Python 2.7 scripts 1/2

- **1_write_cats-vs-dogs_images.py**
  - Creates in **input/jpg** the folders **test**, **train**, **val**, **calib**
  - To be executed only once forever (folder **calib** is needed only for Quantization with DeePhi)
  - This script is different from the one developed for CIFAR10
- **2a_create_lmdb.py**
  - It creates the LMDB databases **input/lmdb/train_lmdb** and **input/lmdb/valid_lmdb** for the training step
  - To be executed only once forever
  - This script is different from the one developed for CIFAR10
- **2b_compute_mean.py**
  - It computes the mean values for the **train_lmdb** database in **input/mean.binaryproto**
  - To be executed only once forever
- **3_read_lmdb.py**
  - Just to debug the first 2 scripts

Python 2.7 scripts 2/2

- **`4_training.py`**
  - To launch the training process in Caffe, given solver and CNN description prototxt files
  - To be used for any trial of training
- **`5_plot_learning_curve.py + plot_training_log.py`**
  - To be launched at the end of the training to plot the learning curves of accuracy and lost (in different ways)
- **`6_make_predictions.py`**
  - To be launched at the end of the training to measure the prediction accuracy achieved by the CNN just trained. You need to have scikit library installed
  - This script is different from the one developed for CIFAR10

- All those scripts are orchestrated in the shell script called **`caffe_flow_CNN.sh`** (see next page). Note that first 4 scripts (1_*, 2a_*, 2b_*, 3_*) are commented in the shell, as you run them only once

caffe_flow_AlexNet.sh script

```
#/bin/sh
HOME_DIR=/home/danieleb

CAFFE_ROOT=$HOME_DIR/caffe_tools/Ristretto
CAFFE_TOOLS_DIR=$CAFFE_ROOT/distribute
#working dir
WORK_DIR=$HOME_DIR/ML/cats-vs-dogs/caffe

NUMIT=12000  # number of iterations
NET=alexnetBNnoLRN
MOD_NUM=2   # model number

: '
# ##############################################################################################################
# SCRIPTS 1 2 3 (DATABASE AND MEAN VALUES)
echo "DATABASE: training and validation in LMDB, test in JPG and MEAN values"

# prepare the databases
python $WORK_DIR/code/1_write_cats-vs-dogs_images.py -p $HOME_DIR/ML/cats-vs-dogs/input/jpg

#create LMDB databases -training (20K), validation (4K), test (1K) images - and compute mean values
python $WORK_DIR/code/2a_create_lmdb.py -i $HOME_DIR/ML/cats-vs-dogs/input/jpg/ -o $HOME_DIR/ML/cats-vs-dogs/input/lmdb
python $WORK_DIR/code/2b_compute_mean.py -i lmdb/train_lmdb -o mean.binaryproto -w $WORK_DIR/../input/
'

# ##############################################################################################################
# SCRIPT 4  (SOLVER AND TRAINING AND LEARNING CURVE)
echo "TRAINING. Remember that: <Epoch_index = floor((iteration_index * batch_size) / (# data_samples))>"

python $WORK_DIR/code/4_training.py -s ./models/$NET/m$MOD_NUM/solver_$MOD_NUM\_$NET.prototxt -l ./models/$NET/m$MOD_NUM/logfile_$MOD_NUM\_$NET.log -c $CAFFE_ROOT

# print image of CNN architecture
echo "PRINT CNN BLOCK DIAGRAM"
python $CAFFE_TOOLS_DIR/python/draw_net.py $WORK_DIR/models/$NET/m$MOD_NUM/train_val_$MOD_NUM\_$NET.prototxt $WORK_DIR/models/$NET/m$MOD_NUM/bd_$MOD_NUM\_$NET.png

# ##############################################################################################################
# SCRIPT 5: plot the learning curve
echo "PLOT LEARNING CURVERS"
python ./code/5_plot_learning_curve.py $WORK_DIR/models/$NET/m$MOD_NUM/logfile_$MOD_NUM\_$NET.log $WORK_DIR/models/$NET/m$MOD_NUM/plt_train_val_$MOD_NUM\_$NET.png

python ./code/plot_training_log.py 6 $WORK_DIR/models/$NET/m$MOD_NUM/plt_trainLoss_$MOD_NUM\_$NET.png     $WORK_DIR/models/$NET/m$MOD_NUM/logfile_$MOD_NUM\_$NET.log
python ./code/plot_training_log.py 2 $WORK_DIR/models/$NET/m$MOD_NUM/plt_testLoss_$MOD_NUM\_$NET.png      $WORK_DIR/models/$NET/m$MOD_NUM/logfile_$MOD_NUM\_$NET.log
python ./code/plot_training_log.py 0 $WORK_DIR/models/$NET/m$MOD_NUM/plt_testAccuracy_$MOD_NUM\_$NET.png  $WORK_DIR/models/$NET/m$MOD_NUM/logfile_$MOD_NUM\_$NET.log

# ##############################################################################################################
# SCRIPT 6 (PREDICTION)
echo "COMPUTE PREDICTIONS"
python ./code/6_make_predictions.py -d ./models/$NET/m$MOD_NUM/deploy_$MOD_NUM\_$NET.prototxt -w ./models/$NET/m$MOD_NUM/snapshot_$MOD_NUM\_$NET\__iter_$NUMIT.caffe
$model 2>&1 | tee ./models/$NET/m$MOD_NUM/predictions_$MOD_NUM\_$NET.txt


-:**-  caffe_flow_AlexNet.sh   All L44    (Shell-script[sh])
```

