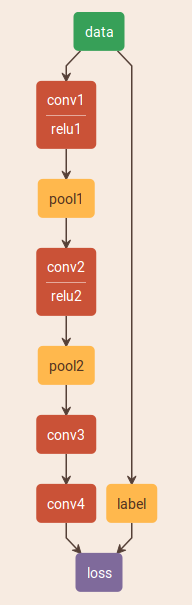
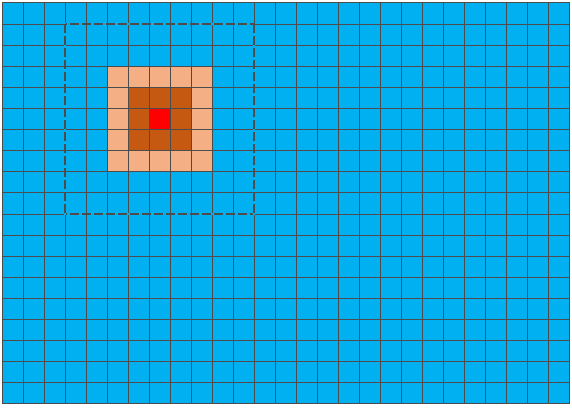
# 基于全卷积神经网络进行人脸检测

## 网络设计



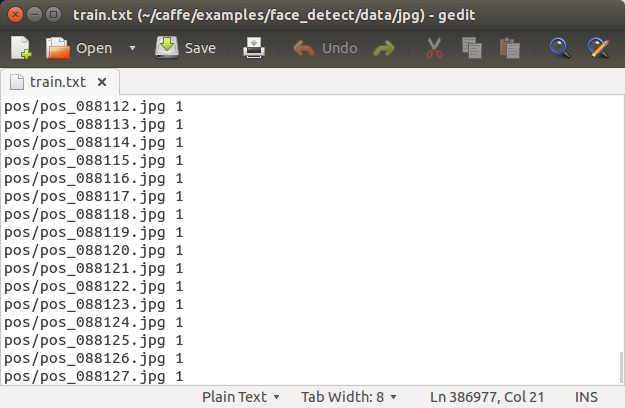
全卷积神经网络与一般的卷积神经网络不同，全卷积神经网络没有全连接层，最后的卷积层直接输入到损失层。对于二分类任务，最后的卷积层共有2个通道，每一个通道图像上的某个像素值，表示以该像素为中心，宽为W的窗口图像以属于该通道对应类别的概率，其中W为训练时所使用的样本宽度，即，训练样本为W\*W的图像。



在训练时，使用W\*W的样本，输入到网络中，卷积层4的输出为2\*1\*1的feature map，即2个通道，每个通道图像只有1个像素，每个像素的值代表属于该类的可能性，在0～1之间。

## 网络训练

使用78143张正样本和308848张负样本进行作为训练样本，10000张正样本和10000张负样本作为测试样本。转换成lmdb数据文件。





训练使用2000次迭代即可达到99%以上准确率。

生成lmdb数据、均值文件及训练步骤略。

## 测试

#include <opencv2/dnn.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/highgui.hpp>

using namespace cv;

using namespace cv::dnn;

#include <fstream>

#include <iostream>

#include <cstdlib>

using namespace std;

//g++ -o detect detect.cpp -lopencv\_dnn -lopencv\_highgui -lopencv\_imgcodecs -lopencv\_imgproc -lstdc++ -lopencv\_core -L/usr/local/lib

int main(int argc, char \*\*argv)

{

String modelTxt="deploy.prototxt";

String modelBin="face\_detect\_iter\_20000.caffemodel";

Ptr<dnn::Importer> importer;

try

{

importer=dnn::createCaffeImporter(modelTxt,modelBin);

}

catch(const cv::Exception &err)

{

std::cerr<<err.msg<<std::endl;

}

if(!importer)

