```
# This script is used to estimate an accuracy of different face detection models.
# COCO evaluation tool is used to compute an accuracy metrics (Average Precision).
# Script works with different face detection datasets.
import os
import json
from fnmatch import fnmatch
from math import pi
import cv2 as cv
import argparse
import os
import sys
from pycocotools.coco import COCO
from pycocotools.cocoeval import COCOeval

parser = argparse.ArgumentParser(
        description='Evaluate OpenCV face detection algorithms '
                    'using COCO evaluation tool, http://cocodataset.org/#detections-eval')
parser.add_argument('--proto', help='Path to .prototxt of Caffe model or .pbtxt of TensorFlow graph')
parser.add_argument('--model', help='Path to .caffemodel trained in Caffe or .pb from TensorFlow')
parser.add_argument('--cascade', help='Optional path to trained Haar cascade as '
                                      'an additional model for evaluation')
parser.add_argument('--ann', help='Path to text file with ground truth annotations')
parser.add_argument('--pics', help='Path to images root directory')
parser.add_argument('--fddb', help='Evaluate FDDB dataset, http://vis-www.cs.umass.edu/fddb/',
action='store_true')
parser.add_argument('--wider', help='Evaluate WIDER FACE dataset,
http://mmlab.ie.cuhk.edu.hk/projects/WIDERFace/', action='store_true')
args = parser.parse_args()

dataset = {}
dataset['images'] = []
dataset['categories'] = [{ 'id': 0, 'name': 'face' }]
dataset['annotations'] = []

def ellipse2Rect(params):
    rad_x = params[0]
    rad_y = params[1]
    angle = params[2] * 180.0 / pi
    center_x = params[3]
    center_y = params[4]
    pts = cv.ellipse2Poly((int(center_x), int(center_y)), (int(rad_x), int(rad_y)),
                          int(angle), 0, 360, 10)
    rect = cv.boundingRect(pts)
    left = rect[0]
    top = rect[1]
    right = rect[0] + rect[2]
    bottom = rect[1] + rect[3]
    return left, top, right, bottom

def addImage(imagePath):
    assert('images' in  dataset)
    imageId = len(dataset['images'])
    dataset['images'].append({
        'id': int(imageId),
        'file_name': imagePath
    })
    return imageId

def addBBox(imageId, left, top, width, height):
    assert('annotations' in  dataset)
    dataset['annotations'].append({
        'id': len(dataset['annotations']),
        'image_id': int(imageId),
        'category_id': 0,  # Face
```

```
        'bbox': [int(left), int(top), int(width), int(height)],
        'iscrowd': 0,
        'area': float(width * height)
    })

def addDetection(detections, imageId, left, top, width, height, score):
    detections.append({
      'image_id': int(imageId),
      'category_id': 0,  # Face
      'bbox': [int(left), int(top), int(width), int(height)],
      'score': float(score)
    })


def fddb_dataset(annotations, images):
    for d in os.listdir(annotations):
        if fnmatch(d, 'FDDB-fold-*-ellipseList.txt'):
            with open(os.path.join(annotations, d), 'rt') as f:
                lines = [line.rstrip('\n') for line in f]
                lineId = 0
                while lineId < len(lines):
                    # Image
                    imgPath = lines[lineId]
                    lineId += 1
                    imageId = addImage(os.path.join(images, imgPath) + '.jpg')

                    img = cv.imread(os.path.join(images, imgPath) + '.jpg')

                    # Faces
                    numFaces = int(lines[lineId])
                    lineId += 1
                    for i in range(numFaces):
                        params = [float(v) for v in lines[lineId].split()]
                        lineId += 1
                        left, top, right, bottom = ellipse2Rect(params)
                        addBBox(imageId, left, top, width=right - left + 1,
                                height=bottom - top + 1)


def wider_dataset(annotations, images):
    with open(annotations, 'rt') as f:
        lines = [line.rstrip('\n') for line in f]
        lineId = 0
        while lineId < len(lines):
            # Image
            imgPath = lines[lineId]
            lineId += 1
            imageId = addImage(os.path.join(images, imgPath))

            # Faces
            numFaces = int(lines[lineId])
            lineId += 1
            for i in range(numFaces):
                params = [int(v) for v in lines[lineId].split()]
                lineId += 1
                left, top, width, height = params[0], params[1], params[2], params[3]
                addBBox(imageId, left, top, width, height)

def evaluate():
    cocoGt = COCO('annotations.json')
    cocoDt = cocoGt.loadRes('detections.json')
    cocoEval = COCOeval(cocoGt, cocoDt, 'bbox')
    cocoEval.evaluate()
    cocoEval.accumulate()
    cocoEval.summarize()
```

```
### Convert to COCO annotations format ###########################################
assert(args.fddb or args.wider)
if args.fddb:
    fddb_dataset(args.ann, args.pics)
elif args.wider:
    wider_dataset(args.ann, args.pics)

with open('annotations.json', 'wt') as f:
    json.dump(dataset, f)

### Obtain detections ############################################################
detections = []
if args.proto and args.model:
    net = cv.dnn.readNet(args.proto, args.model)

    def detect(img, imageId):
        imgWidth = img.shape[1]
        imgHeight = img.shape[0]
        net.setInput(cv.dnn.blobFromImage(img, 1.0, (300, 300), (104., 177., 123.), False, False))
        out = net.forward()

        for i in range(out.shape[2]):
            confidence = out[0, 0, i, 2]
            left = int(out[0, 0, i, 3] * img.shape[1])
            top = int(out[0, 0, i, 4] * img.shape[0])
            right = int(out[0, 0, i, 5] * img.shape[1])
            bottom = int(out[0, 0, i, 6] * img.shape[0])

            x = max(0, min(left, img.shape[1] - 1))
            y = max(0, min(top, img.shape[0] - 1))
            w = max(0, min(right - x + 1, img.shape[1] - x))
            h = max(0, min(bottom - y + 1, img.shape[0] - y))

            addDetection(detections, imageId, x, y, w, h, score=confidence)

elif args.cascade:
    cascade = cv.CascadeClassifier(args.cascade)

    def detect(img, imageId):
        srcImgGray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        faces = cascade.detectMultiScale(srcImgGray)

        for rect in faces:
            left, top, width, height = rect[0], rect[1], rect[2], rect[3]
            addDetection(detections, imageId, left, top, width, height, score=1.0)

for i in range(len(dataset['images'])):
    sys.stdout.write('\r%d / %d' % (i + 1, len(dataset['images'])))
    sys.stdout.flush()

    img = cv.imread(dataset['images'][i]['file_name'])
    imageId = int(dataset['images'][i]['id'])

    detect(img, imageId)

with open('detections.json', 'wt') as f:
    json.dump(detections, f)

evaluate()


def rm(f):
    if os.path.exists(f):
```

```
        os.remove(f)

rm('annotations.json')
rm('detections.json')
```