CMPE 297 Tutorial

Build Your Own DNN Model

Author: Bhushan Muthiyan

1. **Create Image dataset with appropriate labelling as per number of classes you want to classify your dataset as.**

   The following is the script file for doing so:

```
#!/usr/bin/env sh
# Create the imagenet lmdb inputs
# N.B. set the path to the imagenet train + val data dirs
set -e

EXAMPLE=Path to storing the db file
DATA=~/caffe/examples/illegal  // Path to your folder having label
file
TOOLS=~/caffe/build/tools // Path to caffe build tools folder

TRAIN_DATA_ROOT= Path to training data set images
VAL_DATA_ROOT= Path to testing data set images
# Set RESIZE=true to resize the images to 256x256. Leave as false if
images have
# already been resized using another tool.
#RESIZE=false
RESIZE=true
if $RESIZE; then
  RESIZE_HEIGHT=256
  RESIZE_WIDTH=256
else
  RESIZE_HEIGHT=0
  RESIZE_WIDTH=0
fi

if [ ! -d "$TRAIN_DATA_ROOT" ]; then
  echo "Error: TRAIN_DATA_ROOT is not a path to a directory:
$TRAIN_DATA_ROOT"
  echo "Set the TRAIN_DATA_ROOT variable in create_imagenet.sh to the
path" \
       "where the ImageNet training data is stored."
  exit 1
fi
```

```
if [ ! -d "$VAL_DATA_ROOT" ]; then
  echo "Error: VAL_DATA_ROOT is not a path to a directory:
$VAL_DATA_ROOT"
  echo "Set the VAL_DATA_ROOT variable in create_imagenet.sh to the
path" \
       "where the ImageNet validation data is stored."
  exit 1
fi

echo "Creating train lmdb..."

GLOG_logtostderr=1 $TOOLS/convert_imageset \
    --resize_height=$RESIZE_HEIGHT \
    --resize_width=$RESIZE_WIDTH \
    --shuffle \
    $TRAIN_DATA_ROOT \
    $DATA/train.txt \
    $EXAMPLE/name of your train lmdb file
echo "Creating val lmdb..."

GLOG_logtostderr=1 $TOOLS/convert_imageset \
    --resize_height=$RESIZE_HEIGHT \
    --resize_width=$RESIZE_WIDTH \
    --shuffle \
    $VAL_DATA_ROOT \
    $DATA/val.txt \
    $EXAMPLE/ name of your test or val lmdb file


echo "Done."
```

Run this script from the caffe root

2. Generation of Label file which is provided to the above script:

The label file is nothing but a text file containing names of the dataset images with labelled class to which they belong to. The classes depend upon the number of classification one wants to perform. We provide this labelled data for the testing and training purpose. Supply this file to the DATA parameter above. Create a similar one for testing as well.

The following is the label file for training

```
Furniture.1.jpg 0 → Class number
Furniture.2.jpg 0
Furniture.3.jpg 0
Furniture.4.jpg 0
Furniture.5.jpg 0
Furniture.6.jpg 0
Furniture.7.jpg 0
Furniture.8.jpg 0
Furniture.9.jpg 0
Furniture.10.jpg 0
Furniture.11.jpg 0
Furniture.12.jpg 0
Furniture.13.jpg 0
Furniture.14.jpg 0
Furniture.15.jpg 0
Furniture.16.jpg 0
Furniture.17.jpg 0
Furniture.18.jpg 0
mattress.1.jpg 1 → Class number
mattress.2.jpg 1
mattress.3.jpg 1
mattress.4.jpg 1
mattress.5.jpg 1
mattress.6.jpg 1
mattress.7.jpg 1
mattress.8.jpg 1
mattress.9.jpg 1
mattress.10.jpg 1
mattress.11.jpg 1
mattress.12.jpg 1
mattress.13.jpg 1
mattress.14.jpg 1
mattress.15.jpg 1
mattress.16.jpg 1
mattress.17.jpg 1
mattress.18.jpg 1
mattress.19.jpg 1
mattress.20.jpg 1
Waste.1.jpg 2
Waste.2.jpg 2
Waste.3.jpg 2
Waste.4.jpg 2
Waste.5.jpg 2
Waste.6.jpg 2
```

```
Waste.7.jpg 2
Waste.8.jpg 2
Waste.9.jpg 2
Waste.10.jpg 2
Waste.11.jpg 2
Waste.12.jpg 2
Waste.13.jpg 2
Waste.14.jpg 2
```

3. Mean File generation

Generate image mean file for the above created database. Caffe has in built function for this as well. The aim of mean file generation is to perform preprocessing on the given input dataset. The extension for mean file is .binaryproto. This is not mandatory but gives better result in terms of accuracy and better classification.

The following is the script for doing so:

```
#!/usr/bin/env sh
# Compute the mean image from the imagenet training lmdb
# N.B. this is available in data/ilsvrc12

EXAMPLE =Path to storing the mean file
DATA= Path to your folder
TOOLS=/home/bhushan/caffe/build/tools //Path to caffe build tools
$TOOLS/compute_image_mean $EXAMPLE/ Train db file \
  $DATA/ Name of meanfile.binaryproto

echo "Done."
```

4. Model file for training

Next choose any existing model and modify it. Change the model file to point to your train and test lmdb files. Supply your mean file. In the last layer change your number of output to the number of classes you have in your label file. There are two files here. One is the solver file and other is training model file. The solver file has first line as the path to the training model file. Also specify #max iterations, snapshot prefix path, # step size. The snapshot is capture which can be used to resume a training from a particular point.

