultimate convolutional layer (just before last ReLU). In you plan to cluster the features, don't forget to normalize and reduce/whiten them. * AlexNet-imnetfeatures * VGG16-imnetfeatures

Running the unsupervised training

Unsupervised training can be launched by running:

```
1 $ ./main.sh
```

Please provide the path to the data folder:

```
1 DIR=/datasets01/imagenet_full_size/061417/train
```

To train an AlexNet network, specify `ARCH=alexnet` whereas to train a VGG-16 convnet use `ARCH=vgg16`.

You can also specify where you want to save the clustering logs and checkpoints using:

```
1 EXP=exp
```

During training, models are saved every other n iterations (set using the `--checkpoints` flag), and can be found in for instance in `${EXP}/checkpoints/checkpoint_0.pth.tar`. A log of the assignments in the clusters at each epoch can be found in the pickle file `${EXP}/clusters`.

Full documentation of the unsupervised training code `main.py`:

```
1  usage: main.py [-h] [--arch ARCH] [--sobel] [--clustering {Kmeans,PIC}]
2                  [--nmb_cluster NMB_CLUSTER] [--lr LR] [--wd WD]
3                  [--reassign REASSIGN] [--workers WORKERS] [--epochs
4                      EPOCHS]
5                  [--start_epoch START_EPOCH] [--batch BATCH]
6                  [--momentum MOMENTUM] [--resume PATH]
7                  [--checkpoints CHECKPOINTS] [--seed SEED] [--exp EXP]
8                  [--verbose]
9                  DIR
10
11  PyTorch Implementation of DeepCluster
12
13  positional arguments:
14      DIR                path to dataset
15
16  optional arguments:
17      -h, --help          show this help message and exit
18      --arch ARCH, -a ARCH  CNN architecture (default: alexnet)
19      --sobel              Sobel filtering
20      --clustering {Kmeans,PIC}
21                          clustering algorithm (default: Kmeans)
22      --nmb_cluster NMB_CLUSTER, --k NMB_CLUSTER
23                          number of cluster for k-means (default: 10000)
24      --lr LR              learning rate (default: 0.05)
25      --wd WD              weight decay pow (default: -5)
26      --reassign REASSIGN  how many epochs of training between two
27                          consecutive
28                          reassignments of clusters (default: 1)
29      --workers WORKERS    number of data loading workers (default: 4)
30      --epochs EPOCHS      number of total epochs to run (default: 200)
31      --start_epoch START_EPOCH
32                          manual epoch number (useful on restarts) (
33                          default: 0)
34      --batch BATCH        mini-batch size (default: 256)
35      --momentum MOMENTUM  momentum (default: 0.9)
36      --resume PATH        path to checkpoint (default: None)
37      --checkpoints CHECKPOINTS
38                          how many iterations between two checkpoints (
39                          default:
39                          25000)
39      --seed SEED          random seed (default: 31)
39      --exp EXP            path to exp folder
39      --verbose            chatty
```

Evaluation protocols

Pascal VOC

To run the classification task with fine-tuning launch:

```
1 ./eval_voc_classif_all.sh
```

and with no finetuning:

```
1 ./eval_voc_classif_fc6_8.sh
```

Both these scripts download this code. You need to download the VOC 2007 dataset. Then, specify in both `./eval_voc_classif_all.sh` and `./eval_voc_classif_fc6_8.sh` scripts the path `CAFFE` to point to the caffe branch, and `VOC` to point to the Pascal VOC directory. Indicate in `PROTO` and `MODEL` respectively the path to the prototxt file of the model and the path to the model weights of the model to evaluate. The flag `--train-from` allows to indicate the separation between the frozen and to-train layers.

We implemented voc classification with PyTorch.

Erratum: When training the MLP only (fc6-8), the parameters of scaling of the batch-norm layers in the whole network are trained. With freezing these parameters we get 70.4 mAP.

