# Caffe

Deep learning framework by the BVLC

Created by
Yangqing Jia
Lead Developer
Evan Shelhamer

View On GitHub

# Fine-tuning CaffeNet for Style Recognition on "Flickr Style" Data

Fine-tuning takes an already learned model, adapts the architecture, and resumes training from the already learned model weights. Let's fine-tune the BVLC-distributed CaffeNet model on a different dataset, Flickr Style, to predict image style instead of object category.

## Explanation

The Flickr-sourced images of the Style dataset are visually very similar to the ImageNet dataset, on which the `bvlc_reference_caffenet` was trained. Since that model works well for object category classification, we'd like to use it architecture for our style classifier. We also only have 80,000 images to train on, so we'd like to start with the parameters learned on the 1,000,000 ImageNet images, and fine-tune as needed. If we give provide the `weights` argument to the `caffe train` command, the pretrained weights will be loaded into our model, matching layers by name.

Because we are predicting 20 classes instead of a 1,000, we do need to change the last layer in the model. Therefore, we change the name of the last layer from `fc8` to `fc8_flickr` in our prototxt. Since there is no layer named that in the `bvlc_reference_caffenet`, that layer will begin training with random

weights.

We will also decrease the overall learning rate `base_lr` in the solver prototxt, but boost the `lr_mult` on the newly introduced layer. The idea is to have the rest of the model change very slowly with new data, but let the new layer learn fast. Additionally, we set `stepsize` in the solver to a lower value than if we were training from scratch, since we're virtually far along in training and therefore want the learning rate to go down faster. Note that we could also entirely prevent fine-tuning of all layers other than `fc8_flickr` by setting their `lr_mult` to 0.

## Procedure

All steps are to be done from the caffe root directory.

The dataset is distributed as a list of URLs with corresponding labels. Using a script, we will download a small subset of the data and split it into train and val sets.

```
caffe % ./examples/finetune_flickr_style/assemble_data.py -h
usage: assemble_data.py [-h] [-s SEED] [-i IMAGES] [-w WORKERS]

Download a subset of Flickr Style to a directory

optional arguments:
  -h, --help            show this help message and exit
  -s SEED, --seed SEED  random seed
  -i IMAGES, --images IMAGES
                        number of images to use (-1 for all)
  -w WORKERS, --workers WORKERS
                        num workers used to download images. -x uses (all - x)
                        cores.

caffe % python examples/finetune_flickr_style/assemble_data.py --workers=-1 --images=2000 -
-seed 831486
Downloading 2000 images with 7 workers...
Writing train/val for 1939 successfully downloaded images.
```

This script downloads images and writes train/val file lists into `data/flickr_style`. The prototxts in this example assume this, and also assume the presence of the ImageNet mean file (run `get_ilsvrc_aux.sh` from `data/ilsvrc12` to obtain this if you haven't yet).

We'll also need the ImageNet-trained model, which you can obtain by running `./scripts/download_model_binary.py models/bvlc_reference_caffenet`.

Now we can train! (You can fine-tune in CPU mode by leaving out the `-gpu` flag.)

```
caffe % ./build/tools/caffe train -solver models/finetune_flickr_style/solver.prototxt -
weights models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel -gpu 0

[...]

I0828 22:10:04.025378  9718 solver.cpp:46] Solver scaffolding done.
I0828 22:10:04.025388  9718 caffe.cpp:95] Use GPU with device ID 0
I0828 22:10:04.192004  9718 caffe.cpp:107] Finetuning from
models/bvlc_reference_caffenet/bvlc_reference_caffenet.caffemodel

[...]

I0828 22:17:48.338963 11510 solver.cpp:165] Solving FlickrStyleCaffeNet
I0828 22:17:48.339010 11510 solver.cpp:251] Iteration 0, Testing net (#0)
I0828 22:18:14.313817 11510 solver.cpp:302]     Test net output #0: accuracy = 0.0308
I0828 22:18:14.476822 11510 solver.cpp:195] Iteration 0, loss = 3.78589
I0828 22:18:14.476878 11510 solver.cpp:397] Iteration 0, lr = 0.001
I0828 22:18:19.700408 11510 solver.cpp:195] Iteration 20, loss = 3.25728
I0828 22:18:19.700461 11510 solver.cpp:397] Iteration 20, lr = 0.001
I0828 22:18:24.924685 11510 solver.cpp:195] Iteration 40, loss = 2.18531
I0828 22:18:24.924741 11510 solver.cpp:397] Iteration 40, lr = 0.001
I0828 22:18:30.114858 11510 solver.cpp:195] Iteration 60, loss = 2.4915
I0828 22:18:30.114910 11510 solver.cpp:397] Iteration 60, lr = 0.001
I0828 22:18:35.328071 11510 solver.cpp:195] Iteration 80, loss = 2.04539
I0828 22:18:35.328127 11510 solver.cpp:397] Iteration 80, lr = 0.001
I0828 22:18:40.588317 11510 solver.cpp:195] Iteration 100, loss = 2.1924
I0828 22:18:40.588373 11510 solver.cpp:397] Iteration 100, lr = 0.001
I0828 22:18:46.171576 11510 solver.cpp:195] Iteration 120, loss = 2.25107
I0828 22:18:46.171669 11510 solver.cpp:397] Iteration 120, lr = 0.001
I0828 22:18:51.757809 11510 solver.cpp:195] Iteration 140, loss = 1.355
I0828 22:18:51.757863 11510 solver.cpp:397] Iteration 140, lr = 0.001
I0828 22:18:57.345080 11510 solver.cpp:195] Iteration 160, loss = 1.40815
I0828 22:18:57.345135 11510 solver.cpp:397] Iteration 160, lr = 0.001
I0828 22:19:02.928794 11510 solver.cpp:195] Iteration 180, loss = 1.6558
I0828 22:19:02.928850 11510 solver.cpp:397] Iteration 180, lr = 0.001
I0828 22:19:08.514497 11510 solver.cpp:195] Iteration 200, loss = 0.88126
I0828 22:19:08.514552 11510 solver.cpp:397] Iteration 200, lr = 0.001

[...]

I0828 22:22:40.789010 11510 solver.cpp:195] Iteration 960, loss = 0.112586
I0828 22:22:40.789175 11510 solver.cpp:397] Iteration 960, lr = 0.001
I0828 22:22:46.376626 11510 solver.cpp:195] Iteration 980, loss = 0.0959077
I0828 22:22:46.376682 11510 solver.cpp:397] Iteration 980, lr = 0.001
I0828 22:22:51.687258 11510 solver.cpp:251] Iteration 1000, Testing net (#0)
I0828 22:23:17.438894 11510 solver.cpp:302]     Test net output #0: accuracy = 0.2356
```