The following is the model file

```
name: "CaffeNet"
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 256
    mean_file: "Path to mean file"
  }
# mean pixel / channel-wise mean instead of mean image
#  transform_param {
#    crop_size: 256
#    mean_value: 104
#    mean_value: 117
#    mean_value: 123
```

```
#    mirror: true
#  }
  data_param {
    source: "Path to train lmdb file"
    batch_size: 50 Specify batch size here
    backend: LMDB
  }
}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mirror: false
    crop_size: 256
    mean_file: "  Path to mean file "
  }
# mean pixel / channel-wise mean instead of mean image
#  transform_param {
#    crop_size: 227
#    mean_value: 104
#    mean_value: 117
#    mean_value: 123
```

```
#    mirror: false
#  }
  data_param {
    source: "Path to test lmdb file"
    batch_size: 50
    backend: LMDB
  }
}
```

In the last layer

```
  inner_product_param {
    num_output: 2 → Change this to the number of classes in your label file.
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "fc8"
  bottom: "label"
```

```
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc8"
  bottom: "label"
  top: "loss"
}
```

Solver file: Specify your number of iteration learning rate CPU/GPU mode etc here and point it to your training model file

```
net: "Path to your training model file"
test_interval: 225
base_lr: 0.01
lr_policy: "step"
gamma: 0.1
stepsize: 225
display: 200
max_iter: 550
momentum: 0.9
weight_decay: 0.0005
snapshot: 225
snapshot_prefix: "Path to your model files/Name of snapshot model"
```

```
solver_mode: CPU
```

Here is the script file to invoke and start the training process.

```
#!/usr/bin/env sh

set -e


./build/tools/caffe train \

    --solver=Path to solver file $@
```

5. Testing Phase

The testing script requires the path to deploy.prototxt. Make the similar changes as per the training file. Ensure that the image size is kept uniform. It doesn't affect the convolution layer but it does affect the fully connected layer which might crash due to parameter mismatch. The weight parameter in the testing script stands for the trained model created using the above steps. The path to input test image is also supplied. The input image is also resized as per above specifications. All of the above was generated with the python interface of caffe. There are two methods to predict one using classifier and other using caffe.net. It should be noted that the output classification varies depending upon the prediction method used.

The following is the deploy.prototxt file for testing purpose.

```
name: "CaffeNet"

layer {

  name: "data"

  type: "Input"

  top: "data"

  input_param { shape: { dim: 10 dim: 3 dim: 256 dim: 256 } } → keep this consistent

}

layer {

  name: "fc8"

  type: "InnerProduct"

  bottom: "fc7"

  top: "fc8"

  inner_product_param {

    num_output: 2 → Change this as per your number of classes in label file

  }

}

layer {

  name: "prob"
```

```
  type: "Softmax"

  bottom: "fc8"

  top: "prob"

}
```

The following is the python script.

```
import numpy as np

import matplotlib.pyplot as plt



# Make sure that caffe is on the python path:

caffe_root = './'  # this file is expected to be in
{caffe_root}/examples

import sys

sys.path.insert(0, caffe_root + 'python')



import caffe



# Set the right path to your model definition file, pretrained model
weights,

# and the image you would like to classify.

#MODEL_FILE = './examples/cifar10/cifar10_quick.prototxt'  //
deploy.prototxt file for our model

MODEL_FILE = ' Path to  deploy.prototxt file for our model’

PRETRAINED = ' Path to our model file ’

IMAGE_FILE = ' Path to input test image’



caffe.set_mode_cpu()

net = caffe.Classifier(MODEL_FILE, PRETRAINED,
```

```
                              mean=np.load(caffe_root + 'Path to npy
file').mean(1).mean(1),
                              channel_swap=(2,1,0),
                              raw_scale=255,
                              image_dims=(256, 256))
input_image = caffe.io.load_image(IMAGE_FILE)
plt.imshow(input_image)


prediction = net.predict([input_image])  # predict takes any number of
images, and formats them for the Caffe net automatically
print 'prediction shape:', prediction[0].shape
plt.plot(prediction[0])
print 'predicted class:', prediction[0].argmax()
plt.show()
```

The following are the different parameters from the testing script function.

The `channel_swap` is to reverse `RGB` into `BGR`, which is apparently necessary if you use a reference image net model, based on a comment in [1]. In your case the images are greyscale, so you probably do not have three channels. You might need to set it to (0, 0, 0), but even that might not help (I am unsure on the exact implementation of `channel_swap`). If that does not help, the simplest solution might be to preprocess you data by splitting every pixel into three values (RGB) with equal values. After that you might drop `channel_swap` altogether, because your channels have the same value, and swapping them is a no-op.

Mean is what will be subtracted from your input data to center it. (Remember that neural networks need the data to have zero mean, while the input images usually have positive mean, hence the need of the subtraction). The mean you subtract should be the same that was used for training, so using mean from the file associated with the model is correct. I am not sure, however, on whether you should call `.mean(1)` on it -- did you get that line from some example? If yes, then it is most likely the correct thing to do.

`raw_scale` is a scale of your input data. The model expects pixels to be normalized, so if your input data has values between 0 and 255, then `raw_scale` set to 255 is correct. If your data has values between 0 and 1, then `raw_scale` should be set to 1.

Reference: http://stackoverflow.com/questions/33765029/caffe-how-to-predict-from-a-pretrained-net