Face recognition with convoluted neural networks - OT1-PJ12

FOSSART Alexis, RENAULT Benoit

January 31st 2018

Table des matières

1 Introduction

2 Running the project

2.1 Prerequisites

Please run it preferably on a GNU/Linux distribution. You should also have installed the latest version available to you of Git and Docker (google them to find their installation instructions for your OS).

2.2 Quickstart

Clone this repository :

```
git clone
```

3 Description of the work

3.1 Setting up Caffe to be used through PyCaffe and Docker

3.2 Training the network

3.2.1 Code

```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import caffe

caffe.set_mode_cpu()
solver = caffe.SGDSolver('solver.prototxt')
solver.solve()
```

3.2.2 Adjusting the Docker image for better performances during training

3.3 Evaluating the detection performance of our NN

3.3.1 Code

Basic setup of neural network

```python
import caffe

caffe.set_mode_cpu()

model_train = 'conv.prototxt'
model_test = 'deploy.prototxt'
weights = 'deep_iter_100000.caffemodel'

net_training = caffe.Net(model_train, weights, caffe.TEST)
net_testing = caffe.Net(model_test, weights, caffe.TEST)
```

Compute the accuracy of the previously trained neural network on the training dataset (see 01_CNN_Training notebook) by using the accuracy layer of the network (defined in conv.prototxt)

```
NUMBER_OF_IMAGES_ON_TRAINING_SET_WITHOUT_FACE = 26950
NUMBER_OF_IMAGES_ON_TRAINING_SET_WITH_A_FACE = 64770

TOTAL_NUMBER_OF_IMAGES_ON_TRAINING_SET = (NUMBER_OF_IMAGES_ON_TRAINING_SET_WITHOUT_FACE +
                                          NUMBER_OF_IMAGES_ON_TRAINING_SET_WITH_A_FACE)

total_accuracy = 0
batch_size = net_training.blobs['data'].num
test_iters = TOTAL_NUMBER_OF_IMAGES_ON_TRAINING_SET / batch_size

for i in range(test_iters):
    net_training.forward()
    batch_accuracy = net_training.blobs['accuracy'].data
    total_accuracy += batch_accuracy
accuracy = total_accuracy / test_iters

print "Accuracy of trained network on train data: {}".format(accuracy)
print "Number of well classified images on train data: {}".format(
    int(accuracy * TOTAL_NUMBER_OF_IMAGES_ON_TRAINING_SET))
```

Evaluate the accuracy of the previously trained neural network on the test dataset

```
NUMBER_OF_IMAGES_ON_GOOGLEFACE_TEST = 632
NUMBER_OF_IMAGES_ON_GOOGLE_IMAGES = 6831
NUMBER_OF_IMAGES_ON_YALEFACES_TEST = 165

TOTAL_NUMBER_OF_IMAGES_ON_TEST_SET = (NUMBER_OF_IMAGES_ON_YALEFACES_TEST +
                                      NUMBER_OF_IMAGES_ON_GOOGLE_IMAGES +
                                      NUMBER_OF_IMAGES_ON_GOOGLEFACE_TEST)

img_classified_as_faces = 0
img_classified = 0

total_accuracy = 0

batch_size = net_testing.blobs['data'].num
test_iters = TOTAL_NUMBER_OF_IMAGES_ON_TEST_SET / batch_size

for i in range(test_iters):
    net_testing.forward()
    batch_accuracy = net_testing.blobs['accuracy'].data
    total_accuracy += batch_accuracy
    for i in range(batch_size):
        img_classified += 1
        if net_testing.blobs['prob'].data[i].argmax():
            img_classified_as_faces += 1


accuracy = total_accuracy / test_iters
```

```
print "Number of faces found on test : {} / {} images".format(img_classified_as_faces,
                                                              img_classified)
print "Accuracy of trained network on test data: {}".format(accuracy)
print "Number of well classified images on test data: {}".format(
    int(accuracy * TOTAL_NUMBER_OF_IMAGES_ON_TEST_SET))
```

3.4 Implementing a naive face detector

Basic setup of neural network

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import cv2
import caffe
import os

caffe.set_mode_cpu()
model = 'deploy.prototxt'
weights = 'deep_iter_100000.caffemodel'
net = caffe.Net(model, weights, caffe.TEST)

try:
    os.mkdir("../data/results/")
except OSError: pass
try:
    os.mkdir("../data/results/1/")
except OSError: pass
```

Implementation of face detector Downscaling done by factor of 2 (using an integrated function of opencv which seems to be the only way to do pyramid scaling without destroying the quality of the images). Naive algorithm implementation, without heuristics : we simply use a shifting window of size 36*36px and offset 4px.

```
for id_img in range(1, 8):
    # Used on 7 images named 1.jpg to 7.jpg
    image_path = '../data/' + str(id_img) + '.jpg'

    # Open image and convert to gray scale
    im =  cv2.imread(image_path)
    im = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)

    # Save base image for future comparison
    imbase = im

    scale = 1

    base_save_path = "../data/results/"
    img_base_name = str(id_img) + "_"

    # Used to batch save detected faces, and avoid writing to disk at every loop
    keep = dict()
```

```python
# Tuple with (width, height)
shifting_window_size = 36, 36
offset = 4

min_probability_for_a_match = 0.99

while len(im) >= shifting_window_size[0] * 2 and len(im[0]) > shifting_window_size[1] * 2:
    # Downscale of the image using opencv pyramid scaling
    im = cv2.pyrDown(im)
    scale*=2

    img_scale_name = img_base_name + str(scale) + "_"

    showarray(im)

    # Face detector
    for x in range(0, len(im) - shifting_window_size[0], offset):
        for y in range(0, len(im[0]) - shifting_window_size[1], offset):
            img_name = img_scale_name + str(x) + "_" + str(y) + ".png"

            # Create the shift window image on the original scaled image
            imtmp = np.array(im [x:x + shifting_window_size[0], y:y + shifting_window_size[1]])

            # Transform the data to be compatible as an entry to the neural network
            im_input = imtmp[np.newaxis, np.newaxis, :, :] / 256.0

            # Input the data in the NN
            net.blobs['data'].reshape(*im_input.shape)
            net.blobs['data'].data[...] = imtmp

            # Run the NN
            output = net.forward()

            # If the probability that the image that is in the
            #shift window is a face is higher than 99%
            if output['prob'][0][1] > min_probability_for_a_match:
                # We add a bounding box in the original image
                white = 255
                for i in range (x * scale, (x + shifting_window_size[0]) * scale):
                    imbase[i][y * scale] = white
                    imbase[i][(y + shifting_window_size[1]) * scale] = white
                for j in range(y * scale, (y + shifting_window_size[1]) * scale):
                    imbase[x * scale][j] = white
                    imbase[(x + shifting_window_size[0]) * scale][j] = white

                # Save this in memory
                save_dir = "1/"
                save_path = base_save_path + save_dir + img_name
                keep[save_path] = imtmp

# When everything is done we show the original image
```

```
#with the bounding boxes and save it as a file
showarray(imbase)
save_results = "../data/results/" + str(id_img) + ".jpg"
cv2.imwrite(save_results, imbase)

# We also save the face subimages we found to improve the NN in the future
for path,img in keep.iteritems():
    cv2.imwrite(path, img)
```

4 Conclusion