Execute the commands:

- `cd ~/ML/cats-vs-dogs/caffe`
- `source caffe_flow_AlexNet.sh >2&1 | tee logfile_caffe_flow_AlexNet.txt`

The best model is m2: 94% top -1 accuracy

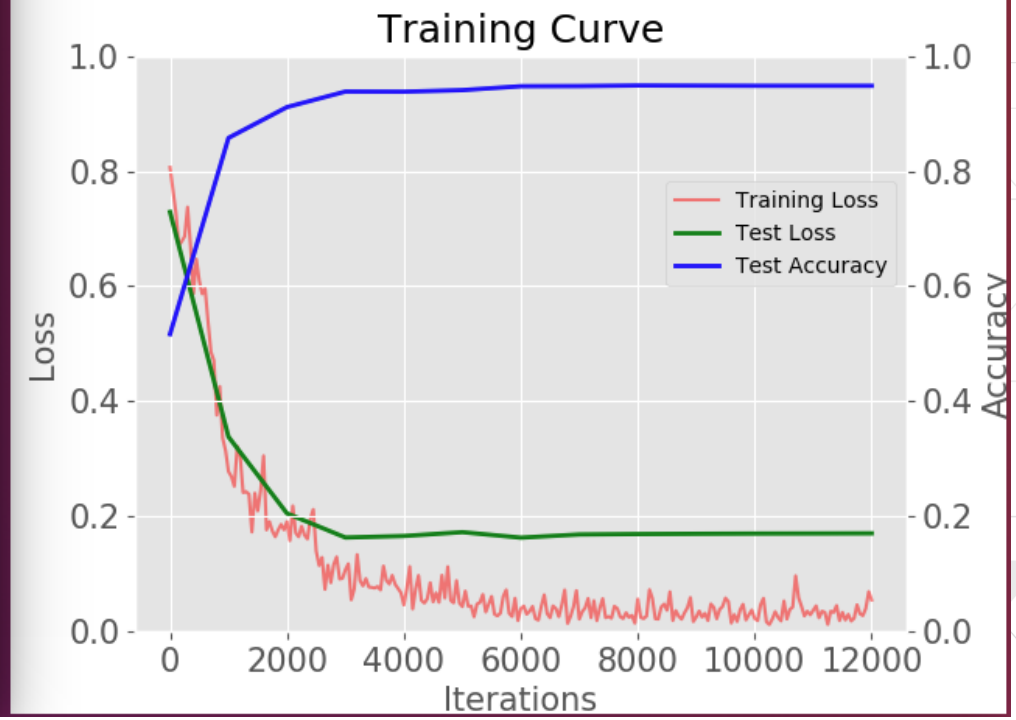
predictions_2_alexnetBNnoLRN.txt [Read-Only] (/home/danieleb/ML/cats-vs-dogs/caffe/mode