Linear classification on activations

You can run these transfer tasks using:

```
1 $ ./eval_linear.sh
```

You need to specify the path to the supervised data (ImageNet or Places):

```
1 DATA=/datasets01/imagenet_full_size/061417/
```

the path of your model:

```
1 MODEL=/private/home/mathilde/deepcluster/checkpoint.pth.tar
```

and on top of which convolutional layer to train the classifier:

```
1 CONV=3
```

You can specify where you want to save the output of this experiment (checkpoints and best models) with

```
1 EXP=exp
```

Full documentation for this task:

```
1 usage: eval_linear.py [-h] [--data DATA] [--model MODEL] [--conv
    {1,2,3,4,5}]
2                        [--tencrops] [--exp EXP] [--workers WORKERS]
3                        [--epochs EPOCHS] [--batch_size BATCH_SIZE] [--lr
    LR]
4                        [--momentum MOMENTUM] [--weight_decay
    WEIGHT_DECAY]
5                        [--seed SEED] [--verbose]
6
7 Train linear classifier on top of frozen convolutional layers of an
    AlexNet.
8
9 optional arguments:
10  -h, --help            show this help message and exit
11  --data DATA          path to dataset
12  --model MODEL         path to model
13  --conv {1,2,3,4,5}   on top of which convolutional layer train
    logistic
14                        regression
15  --tencrops            validation accuracy averaged over 10 crops
16  --exp EXP            exp folder
17  --workers WORKERS    number of data loading workers (default: 4)
18  --epochs EPOCHS      number of total epochs to run (default: 90)
19  --batch_size BATCH_SIZE
20                        mini-batch size (default: 256)
21  --lr LR             learning rate
22  --momentum MOMENTUM  momentum (default: 0.9)
23  --weight_decay WEIGHT_DECAY, --wd WEIGHT_DECAY
24                        weight decay pow (default: -4)
25  --seed SEED         random seed
26  --verbose            chatty
```

Instance-level image retrieval

You can run the instance-level image retrieval transfer task using:

```
1 ./eval_retrieval.sh
```

Visualisation

We provide two standard visualisation methods presented in our paper.

Filter visualisation with gradient ascent

First, it is possible to learn an input image that maximizes the activation of a given filter. We follow the process described by Yosinski et al. with a cross entropy function between the target filter and the other filters in the same layer. From the visu folder you can run

```
1 ./gradient_ascent.sh
```

You will need to specify the model path `MODEL`, the architecture of your model `ARCH`, the path of the folder in which you want to save the synthetic images `EXP` and the convolutional layer to consider `CONV`.

Full documentation:

```
1 usage: gradient_ascent.py [-h] [--model MODEL] [--arch {alexnet,vgg16}]
2                           [--conv CONV] [--exp EXP] [--lr LR] [--wd WD]
3                           [--sig SIG] [--step STEP] [--niter NITER]
4                           [--idim IDIM]
5
6 Gradient ascent visualisation
7
8 optional arguments:
9   -h, --help            show this help message and exit
10  --model MODEL          Model
11  --arch {alexnet,vgg16}
12                        arch
13  --conv CONV            convolutional layer
14  --exp EXP              path to res
15  --lr LR                learning rate (default: 3)
16  --wd WD                weight decay (default: 10^-5)
17  --sig SIG              gaussian blur (default: 0.3)
18  --step STEP            number of iter between gaussian blurs (default:
19                        5)
20  --niter NITER          total number of iterations (default: 1000)
21  --idim IDIM            size of input image (default: 224)
```

I recommend you play with the hyper-parameters to find a regime where the visualisations are good. For example with the pre-trained deepcluster AlexNet, for conv1 using a learning rate of 3 and 30.000 iterations works well. For conv5, using a learning rate of 30 and 3.000 iterations gives nice images with the other parameters set to their default values.

Top 9 maximally activated images in a dataset

Finally, we provide code to retrieve images in a dataset that maximally activate a given filter in the convnet. From the visu folder, after having changed the fields `MODEL`, `EXP`, `CONV` and `DATA`, run

```
1 ./activ-retrieval.sh
```

DeeperCluster

We have proposed another unsupervised feature learning paper at ICCV 2019. We have shown that unsupervised learning can be used to pre-train convnets, leading to a boost in performance on ImageNet classification. We achieve that by scaling DeepCluster to 96M images and mixing it with RotNet self-supervision. Check out the paper and code.

License

You may find out more about the license here.

Reference

If you use this code, please cite the following paper:

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. “Deep Clustering for Unsupervised Learning of Visual Features.” Proc. ECCV (2018).

```
1 @InProceedings{caron2018deep,
2   title={Deep Clustering for Unsupervised Learning of Visual Features},
3   author={Caron, Mathilde and Bojanowski, Piotr and Joulin, Armand and
4     Douze, Matthijs},
5   booktitle={European Conference on Computer Vision},
6   year={2018},
7 }
```