{

std::cerr<<"cant load network!"<<std::endl;

return 0;

}

dnn::Net net;

importer->populateNet(net);

importer.release();

Mat img = imread("000001.jpg",1);

if(img.empty())

{

std::cerr<<"cant load image!"<<std::endl;

return 0;

}

dnn::Blob inputBlob = dnn::Blob(img);

net.setBlob(".data",inputBlob);

net.forward();

dnn::Blob prob=net.getBlob("conv4");

imshow("src",img);

for(int i=0;i<2;i++)

{

Mat featureImg=prob.getPlane(0,i);

featureImg.convertTo(featureImg,CV\_8UC1);

resize(featureImg, featureImg,img.size());

Mat heatmap;

applyColorMap(featureImg, heatmap, COLORMAP\_JET);

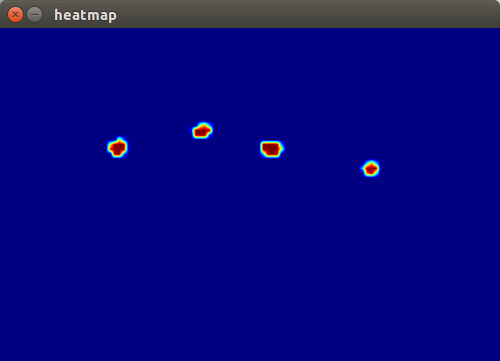
imshow("heatmap", heatmap);

waitKey(0);

}

}





说明：

以上得到的红色区域给出了人脸的位置，但并没有给出人脸的大小。人脸大小可以按照以下方式得到。

将原图像以建立图像金字塔，将金字塔每一层都输入到网络中，对于每一个层都可以得到一个heatmap，然后可以画出一些人脸框，将这些图像再归一化到原图像大小，将这些缩放了的框进行合并。

## 相关文件

### 4.1 train\_val.prototxt

name: "CaffeNet"

layer {

name: "data"

type: "Data"

top: "data"

top: "label"

include {

phase: TRAIN

}

transform\_param {

mirror: true

crop\_size: 227

mean\_file: "examples/cat\_dog/data/mean.binaryproto"

}

data\_param {

source: "examples/cat\_dog/data/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\_file: "examples/cat\_dog/data/mean.binaryproto"

}

data\_param {

source: "examples/cat\_dog/data/lmdb/val\_lmdb"

batch\_size: 50

backend: LMDB

}

}

layer {

name: "conv1"

type: "Convolution"

bottom: "data"

top: "conv1"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 96

kernel\_size: 11

stride: 4

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu1"

type: "ReLU"

bottom: "conv1"

top: "conv1"

}

layer {

name: "pool1"

type: "Pooling"

bottom: "conv1"

top: "pool1"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "norm1"

type: "LRN"

bottom: "pool1"

top: "norm1"

lrn\_param {

local\_size: 5

alpha: 0.0001

beta: 0.75

}

}

layer {

name: "conv2"

type: "Convolution"

bottom: "norm1"

top: "conv2"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 2

kernel\_size: 5

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu2"

type: "ReLU"

bottom: "conv2"

top: "conv2"

}

layer {

name: "pool2"

type: "Pooling"

bottom: "conv2"

top: "pool2"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "norm2"

type: "LRN"

bottom: "pool2"

top: "norm2"

lrn\_param {

local\_size: 5

alpha: 0.0001

beta: 0.75

}

}

layer {

name: "conv3"

type: "Convolution"

bottom: "norm2"

top: "conv3"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 384

pad: 1

kernel\_size: 3

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu3"

type: "ReLU"

bottom: "conv3"

top: "conv3"

}

layer {

name: "conv4"

type: "Convolution"

bottom: "conv3"

top: "conv4"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 384

pad: 1

kernel\_size: 3

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu4"

type: "ReLU"

bottom: "conv4"

top: "conv4"

