무료 전자 책

배우기 caffe

Free unaffiliated eBook created from
Stack Overflow contributors.

#caffe

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: caffe

It is an unofficial and free caffe ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official caffe.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: caffe

Caffe Convolutional Neural Networks (CNN) C ++ . Caffe Berkeley Vision and Learning Center (BVLC) .

Caffe " " . . Caffe CNN / . "" Caffe Caffe .

Caffe BSD 2 .

Caffe GitHub .

Caffe .

- CNN , DAG (Directed Acylclic Graph) .
- GPU (4) . GPU . GPU . , TitanX GTX 980 . (: Kepler Fermi 3) .

Caffe (OOP) .

caffe Caffe .

1.0	2017-04-19

Examples

Ubuntu 14.04 x64 14.10 x64 Caffe, pycaffe .

. "bash compile_caffe_ubuntu_14.sh"(Ubuntu 30 ~ 60).

```
# This script installs Caffe and pycaffe.
# CPU only, multi-threaded Caffe.

# Usage:
# 0. Set up here how many cores you want to use during the installation:
# By default Caffe will use all these cores.
NUMBER_OF_CORES=4

sudo apt-get install -y libprotobuf-dev libleveldb-dev libsnappy-dev
sudo apt-get install -y libopencv-dev libhdf5-serial-dev
sudo apt-get install -y --no-install-recommends libboost-all-dev
sudo apt-get install -y libatlas-base-dev
sudo apt-get install -y python-dev
sudo apt-get install -y python-pip git

# For Ubuntu 14.04
sudo apt-get install -y libgflags-dev libgoogle-glog-dev liblmdb-dev protobuf-compiler

# Install LMDB
git clone https://github.com/LMDB/lmdb.git
cd lmdb/libraries/liblmdb
sudo make
sudo make install

# More pre-requisites
sudo apt-get install -y cmake unzip doxygen
sudo apt-get install -y protobuf-compiler
sudo apt-get install -y libffi-dev python-pip python-dev build-essential
sudo pip install lmdb
sudo pip install numpy
sudo apt-get install -y python-numpy
sudo apt-get install -y gfortran # required by scipy
sudo pip install scipy # required by scikit-image
sudo apt-get install -y python-scipy # in case pip failed
sudo apt-get install -y python-nose
sudo pip install scikit-image # to fix https://github.com/BVLC/caffe/issues/50


# Get caffe (http://caffe.berkeleyvision.org/installation.html#compilation)
cd
mkdir caffe
cd caffe
wget https://github.com/BVLC/caffe/archive/master.zip
unzip -o master.zip
cd caffe-master

# Prepare Python binding (pycaffe)
cd python
for req in $(cat requirements.txt); do sudo pip install $req; done

# to be able to call "import caffe" from Python after reboot:
echo "export PYTHONPATH=$(pwd):$PYTHONPATH " >> ~/.bash_profile
source ~/.bash_profile # Update shell
cd ..

# Compile caffe and pycaffe
cp Makefile.config.example Makefile.config
sed -i '8s/.*/CPU_ONLY := 1/' Makefile.config # Line 8: CPU only
sudo apt-get install -y libopenblas-dev
sed -i '33s/.*/BLAS := open/' Makefile.config # Line 33: to use OpenBLAS
# Note that if one day the Makefile.config changes and these line numbers may change
echo "export OPENBLAS_NUM_THREADS=($NUMBER_OF_CORES)" >> ~/.bash_profile
mkdir build
cd build
cmake ..
cd ..
make all -j$NUMBER_OF_CORES # 4 is the number of parallel threads for compilation: typically
equal to number of physical cores
make pycaffe -j$NUMBER_OF_CORES
make test
make runtest
#make matcaffe
make distribute

# Afew few more dependencies for pycaffe
sudo pip install pydot
sudo apt-get install -y graphviz
sudo pip install scikit-learn
```

"source ~ / .bash_profile" 'python import caffe' .

Caffe

Caffe . Caffe ATLAS OpenBLAS . .