Open Save

```
-------
IMAGE: /home/danieleb/ML/cats-vs-dogs/input/jpg/test/dog/dog.7833.jpg
PREDICTED: 1
EXPECTED : 1
-------
IMAGE: /home/danieleb/ML/cats-vs-dogs/input/jpg/test/dog/dog.5271.jpg
PREDICTED: 1
EXPECTED : 1
-------
              precision    recall  f1-score   support

         cat       0.94      0.95      0.94       500
         dog       0.95      0.94      0.94       500

avg / total       0.94      0.94      0.94      1000

SKLEARN Accuracy = 0.94


TOTAL NUMBER OF TRUE  PREDICTIONS =  944
TOTAL NUMBER OF FALSE PREDICTIONS =  56
TOTAL NUMBER OF true  dog PREDICTIONS =  470
TOTAL NUMBER OF true  cat PREDICTIONS =  474
TOTAL NUMBER OF cat predicted as dog  =  26
TOTAL NUMBER OF dog predicted as cat  =  30
MANUALLY COMPUTED RECALL = 0.94
```

Plain Text Tab Width: 8 Ln 4630, Col 1 INS

plt_train_val_2_alexnetBNnoLRN.png



Solver training parameters

File Edit Options Buffers Tools Help

Save Undo

```
net: "/home/root2/ML/cats-vs-dogs/caffe/models/alexnetBNnoLRN/m2/train_val_2_alexnetBNnoLRN.prototxt"

test_iter: 80 # test_iter = validation dataset size / validation batch size

test_interval: 1000

base_lr: 0.001

lr_policy: "step"
gamma: 0.1
stepsize: 2500

display: 50

max_iter: 12000
momentum: 0.9
weight_decay: 0.0005

snapshot: 5000

snapshot_prefix: "/home/root2/ML/cats-vs-dogs/caffe/models/alexnetBNnoLRN/m2/snapshot_2_alexnetBNnoLRN_"

solver_mode: GPU

#type: "SGD"
type: "Adam"

random_seed: 1201
```

- type: "Adam"
- batch_size: 256

-:--- solver_2_alexnetBNnoLRN.prototxt All L1 (Fundamental)

```
name: "alexnetBNnoLRN m2 (as m3 but less DROP and less BN)"
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 227
    mean_value: 106
    mean_value: 116
    mean_value: 124
  }
  data_param {
    source: "/home/root2/ML/cats-vs-dogs/input/lmdb/train_lmdb"
    batch_size: 256
    backend: LMDB
  }
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mirror: false
    crop_size: 227
    mean_value: 106
    mean_value: 116
    mean_value: 124
  }
  data_param {
    source: "/home/root2/ML/cats-vs-dogs/input/lmdb/valid_lmdb"
    batch_size: 50
    backend: LMDB
  }
}
################################# CONV1 => BN1/SC1/RELU1 => MAXPOOL1
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
```

-:**- train_val_2_alexnetBNnoLRN.prototxt Top L43 (Fundamental)

alexnetBNnoLRN: block diagram

Note. DeePhi' DPU does not support RELU before BN, as shown in Figure:
DNNC will generated an error with a misleading message, therefore in the real model they have been swapped.

The real model follow this structure:

