Traffic Sign Detection

Lewis Collum

Updated: May 12, 2020

Started 02/18/2020

- ☐ Add moving .ppm images from data to data/images_ppm to script
- ☐ Complete 'Path Organization Section'

Downloading the German Traffic Sign Detection Benchmark, GTSDB

The following downloads the dataset, unzips it, and moves the internal directories around (and changes their name) to match my prefered directory convention.

```
setName="FullIJCNN2013"
set=$setName.zip
dataDirectory="data"

function main() {
    #downloadSet
    #unzipSet
    configureSet
    echo "DONE!"
}

function downloadSet() {
    if [[ ! -f $set ]]; then
        wget https://sid.erda.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/$set
    fi;
}

function unzipSet() {
    if [[ -f $set && ! -d $dataDirectory ]]; then
        unzip $set
        mv $setName $dataDirectory
    fi
}

function configureSet() {
    sed 's/;/,/g' $dataDirectory/gt.txt > $dataDirectory/gt.csv
}

main $@
```

Creating Darknet Annotations using `gt.txt`

We need to convert the `gt.txt` we are given with GTSDB, to a `labels` directory, with a per image file containing the class and the bounding box information.

So, from

```
00000.ppm;774;411;815;446;11
```

to this, in label directory file `00000.txt`

```
11 <normalized center x> <normalized center y> <normalized width> <normalized height>
```

```
import os
import pandas

data = "../data"
obj = f"{data}/obj"
images = f"{data}/images_jpg"

gt = pandas.read_csv(f"{data}/gt.txt", sep=';', names=['file', 'x1', 'y1', 'x2', 'y2', 'class'])
imageWidth = 1360
imageHeight = 800


if not os.path.exists(obj):
    os.makedirs(obj)

for i in os.listdir(images):
    imageName = os.path.splitext(i)[0]
    annotation = f"{obj}/{imageName}.txt"
    with open(annotation, 'w'): pass

for index, row in gt.iterrows():
    with open(f"{obj}/{os.path.splitext(row['file'])[0]}.txt", 'a') as f:
        boxWidth = (row['x2']-row['x1'])/imageWidth
        boxHeight = (row['y2']-row['y1'])/imageHeight
        boxXCenter = row['x1']/imageWidth + boxWidth/2
        boxYCenter = row['y1']/imageHeight + boxHeight/2

        f.write(f"{row['class']} {boxXCenter} {boxYCenter} {boxWidth} {boxHeight}\n")
```

Converting PPM Images to JPG

```
from PIL import Image
import os

ppms = "images_ppm"
jpgs = "images_jpg"

if not os.path.isdir(jpgs): os.makedirs(jpgs)
for imageName in os.listdir(ppms):
```

```
    image = Image.open(f"{ppms}/{imageName}")
    image.save(f"{jpgs}/{os.path.splitext(imageName)[0]}.jpg")
```

Copying JPG Images with Corresponding Annotations to the `obj` Directory for Training

```
import os
import shutil

data = "../data"
source = f"{data}/images_jpg"
destination = f"{data}/obj"

for name in [os.path.splitext(i)[0] for i in os.listdir(destination)]:
    if os.path.isfile(f"{destination}/{name}.txt"):
        shutil.copyfile(f"{source}/{name}.jpg", f"{destination}/{name}.jpg")
```

Splitting Training and Testing Images

```
import random
import os
import subprocess
import sys

f_val = open("../data/test.txt", 'w')
f_train = open("../data/train.txt", 'w')

images = "../data/obj"

data_size = len(os.listdir(images))
data_test_size = int(0.2 * data_size)
test_array = random.sample(range(data_size), k=data_test_size)

for ind, f in enumerate(os.listdir(images)):
    if(f.split(".")[1] == "jpg"):
        if ind in test_array:
            f_val.write(os.path.abspath(images)+'/'+f+'\n')
        else:
            f_train.write(os.path.abspath(images)+'/'+f+'\n')

f_val.close()
f_train.close()
```

Distribution of Classes

Training Script

```
~/darknet/darknet detector train ../data/sign.data ../cfg/yolov3-tiny-prn.cfg ../cfg/yolov3-tiny.conv.
```

Test Single Image

```
~/darknet/darknet detector test ../data/sign.data ../cfg/my-lite.cfg ../weights/my-lite_last.weights $
```

Accuracy Mapping

```
~/darknet/darknet detector map ../data/sign.data ../cfg/my-lite.cfg ../weights/my-lite_last.weights -i
```