1. `sudo apt-get install -y libopenblas-dev`
2. Caffe `Makefile.config` `BLAS := atlas` `BLAS := open` `BLAS := open`
3. Caffe `export OPENBLAS_NUM_THREADS=4` Caffe 4 .

Caffe ( )

`weight_decay` `regularization_type` .

. `decay_mult` . `decay_mult` `decay_mult` `decay_mult*weight_decay` .

, .

```
layer {
  name: "Convolution1"
  type: "Convolution"
  bottom: "data"
  top: "Convolution1"
  param {
    decay_mult: 0
  }
  convolution_param {
    num_output: 32
    pad: 0
    kernel_size: 3
    stride: 1
    weight_filler {
      type: "xavier"
    }
  }
}
```

.

caffe : https://riptutorial.com/ko/caffe/topic/4382/caffe-

2: pycaffe Caffe

Examples

PyCaffe Python (Iris) Caffe . .

iris_tuto.py

```
import subprocess
import platform
import copy

from sklearn.datasets import load_iris
import sklearn.metrics
import numpy as np
from sklearn.cross_validation import StratifiedShuffleSplit
import matplotlib.pyplot as plt
import h5py
import caffe
import caffe.draw


def load_data():
    '''
    Load Iris Data set
    '''
    data = load_iris()
    print(data.data)
    print(data.target)
    targets = np.zeros((len(data.target), 3))
    for count, target in enumerate(data.target):
        targets[count][target]= 1
    print(targets)

    new_data = {}
    #new_data['input'] = data.data
    new_data['input'] = np.reshape(data.data, (150,1,1,4))
    new_data['output'] = targets
    #print(new_data['input'].shape)
    #new_data['input'] = np.random.random((150, 1, 1, 4))
    #print(new_data['input'].shape)
    #new_data['output'] = np.random.random_integers(0, 1, size=(150,3))
    #print(new_data['input'])

    return new_data

def save_data_as_hdf5(hdf5_data_filename, data):
    '''
    HDF5 is one of the data formats Caffe accepts
    '''
    with h5py.File(hdf5_data_filename, 'w') as f:
        f['data'] = data['input'].astype(np.float32)
        f['label'] = data['output'].astype(np.float32)


def train(solver_prototxt_filename):
    '''
    Train the ANN
    '''
    caffe.set_mode_cpu()
    solver = caffe.get_solver(solver_prototxt_filename)
    solver.solve()


def print_network_parameters(net):
    '''
    Print the parameters of the network
    '''
    print(net)
    print('net.inputs: {0}'.format(net.inputs))
    print('net.outputs: {0}'.format(net.outputs))
    print('net.blobs: {0}'.format(net.blobs))
    print('net.params: {0}'.format(net.params))

def get_predicted_output(deploy_prototxt_filename, caffemodel_filename, input, net = None):
    '''
    Get the predicted output, i.e. perform a forward pass
    '''
    if net is None:
        net = caffe.Net(deploy_prototxt_filename,caffemodel_filename, caffe.TEST)

    #input = np.array([[ 5.1,  3.5,  1.4,  0.2]])
    #input = np.random.random((1, 1, 1))
    #print(input)
    #print(input.shape)
    out = net.forward(data=input)
    #print('out: {0}'.format(out))
    return out[net.outputs[0]]


import google.protobuf
def print_network(prototxt_filename, caffemodel_filename):
    '''
    Draw the ANN architecture
    '''
    _net = caffe.proto.caffe_pb2.NetParameter()
    f = open(prototxt_filename)
    google.protobuf.text_format.Merge(f.read(), _net)
    caffe.draw.draw_net_to_file(_net, prototxt_filename + '.png' )
    print('Draw ANN done!')


def print_network_weights(prototxt_filename, caffemodel_filename):
    '''
    For each ANN layer, print weight heatmap and weight histogram
    '''
    net = caffe.Net(prototxt_filename,caffemodel_filename, caffe.TEST)
    for layer_name in net.params:
        # weights heatmap