CONV1=>BN1=>SC1=>RELU1=>
POOL1=>DROP1 =>

CONV2=>BN2=>SC2=>RELU2=>
POOL2=>DROP2 =>

CONV3=>BN3=>SC3=>RELU3=>
CONV4=>BN4=>SC4=>RELU4=>
CONV5=>BN5=>SC5=>RELU5=>
POOL5=>DROP5 =>

FC6=>BN6=>SC6=>RELU6=>DROP6 =>

FC7=>BN7=>SC7=>RELU7=>DROP7=>

FC8 =>SOFTMAX

ID	name	type	batch	ch_in	dim_in	ch_out	dim_out	ops	mem
1	data	Input		3	227x227	3	227x227		activation 154.59k
2	conv1	Convolution		3	227x227	96	55x55	macc 105.42M	activation 290.4k param 34.94k
3	relu1	ReLU		96	55x55	96	55x55	comp 290.4k	activation 290.4k
4	bn1	BatchNorm		96	55x55	96	55x55	add 290.4k div 290.4k	activation 290.4k param 192
5	scale1	Scale		96	55x55	96	55x55	macc 290.4k	activation 290.4k
6	pool1	Pooling		96	55x55	96	27x27	comp 629.86k	activation 69.98k
7	drop1	Dropout		96	27x27	96	27x27	comp 69.98k	activation 69.98k
8	conv2	Convolution		96	27x27	256	27x27	macc 447.9M	activation 186.62k param 614.66k
9	relu2	ReLU		256	27x27	256	27x27	comp 186.62k	activation 186.62k
10	bn2	BatchNorm		256	27x27	256	27x27	add 186.62k div 186.62k	activation 186.62k param 512
11	scale2	Scale		256	27x27	256	27x27	macc 186.62k	activation 186.62k
12	pool2	Pooling		256	27x27	256	13x13	comp 389.38k	activation 43.26k
13	drop2	Dropout		256	13x13	256	13x13	comp 43.26k	activation 43.26k
14	conv3	Convolution		256	13x13	384	13x13	macc 149.52M	activation 64.9k param 885.12k
15	relu3	ReLU		384	13x13	384	13x13	comp 64.9k	activation 64.9k
16	bn3	BatchNorm		384	13x13	384	13x13	add 64.9k div 64.9k	activation 64.9k param 768
17	scale3	Scale		384	13x13	384	13x13	macc 64.9k	activation 64.9k
18	conv4	Convolution		384	13x13	384	13x13	macc 224.28M	activation 64.9k param 1.33M
19	relu4	ReLU		384	13x13	384	13x13	comp 64.9k	activation 64.9k
20	bn4	BatchNorm		384	13x13	384	13x13	add 64.9k div 64.9k	activation 64.9k param 768
21	scale4	Scale		384	13x13	384	13x13	macc 64.9k	activation 64.9k
22	conv5	Convolution		384	13x13	256	13x13	macc 149.52M	activation 43.26k param 884.99k
23	relu5	ReLU		256	13x13	256	13x13	comp 43.26k	activation 43.26k
24	bn5	BatchNorm		256	13x13	256	13x13	add 43.26k div 43.26k	activation 43.26k param 512
25	scale5	Scale		256	13x13	256	13x13	macc 43.26k	activation 43.26k
26	pool5	Pooling		256	13x13	256	6x6	comp 82.94k	activation 9.22k
27	drop5	Dropout		256	6x6	256	6x6	comp 9.22k	activation 9.22k
28	fc6	InnerProduct		256	6x6	4096	1x1	macc 37.75M	activation 4.1k param 37.75M
29	relu6	ReLU		4096	1x1	4096	1x1	comp 4.1k	activation 4.1k
30	bn6	BatchNorm		4096	1x1	4096	1x1	add 4.1k div 4.1k	activation 4.1k param 8.19k
31	scale6	Scale		4096	1x1	4096	1x1	macc 4.1k	activation 4.1k
32	drop6	Dropout		4096	1x1	4096	1x1	comp 4.1k	activation 4.1k
33	fc7	InnerProduct		4096	1x1	4096	1x1	macc 16.78M	activation 4.1k param 16.78M
34	relu7	ReLU		4096	1x1	4096	1x1	comp 4.1k	activation 4.1k
35	bn7	BatchNorm		4096	1x1	4096	1x1	add 4.1k div 4.1k	activation 4.1k param 8.19k
36	scale7	Scale		4096	1x1	4096	1x1	macc 4.1k	activation 4.1k
37	drop7	Dropout		4096	1x1	4096	1x1	comp 4.1k	activation 4.1k
38	fc8	InnerProduct		4096	1x1	2	1x1	macc 8.19k	activation 2 param 8.19k
39	prob	Softmax		2	1x1	2	1x1	add 2 div 2 exp 2	activation 2
	TOTAL							macc 1.13G comp 1.89M add 658.27k div 658.27k exp 2	activation 3.04M param 58.31M

alexnetBNnoLRN: Summary

- Prediction: 94% top-1average accuracy on the 2 classes
- 56.88 M parameters in the CNN model (with about 8 layers, counting only the CONV and FC layers)

- Caffe Training: 12000 iterations @ 256 batch size (20K training and 4K validation images)
  - **Elapsed time on GPU P6000 @24GB: 64min**
  - Elapsed time on GPU GTX1080 @ 8GB: 162min *(estimated)*
  - **Elapsed time on GPU K80(AWS) @12GB: 152**min
  - **Elapsed time on GPU K1000M *(*)* @ 2GB: 263min**

- *(*) Note: measured on batch_size=8 and with SDG instead of ADAM to avoid Out Of Memory. Nevertheless, the CNN training process does not converge, as the batch size is too small*