Note how rapidly the loss went down. Although the 23.5% accuracy is only modest, it was achieved in only 1000, and evidence that the model is starting to learn quickly and well. Once the model is fully fine-tuned on the whole training set over 100,000 iterations the final validation accuracy is 39.16%. This takes ~7 hours in Caffe on a K40 GPU.

For comparison, here is how the loss goes down when we do not start with a pre-trained model:

```
I0828 22:24:18.624004 12919 solver.cpp:165] Solving FlickrStyleCaffeNet
I0828 22:24:18.624099 12919 solver.cpp:251] Iteration 0, Testing net (#0)
I0828 22:24:44.520992 12919 solver.cpp:302]     Test net output #0: accuracy = 0.0366
I0828 22:24:44.676905 12919 solver.cpp:195] Iteration 0, loss = 3.47942
I0828 22:24:44.677120 12919 solver.cpp:397] Iteration 0, lr = 0.001
I0828 22:24:50.152454 12919 solver.cpp:195] Iteration 20, loss = 2.99694
```

```
I0828 22:24:50.152509 12919 solver.cpp:397] Iteration 20, lr = 0.001
I0828 22:24:55.736256 12919 solver.cpp:195] Iteration 40, loss = 3.0498
I0828 22:24:55.736311 12919 solver.cpp:397] Iteration 40, lr = 0.001
I0828 22:25:01.316514 12919 solver.cpp:195] Iteration 60, loss = 2.99549
I0828 22:25:01.316567 12919 solver.cpp:397] Iteration 60, lr = 0.001
I0828 22:25:06.899554 12919 solver.cpp:195] Iteration 80, loss = 3.00573
I0828 22:25:06.899610 12919 solver.cpp:397] Iteration 80, lr = 0.001
I0828 22:25:12.484624 12919 solver.cpp:195] Iteration 100, loss = 2.99094
I0828 22:25:12.484678 12919 solver.cpp:397] Iteration 100, lr = 0.001
I0828 22:25:18.069056 12919 solver.cpp:195] Iteration 120, loss = 3.01616
I0828 22:25:18.069149 12919 solver.cpp:397] Iteration 120, lr = 0.001
I0828 22:25:23.650928 12919 solver.cpp:195] Iteration 140, loss = 2.98786
I0828 22:25:23.650984 12919 solver.cpp:397] Iteration 140, lr = 0.001
I0828 22:25:29.235535 12919 solver.cpp:195] Iteration 160, loss = 3.00724
I0828 22:25:29.235589 12919 solver.cpp:397] Iteration 160, lr = 0.001
I0828 22:25:34.816898 12919 solver.cpp:195] Iteration 180, loss = 3.00099
I0828 22:25:34.816953 12919 solver.cpp:397] Iteration 180, lr = 0.001
I0828 22:25:40.396656 12919 solver.cpp:195] Iteration 200, loss = 2.99848
I0828 22:25:40.396711 12919 solver.cpp:397] Iteration 200, lr = 0.001

[...]

I0828 22:29:12.539094 12919 solver.cpp:195] Iteration 960, loss = 2.99203
I0828 22:29:12.539258 12919 solver.cpp:397] Iteration 960, lr = 0.001
I0828 22:29:18.123092 12919 solver.cpp:195] Iteration 980, loss = 2.99345
I0828 22:29:18.123147 12919 solver.cpp:397] Iteration 980, lr = 0.001
I0828 22:29:23.432059 12919 solver.cpp:251] Iteration 1000, Testing net (#0)
I0828 22:29:49.409044 12919 solver.cpp:302]     Test net output #0: accuracy = 0.0572
```

This model is only beginning to learn.

Fine-tuning can be feasible when training from scratch would not be for lack of time or data. Even in CPU mode each pass through the training set takes ~100 s. GPU fine-tuning is of course faster still and can learn a useful model in minutes or hours instead of days or weeks. Furthermore, note that the model has only trained on < 2,000 instances. Transfer learning a new task like style recognition from the ImageNet pretraining can require much less data than training from scratch.

Now try fine-tuning to your own tasks and data!

## Trained model

We provide a model trained on all 80K images, with final accuracy of 39%. Simply do `./scripts/download_model_binary.py models/finetune_flickr_style` to obtain it.

## License

The Flickr Style dataset as distributed here contains only URLs to images. Some of the images may have copyright. Training a category-recognition model for research/non-commercial use may constitute fair use of this data,

but the result should not be used for commercial purposes.