        arr = net.params[layer_name][0].data
        plt.clf()
        fig = plt.figure(figsize=(10,10))
        ax = fig.add_subplot(111)
        cax = ax.matshow(arr, interpolation='none')
        fig.colorbar(cax, orientation="horizontal")
        plt.savefig('{0}_weights_{1}.png'.format(caffemodel_filename, layer_name), dpi=100,
format='png', bbox_inches='tight') # use format='svg' or 'pdf' for vectorial pictures
        plt.close()
        # weights histogram
        plt.clf()
        plt.hist(arr.tolist(), bins=20)
        plt.savefig('{0}_weights_hist_{1}.png'.format(caffemodel_filename, layer_name),
dpi=100, format='png', bbox_inches='tight') # use format='svg' or 'pdf' for vectorial pictures
        plt.close()


def get_predicted_outputs(deploy_prototxt_filename, caffemodel_filename, inputs):
    '''
    Get several predicted outputs
    '''
    outputs = []
    net = caffe.Net(deploy_prototxt_filename,caffemodel_filename, caffe.TEST)
    for input in inputs:
        #print(input)
        outputs.append(copy.deepcopy(get_predicted_output(deploy_prototxt_filename,
caffemodel_filename, input, net)))
    return outputs


def get_accuracy(true_outputs, predicted_outputs):
    '''

    '''
    number_of_samples = true_outputs.shape[0]
    number_of_outputs = true_outputs.shape[1]
    threshold = 0.0 # 0 if SigmoidCrossEntropyLoss ; 0.5 if EuclideanLoss
    for output_number in range(number_of_outputs):
        predicted_output_binary = []
        for sample_number in range(number_of_samples):
            #print(predicted_outputs)
            #print(predicted_outputs[sample_number][output_number])
            if predicted_outputs[sample_number][0][output_number] < threshold:
                predicted_output = 0
            else:
                predicted_output = 1
            predicted_output_binary.append(predicted_output)

        print('accuracy: {0}'.format(sklearn.metrics.accuracy_score(true_outputs[:,
output_number], predicted_output_binary)))
        print(sklearn.metrics.confusion_matrix(true_outputs[:, output_number],
predicted_output_binary))


def main():
    '''
    This is the main function
    '''

    # Set parameters
    solver_prototxt_filename = 'iris_solver.prototxt'
    train_test_prototxt_filename = 'iris_train_test.prototxt'
    deploy_prototxt_filename  = 'iris_deploy.prototxt'
    deploy_prototxt_filename  = 'iris_deploy.prototxt'
    deploy_prototxt_batch2_filename  = 'iris_deploy_batchsize2.prototxt'
    hdf5_train_data_filename = 'iris_train_data.hdf5'
    hdf5_test_data_filename = 'iris_test_data.hdf5'
    caffemodel_filename = 'iris__iter_5000.caffemodel' # generated by train()
    # Prepare data
    data = load_data()
    print(data)
    train_data = data
    test_data = data
    save_data_as_hdf5(hdf5_train_data_filename, data)
    save_data_as_hdf5(hdf5_test_data_filename, data)

    # Train network
    train(solver_prototxt_filename)

    # Print network
    print_network(deploy_prototxt_filename, caffemodel_filename)
    print_network(train_test_prototxt_filename, caffemodel_filename)
    print_network_weights(train_test_prototxt_filename, caffemodel_filename)

    # Compute performance metrics
    #inputs = input = np.array([[[[ 5.1,  3.5,  1.4,  0.2]]],[[[ 5.9,  3. ,  5.1,  1.8]]]])
    inputs = data['input']
    outputs = get_predicted_outputs(deploy_prototxt_filename, caffemodel_filename, inputs)
    get_accuracy(data['output'], outputs)


if __name__ == "__main__":
    main()
```

iris_train_test.prototxt iris_deploy.prototxt .

