wilsoncai1992 / **CS280MiniPlaces**

forked from tinghuiz/CS280MiniPlaces





Homework 2 for Berkeley CS 280: our version of the MIT Mini Places challenge





tinghuiz update README — Latest commit 53e

| | |
|---|---|
| log | initial commit of MIT mini places cs280 assignment |
| .gitignore | initial commit of MIT mini places cs280 assignment |
| README.md | update README |
| get_miniplaces.sh | initial commit of MIT mini places cs280 assignment |
| sample_submission.csv | initial commit of MIT mini places cs280 assignment |
| train_places_net.py | initial commit of MIT mini places cs280 assignment |

**README.md**

# Mini Places Challenge

Due: Wednesday, Mar 8, 2017, 11:55pm

This is Homework 2 for Computer Vision (CS 280) at UC Berkeley.

The assignment is to build and train a scene classifier for images. The dataset we'll use is the MIT Mini Places dat consisting of images from one of 100 scene categories. It's a subset of the much larger MIT Places2 dataset. The set of the Mini Places dataset has 100,000 images; the validation and test sets have 10,000 images each.

You can work alone or in teams of 2-3. You're encouraged to work in teams to pool your GPU resources (especiall you'll be using cloud instances rather than your own GPU(s)).

## Data

If you aren't using a cloud GPU instance (will be preloaded with the data) download the dataset and development can use the included `get_miniplaces.sh` script, or manually grab from here.

## Example

This repo provides an example of using Caffe to train and evaluate a convnet -- a variant of AlexNet -- for this task Assuming you have run `./get_miniplaces.sh` to download and setup the data, and have Caffe installed and the module available in your `PYTHONPATH`, you should be able to run it by simply doing

```
python train_places_net.py
```

A log of the output you should get from running this is available (see `log/miniplaces.train_log.txt`). The full co used to train and log to the provided file is as follows:

```
python -u train_places_net.py --gpu 0 --cudnn >
    log/miniplaces.train_log.txt 2>&1 &
```

The `-u` flag tells Python not to buffer the output so you can watch training progress, e.g. by doing `tail -f log/miniplaces.train_log.txt`. The `--cudnn` flag will create layers that make use of NVIDIA's CuDNN library for computation, with the slight drawback that results will be somewhat non-deterministic.

At the end of training, the script will measure the model's performance on both the training and validation sets, as producing a file containing your test set predictions for submission to the evaluation server (if you're satisfied with validation set performance).

The model takes about 1.5 hours to train on a Titan X GPU, and achieves 34.25% top 1 accuracy and 64.27% top accuracy on the validation set. See training and evaluation log at `./log/miniplaces.train_log.txt`.

The AWS cloud instances are quite a bit slower, taking roughly ~0.44 seconds per iteration with CuDNN for a total hours to train for 50K iterations. (Without CuDNN, the time more than doubles at ~1 second per iteration.) Without this training would take around a week (use flag `--gpu -1` if you want to try it).

## Your assignment

You should consider the result from the provided example as a baseline and try to come up with a model that does significantly better on this task. (It's definitely possible! See MIT's leaderboard for some motivation -- the winning achieved 86.07% top 5 accuracy on the test set.)

You may **not** use outside data in your submission -- including using pretrained models on, e.g., ImageNet. See **Competition rules** below for details.

Besides that rule, you're free to try whatever techniques you've heard of in and outside of class, and of course free think up new ones. We're excited to see what you come up with!

You're encouraged to use Caffe as the provided example is done in Caffe, and the instructors are more equipped t you out with it, but you're free to use any software you like, and furthermore you're free to design your model and schemes however you like. (Don't buy this whole "deep learning" craze? Hand-engineer away!)

## Competition rules

*Any* use of data outside the provided Mini Places dataset is **not allowed** for competition submissions. This include use of weights from models trained on outside datasets such as ImageNet or the full MIT Places(2) dataset. (Trans learning by way of training on larger datasets is a great idea in general, but not allowed here.)

On the other hand, using outside data *is* permissible for further analysis and results reported in your write-up (as w for any extensions you might do for your final project). However, if you do this, please be very careful to avoid subi any results to the evaluation server from models leveraging outside data.

Also, don't annotate (or have others annotate) the test set, or manually adjust your model's predictions after lookir test set images. Your submission should be the direct output of a fully automatic classification system.

Besides these rules, you're mostly free to do whatever you want!

## Evaluation server and competition leaderboard

You will create a text (CSV) file specifying your model's top 5 predicted classes (in order of confidence) for each im the test set, and submit this file to an evaluation server for scoring as a deliverable of this assignment. A sample is provided in this repo ( `sample_submission.csv` ). If you're using the provided Caffe example ( `train_places_net.p` test set predictions are formatted in this manner for you and dumped to a file `top_5_predictions.test.csv` after You will be limited to a small number of submissions to the evaluation server for the duration of the competition, to allowing overfitting to the test set. You should primarily use the validation set to measure how your method is perfo Scores will be ranked and posted to a leaderboard visible to the class -- healthy competition is encouraged!

This space will be updated with more details and submission instructions -- stay tuned.



## Deliverables

Your team should submit to the evaluation server at least one set of test set predictions from a model that you implemented and trained.

Your team should also prepare a short report (1-4 page PDF, not including references), with:

- Your team's name on the evaluation server, and the names of all team members
- Details on how you designed and trained your best model(s), and other things you tried that didn't work as we
- A couple bits of experimental and/or analytical analysis, e.g.,
  - Ablation studies: how does varying this parameter, removing this layer, etc., affect performance?
  - What categories are easiest/hardest for the model? Why?
  - Visualizing and understanding what the model learns, e.g., through activation mining, feature visualizatior techniques, t-SNE, ...
  - Does using outside data in some way help? (Just be careful not to mix outside data into your competition submissions!)
  - Anything else you can come up with!

## AWS EC2 Cloud GPUs

If you don't have access to your own CUDA-capable GPU and would like to train models using a GPU, you can us instances via AWS's EC2 service. AWS Educate provides $100 worth of free credits per year to Berkeley students, is enough for roughly a week's worth of GPU instance use (g2.2xlarge instance type). You should be careful to turr your GPU instance(s) when not in use for training to get the most out of your credits and maximize GPU time. Con working on designing your model elsewhere, e.g., on a GPU-less AWS instance or on your own machine.

See this document for detailed AWS and AWS Educate signup instructions and pointers to the image preloaded w Caffe and the data for this assignment.

If using the machine image mentioned in that document, once you connect to the instance, Caffe is preinstalled ar data is pre-loaded. You should be able to get started and run the example immediately by doing

```
$ cd CS280MiniPlaces
$ python train_places_net.py --cudnn --iters 100 # just train for 100 iterations to make sure every
```

(The first time you run training on a fresh AWS instance, the startup time may be very slow likely due to disks spinr Sometimes it helps to run the above `python` command once, wait a bit for training to start, then Ctrl+C to interrup rerun the command.)

## Acknowledgements

This assignment was "heavily inspired" by the MIT assignment. Thanks to all of the instructors for creating this ass and challenge dataset, with special thanks to Bolei Zhou and Prof. Aude Oliva for all their help getting us set up!