}

layer {

name: "conv5"

type: "Convolution"

bottom: "conv4"

top: "conv5"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 256

pad: 1

kernel\_size: 3

group: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu5"

type: "ReLU"

bottom: "conv5"

top: "conv5"

}

layer {

name: "pool5"

type: "Pooling"

bottom: "conv5"

top: "pool5"

pooling\_param {

pool: MAX

kernel\_size: 3

stride: 2

}

}

layer {

name: "fc6"

type: "InnerProduct"

bottom: "pool5"

top: "fc6"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 4096

weight\_filler {

type: "gaussian"

std: 0.005

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu6"

type: "ReLU"

bottom: "fc6"

top: "fc6"

}

layer {

name: "drop6"

type: "Dropout"

bottom: "fc6"

top: "fc6"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

name: "fc7"

type: "InnerProduct"

bottom: "fc6"

top: "fc7"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 4096

weight\_filler {

type: "gaussian"

std: 0.005

}

bias\_filler {

type: "constant"

value: 1

}

}

}

layer {

name: "relu7"

type: "ReLU"

bottom: "fc7"

top: "fc7"

}

layer {

name: "drop7"

type: "Dropout"

bottom: "fc7"

top: "fc7"

dropout\_param {

dropout\_ratio: 0.5

}

}

layer {

name: "my\_fc8"

type: "InnerProduct"

bottom: "fc7"

top: "my\_fc8"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

inner\_product\_param {

num\_output: 2

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "accuracy"

type: "Accuracy"

bottom: "my\_fc8"

bottom: "label"

top: "accuracy"

include {

phase: TEST

}

}

layer {

name: "loss"

type: "SoftmaxWithLoss"

bottom: "my\_fc8"

bottom: "label"

top: "loss"

}

### 4.2 deploy.prototxt

name: "FVLN"

input: "data"

input\_shape {

dim: 1 # batchsize

dim: 1 # number of colour channels - rgb

dim: 500 # width

dim: 333 # height

}

layer {

name: "conv1"

type: "Convolution"

bottom: "data"

top: "conv1"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 32

kernel\_size: 5

stride: 1

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu1"

type: "ReLU"

bottom: "conv1"

top: "conv1"

}

layer {

name: "pool1"

type: "Pooling"

bottom: "conv1"

top: "pool1"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

name: "conv2"

type: "Convolution"

bottom: "pool1"

top: "conv2"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 64

kernel\_size: 5

stride: 1

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "relu2"

type: "ReLU"

bottom: "conv2"

top: "conv2"

}

layer {

name: "pool2"

type: "Pooling"

bottom: "conv2"

top: "pool2"

pooling\_param {

pool: MAX

kernel\_size: 2

stride: 2

}

}

layer {

name: "conv3"

type: "Convolution"

bottom: "pool2"

top: "conv3"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 64

kernel\_size: 3

stride: 1

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer {

name: "conv4"

type: "Convolution"

bottom: "conv3"

top: "conv4"

param {

lr\_mult: 1

decay\_mult: 1

}

param {

lr\_mult: 2

decay\_mult: 0

}

convolution\_param {

num\_output: 2

kernel\_size: 10

stride: 1

weight\_filler {

type: "gaussian"

std: 0.01

}

bias\_filler {

type: "constant"

value: 0

}

}

}

layer{

name:"prob"

type:"Softmax"

bottom:"conv4"

top:"prob"

}

### 4.3 solver.prototxt

net: "/home/ermu/caffe/examples/face\_detect/net/train\_FVPN.prototxt"

test\_iter: 100

test\_interval: 500

base\_lr: 0.01

lr\_policy: "step"

gamma: 0.1

stepsize: 20000

display: 100

max\_iter: 100000

momentum: 0.9

weight\_decay: 0.0005

snapshot: 10000

snapshot\_prefix: "/home/ermu/caffe/examples/face\_detect/result/face\_detect"

solver\_mode: GPU