iris_train_test.prototxt :

```
name: "IrisNet"
layer {
  name: "iris"
  type: "HDF5Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  hdf5_data_param {
    source: "iris_train_data.txt"
    batch_size: 1

  }
}

layer {
  name: "iris"
  type: "HDF5Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  hdf5_data_param {
    source: "iris_test_data.txt"
    batch_size: 1

  }
}
```

```
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "data"
  top: "ip1"
  param {
    lr_mult: 1  # the learning rate multiplier for weights
  }
  param {
    lr_mult: 2  # the learning rate multiplier for biases
  }
  inner_product_param {
    num_output: 50
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "drop1"
  type: "Dropout"
  bottom: "ip1"
  top: "ip1"
  dropout_param {
    dropout_ratio: 0.5
  }
}


layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 50
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "drop2"
  type: "Dropout"
  bottom: "ip2"
  top: "ip2"
  dropout_param {
    dropout_ratio: 0.4
  }
}



layer {
  name: "ip3"
  type: "InnerProduct"
  bottom: "ip2"
  top: "ip3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 3
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}

layer {
  name: "drop3"
  type: "Dropout"
  bottom: "ip3"
  top: "ip3"
  dropout_param {
    dropout_ratio: 0.3
  }
}

layer {
  name: "loss"
  type: "SigmoidCrossEntropyLoss"
  # type: "EuclideanLoss"
  # type: "HingeLoss"
  bottom: "ip3"
  bottom: "label"
  top: "loss"
}
```

`iris_deploy.prototxt` :

```
name: "IrisNet"
input: "data"
```

```
input_dim: 1 # batch size
input_dim: 1
input_dim: 1
input_dim: 4


layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "data"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 50
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "drop1"
  type: "Dropout"
  bottom: "ip1"
  top: "ip1"
  dropout_param {
    dropout_ratio: 0.5
  }
}


layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 50
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "drop2"
  type: "Dropout"
  bottom: "ip2"
  top: "ip2"
  dropout_param {
    dropout_ratio: 0.4
  }
}


layer {
  name: "ip3"
  type: "InnerProduct"
  bottom: "ip2"
  top: "ip3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 3
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}

layer {
  name: "drop3"
  type: "Dropout"
  bottom: "ip3"
  top: "ip3"
  dropout_param {
    dropout_ratio: 0.3
  }
}
```

`iris_solver.prototxt` :

```
# The train/test net protocol buffer definition
net: "iris_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
test_iter: 1
# Carry out testing every test_interval training iterations.
test_interval: 1000
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.0001
momentum: 0.001
weight_decay: 0.0005
# The learning rate policy
```

```
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 1000
# The maximum number of iterations
max_iter: 5000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "iris_"
# solver mode: CPU or GPU
solver_mode: CPU # GPU
```

pycaffe Caffe : https://riptutorial.com/ko/caffe/topic/4618/pycaffe-caffe--

3:

Examples

Caffe (,) . "Data" lmdb leveldb . "Data" .

Caffe convert_imageset

caffe caffe (convert_imageset).
caffe make make tools .
$CAFFE_ROOT/build/tools convert_imageset .

: (/path/to/jpegs/).
: <path / to / file> (: /path/to/labels/train.txt) . :

img_0000.jpeg 1
img_0001.jpeg 0
img_0002.jpeg 0

1 0 .

```
~$ GLOG_logtostderr=1 $CAFFE_ROOT/build/tools/convert_imageset \
    --resize_height=200 --resize_width=200 --shuffle  \
    /path/to/jpegs/ \
    /path/to/labels/train.txt \
    /path/to/lmdb/train_lmdb
```

:

- GLOG_logtostderr 1 . convert_imageset stderr .
- --resize_height --resize_width (200x200 .
- --shuffle /path/to/labels/train.txt .
- images , . convert_imageset , .

:

- --backend - lmdb levelDB .
- --gray - .
- --encoded --encoded_type - (jpg / png) .
- --help - . *Flags from tools / convert_imageset.cpp* .

$CAFFE_ROOT/examples/imagenet/convert_imagenet.sh convert_imageset .

.

HDF5 .

Caffe "HDF5Data" . / hdf5 .
python h5py hdf5 caffe "HDF5Data" .

hdf5

'train.txt' .

```
import h5py, os
import caffe
import numpy as np

SIZE = 224 # fixed size to all images
with open( 'train.txt', 'r' ) as T :
    lines = T.readlines()
# If you do not have enough memory split data into
# multiple batches and generate multiple separate h5 files
X = np.zeros( (len(lines), 3, SIZE, SIZE), dtype='f4' )
y = np.zeros( (1,len(lines)), dtype='f4' )
for i,l in enumerate(lines):
    sp = l.split(' ')
    img = caffe.io.load_image( sp[0] )
    img = caffe.io.resize( img, (SIZE, SIZE, 3) ) # resize to fixed size
    # you may apply other input transformations here...
    # Note that the transformation should take img from size-by-size-by-3 and transpose it to
3-by-size-by-size
    X[i] = img
    y[i] = float(sp[1])
with h5py.File('train.h5','w') as H:
    H.create_dataset( 'X', data=X ) # note the name X given to the dataset!
    H.create_dataset( 'y', data=y ) # note the name y given to the dataset!
with open('train_h5_list.txt','w') as L:
    L.write( 'train.h5' ) # list all h5 files you are going to use
```

"HDF5Data"

h5 train_val.prototxt HDF5 .

```
 layer {
   type: "HDF5Data"
   top: "X" # same name as given in create_dataset!
   top: "y"
   hdf5_data_param {
     source: "train_h5_list.txt" # do not give the h5 files directly, but the list.
     batch_size: 32
   }
   include { phase:TRAIN }
 }
```

.

Caffe HDF5 . HDF5 2GB . 2GB .

HDF5 2GB .

```
Check failed: shape[i] <= 2147483647 / count_ (100 vs. 71) blob size exceeds INT_MAX
```

2GB ?

Caffe .

shuffle == true HDF5 HDF5 .

: https://riptutorial.com/ko/caffe/topic/5344/--

4: Caffe - , ,

caffe caffe . . OOP .

caffe (C ++) 4 caffe . . .

4 caffe .

-
-
-
-

caffe .

caffe , .

Examples

.

- CNN caffe . CNN (: ,) . . . Solver Solver ( ) .
- : CNN . CNN . CNN .
- **Net** : Net CNN . Net Solver CNN . Net CNN .
- : CNN . CNN . Caffe **Net** " **Layer** " . (: , , 2D)
- **BLOB** : CNN . CNN . Blob .

Caffe - , , : https://riptutorial.com/ko/caffe/topic/5810/-caffe-----------

5:

:

" 0 / (1) .

[1] .

[...]

[1] S. Ioffe and C. Szegedy, "Batch Normalization : ." arXiv preprint arXiv : 1502.03167 (2015). "

use_global_stats	2016 3 2 rohrbach - .
(use_global_stats)	" / : (: param {lr_mult : 0}) 0 .
(use_global_stats)	batch_norm_layer.cpp prototxt use_global_stats . use_global_stats_ = this-> phase_ == TEST; "

Examples

Prototxt

BatchNorm . BatchNorm . `layerx` `layerx-bn` blob .

```
layer { bottom: 'layerx' top: 'layerx-bn' name: 'layerx-bn' type: 'BatchNorm'
  batch_norm_param {
    use_global_stats: false  # calculate the mean and variance for each mini-batch
    moving_average_fraction: .999  # doesn't effect training
  }
  param { lr_mult: 0 }
  param { lr_mult: 0 }
  param { lr_mult: 0 }}
# channel-wise scale and bias are separate
layer { bottom: 'layerx-bn' top: 'layerx-bn' name: 'layerx-bn-scale' type: 'Scale',
  scale_param {
    bias_term: true
    axis: 1      # scale separately for each channel
    num_axes: 1  # ... but not spatially (default)
    filler { type: 'constant' value: 1 }           # initialize scaling to 1
    bias_filler { type: 'constant' value: 0.001 }  # initialize bias
}}
```

.

Prototxt

use_global_stats true true .   .

```
layer { bottom: 'layerx' top: 'layerx-bn' name: 'layerx-bn' type: 'BatchNorm'
  batch_norm_param {
    use_global_stats: true  # use pre-calculated average and variance
  }
  param { lr_mult: 0 }
  param { lr_mult: 0 }
  param { lr_mult: 0 }}
# channel-wise scale and bias are separate
layer { bottom: 'layerx-bn' top: 'layerx-bn' name: 'layerx-bn-scale' type: 'Scale',
  scale_param {
    bias_term: true
    axis: 1      # scale separately for each channel
    num_axes: 1  # ... but not spatially (default)
}}
```

: https://riptutorial.com/ko/caffe/topic/6575/-

6:

Caffe . . Caffe .

C ++ . .

	blob . *top [i] .data* . **i** BLOB .
	blob . *bottom [i] .data* . **i** BLOB .

- Caffe

Caffe `WITH_PYTHON_LAYER` .

```
WITH_PYTHON_LAYER=1 make && make pycaffe
```

- ?

( ). Caffe (prototxt). , PYTHONPATH .

1. Christopher Bourez
2. Caffe Github
3.

Examples

```
import caffe

class My_Custom_Layer(caffe.Layer):
    def setup(self, bottom, top):
        pass

    def forward(self, bottom, top):
        pass

    def reshape(self, bottom, top):
        pass

    def backward(self, bottom, top):
        pass
```

:

- **caffe.Layer** (*Caffe*);
- : , , ;
- blob *top* *bottom* . *top [i] .data* *bottom [i] .data* . blob blob .

-

Caffe Setup . .

-

blob ( ) / . .

-

Forward .

-

Backward . , convolution-like . ( ).

Prototxt

. *.prototxt* .

```
layer {
  name: "LayerName"
  type: "Python"
  top: "TopBlobName"
  bottom: "BottomBlobName"
  python_param {
    module: "My_Custom_Layer_File"
    layer: "My_Custom_Layer_Class"
    param_str: '{"param1": 1,"param2":True, "param3":"some string"}'
  }
  include{
        phase: TRAIN
  }
}
```

:

- **type** **Python** .
- **python_param** .
- (*.py*) .
- **layer** .
- **param_str** ( ).
- ( ).

param_str *prototxt* . .

```
def setup(self, bottom, top):
    params = eval(self.param_str)
    param1 = params["param1"]
    param2 = params.get('param2', False) #I usually use this when fetching a bool
    param3 = params["param3"]

    #Continue with the setup
    # ...
```

"" / , . Caffe F .

measureLayer.py .

```
#Remark: This class is designed for a binary problem, where the first class would be the
'negative'
# and the second class would be 'positive'

import caffe
TRAIN = 0
TEST = 1

class Measure_Layer(caffe.Layer):
    #Setup method
    def setup(self, bottom, top):
        #We want two bottom blobs, the labels and the predictions
        if len(bottom) != 2:
            raise Exception("Wrong number of bottom blobs (prediction and label)")

        #And some top blobs, depending on the phase
        if self.phase = TEST and len(top) != 3:
            raise Exception("Wrong number of top blobs (acc, FPR, FNR)")
        if self.phase = TRAIN and len(top) != 5:
            raise Exception("Wrong number of top blobs (acc, tp, tn, fp and fn)")

        #Initialize some attributes
        self.TPs = 0.0
        self.TNs = 0.0
        self.FPs = 0.0
        self.FNs = 0.0
        self.totalImgs = 0

    #Forward method
    def forward(self, bottom, top):
        #The order of these depends on the prototxt definition
        predictions = bottom[0].data
        labels = bottom[1].data

        self.totalImgs += len(labels)

        for i in range(len(labels)): #len(labels) is equal to the batch size
                pred = predictions[i]   #pred is a tuple with the normalized probability
                                        #of a sample i.r.t. two classes
                lab = labels[i]

                if pred[0] > pred[1]:
                        if lab == 1.0:
                                self.FNs += 1.0
                        else:
                                self.TNs += 1.0
                else:
                        if lab == 1.0:
                                self.TPs += 1.0
                        else:
                                self.FPs += 1.0

        acc = (self.TPs + self.TNs) / self.totalImgs

        try: #just assuring we don't divide by 0
                fpr = self.FPs / (self.FPs + self.TNs)
        except:
                fpr = -1.0

        try: #just assuring we don't divide by 0
                fnr = self.FNs / (self.FNs + self.TPs)
        except:
                fnr = -1.0

       #output data to top blob
       top[0].data = acc
       if self.phase == TRAIN:
           top[1].data = self.TPs
           top[2].data = self.TNs
           top[3].data = self.FPs
           top[4].data = self.FNs
       elif self.phase == TEST:
           top[1].data = fpr
           top[2].data = fnr

    def reshape(self, bottom, top):
        """
        We don't need to reshape or instantiate anything that is input-size sensitive
        """
        pass

    def backward(self, bottom, top):
        """
        This layer does not back propagate
        """
        pass
```

prototxt :

```
layer {
  name: "metrics"
  type: "Python"
  top: "Acc"
  top: "TPs"
  top: "TNs"
  top: "FPs"
  top: "FNs"

  bottom: "prediction"   #let's supose we have these two bottom blobs
  bottom: "label"

  python_param {
    module: "measureLayer"
    layer: "Measure_Layer"
  }
  include {
    phase: TRAIN
  }
}

layer {
  name: "metrics"
  type: "Python"
  top: "Acc"
  top: "FPR"
  top: "FNR"

  bottom: "prediction"   #let's supose we have these two bottom blobs
  bottom: "label"

  python_param {
    module: "measureLayer"
    layer: "Measure_Layer"
  }
  include {
    phase: TEST
  }
}
```

, , . , Caffe .

dataLayer.py .

```
import caffe

class Custom_Data_Layer(caffe.Layer):
    def setup(self, bottom, top):
        # Check top shape
        if len(top) != 2:
                raise Exception("Need to define tops (data and label)")

        #Check bottom shape
        if len(bottom) != 0:
            raise Exception("Do not define a bottom.")

        #Read parameters
        params = eval(self.param_str)
        src_file = params["src_file"]
        self.batch_size = params["batch_size"]
        self.im_shape = params["im_shape"]
        self.crop_size = params.get("crop_size", False)

        #Reshape top
        if self.crop_size:
            top[0].reshape(self.batch_size, 3, self.crop_size, self.crop_size)
        else:
            top[0].reshape(self.batch_size, 3, self.im_shape, self.im_shape)

        top[1].reshape(self.batch_size)

        #Read source file
        #I'm just assuming we have this method that reads the source file
        #and returns a list of tuples in the form of (img, label)
        self.imgTuples = readSrcFile(src_file)
        self._cur = 0 #use this to check if we need to restart the list of imgs

    def forward(self, bottom, top):
        for itt in range(self.batch_size):
            # Use the batch loader to load the next image.
            im, label = self.load_next_image()

            #Here we could preprocess the image
            # ...

            # Add directly to the top blob
            top[0].data[itt, ...] = im
            top[1].data[itt, ...] = label

    def load_next_img(self):
        #If we have finished forwarding all images, then an epoch has finished
        #and it is time to start a new one
        if self._cur == len(self.imgTuples):
            self._cur = 0
            shuffle(self.imgTuples)

        im, label = self.imgTuples[self._cur]
        self._cur += 1

        return im, label

    def reshape(self, bottom, top):
        """
        There is no need to reshape the data, since the input is of fixed size
        (img shape and batch size)
        """
        pass

    def backward(self, bottom, top):
        """
        This layer does not back propagate
        """
        pass
```

prototxt .

```
layer {
  name: "Data"
  type: "Python"
  top: "data"
  top: "label"

  python_param {
    module: "dataLayer"
    layer: "Custom_Data_Layer"
    param_str: '{"batch_size": 126,"im_shape":256, "crop_size":224, "src_file":
"path_to_TRAIN_file.txt"}'
  }
}
```

: https://riptutorial.com/ko/caffe/topic/10535/--

S. No		Contributors
1	caffe	Community, dontloo, Franck Dernoncourt, GoodDeeds, Parag S. Chandakkar, Shai, Tahir Shahzad, Ujjwal Aryan
2	pycaffe Caffe	Franck Dernoncourt, Parag S. Chandakkar
3		dontloo, malreddysid, Shai, Stephen Leppik
4	Caffe - , ,	Ujjwal Aryan
5		dasWesen, Jonathan, Shai
6		Fernanda Andalo, rafaspadilha