Run Through OpenCV

```
# Usage example:  python3 object_detection_yolo.py --video=run.mp4
#                 python3 object_detection_yolo.py --image=bird.jpg

import cv2 as cv
import argparse
import sys
import numpy as np
import os.path

confThreshold = 0.2  #Confidence threshold
nmsThreshold = 0.4   #Non-maximum suppression threshold
inpWidth = 416       #Width of network's input image
inpHeight = 416      #Height of network's input image

parser = argparse.ArgumentParser(description='Object Detection using YOLO in OPENCV')
parser.add_argument('--image', help='Path to image file.')
parser.add_argument('--video', help='Path to video file.')
parser.add_argument('--nosave', nargs='?', const=True, default=False, help='Do not save the output')
args = parser.parse_args()

# Load names of classes
classesFile = "../data/sign.names"
classes = None
with open(classesFile, 'rt') as f:
    classes = f.read().rstrip('\n').split('\n')

# Give the configuration and weight files for the model and load the network using them.
modelConfiguration = "../cfg/my-lite.cfg"
modelWeights = "../weights/my-lite_last.weights"
```

```
net = cv.dnn.readNetFromDarknet(modelConfiguration, modelWeights)
net.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)

# Get the names of the output layers
def getOutputsNames(net):
    # Get the names of all the layers in the network
    layersNames = net.getLayerNames()
    # Get the names of the output layers, i.e. the layers with unconnected outputs
    return [layersNames[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# Draw the predicted bounding box
def drawPred(classId, conf, left, top, right, bottom):
    # Draw a bounding box.
    cv.rectangle(frame, (left, top), (right, bottom), (255, 178, 50), 3)

    label = '%.2f' % conf

    # Get the label for the class name and its confidence
    if classes:
        assert(classId < len(classes))
        label = '%s:%s' % (classes[classId], label)

    #Display the label at the top of the bounding box
    labelSize, baseLine = cv.getTextSize(label, cv.FONT_HERSHEY_SIMPLEX, 0.5, 1)
    top = max(top, labelSize[1])
    cv.rectangle(frame, (left, top - round(1.5*labelSize[1])), (left + round(1.5*labelSize[0]), top +
    cv.putText(frame, label, (left, top), cv.FONT_HERSHEY_SIMPLEX, 0.75, (0,0,0), 1)

# Remove the bounding boxes with low confidence using non-maxima suppression
def postprocess(frame, outs):
    frameHeight = frame.shape[0]
    frameWidth = frame.shape[1]

    # Scan through all the bounding boxes output from the network and keep only the
    # ones with high confidence scores. Assign the box's class label as the class with the highest sco
    classIds = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            classId = np.argmax(scores)
            confidence = scores[classId]
            if confidence > confThreshold:
                center_x = int(detection[0] * frameWidth)
                center_y = int(detection[1] * frameHeight)
                width = int(detection[2] * frameWidth)
                height = int(detection[3] * frameHeight)
                left = int(center_x - width / 2)
                top = int(center_y - height / 2)
                classIds.append(classId)
```

```
                confidences.append(float(confidence))
                boxes.append([left, top, width, height])

    # Perform non maximum suppression to eliminate redundant overlapping boxes with
    # lower confidences.
    indices = cv.dnn.NMSBoxes(boxes, confidences, confThreshold, nmsThreshold)
    for i in indices:
        i = i[0]
        box = boxes[i]
        left = box[0]
        top = box[1]
        width = box[2]
        height = box[3]
        drawPred(classIds[i], confidences[i], left, top, left + width, top + height)

# Process inputs
winName = 'Deep learning object detection in OpenCV'
cv.namedWindow(winName, cv.WINDOW_NORMAL)

outputFile = "../demo/yolo_out_py.avi"
if (args.image):
    # Open the image file
    if not os.path.isfile(args.image):
        print("Input image file ", args.image, " doesn't exist")
        sys.exit(1)
    cap = cv.VideoCapture(args.image)
    inputFileName = os.path.splitext(os.path.split(args.image)[1])[0]
    outputFile = f'../demo/{inputFileName}.jpg'
elif (args.video):
    # Open the video file
    if not os.path.isfile(args.video):
        print("Input video file ", args.video, " doesn't exist")
        sys.exit(1)
    cap = cv.VideoCapture(args.video)
    inputFileName = os.path.splitext(os.path.split(args.video)[1])[0]
    outputFile = f'../demo/{inputFileName}.avi'
else:
    # Webcam input
    cap = cv.VideoCapture(0)

# Get the video writer initialized to save the output video
if (not args.image and not args.nosave):
    vid_writer = cv.VideoWriter(
        outputFile,
        cv.VideoWriter_fourcc('M','J','P','G'),
        30,
        (round(cap.get(cv.CAP_PROP_FRAME_WIDTH)), round(cap.get(cv.CAP_PROP_FRAME_HEIGHT))))

while cv.waitKey(33) != ord('q'):
    hasFrame, frame = cap.read()

    # Stop the program if reached end of video
```

```
    if not hasFrame:
        print("Done processing !!!")
        print("Output file is stored as ", outputFile)
        cv.waitKey(3000)
        # Release device
        cap.release()
        break

    # Create a 4D blob from a frame.
    blob = cv.dnn.blobFromImage(frame, 1/255, (inpWidth, inpHeight), [0,0,0], 1, crop=False)

    # Sets the input to the network
    net.setInput(blob)

    # Runs the forward pass to get output of the output layers
    outs = net.forward(getOutputsNames(net))

    # Remove the bounding boxes with low confidence
    postprocess(frame, outs)

    # Put efficiency information. The function getPerfProfile returns the overall time for inference(t
    t, _ = net.getPerfProfile()
    label = 'Inference time: %.2f ms' % (t * 1000.0 / cv.getTickFrequency())
    cv.putText(frame, label, (0, 15), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255))

    # Write the frame with the detection boxes
    if (not args.nosave):
        if (args.image):
            cv.imwrite(outputFile, frame.astype(np.uint8))
        else:
            vid_writer.write(frame.astype(np.uint8))

    cv.imshow(winName, frame)
```

Making CLAHE data

```
import os
import cv2

data = "../data"
source = f"{data}/images_jpg"
destination = f"{data}/images_clahe"

clahe = cv2.createCLAHE(tileGridSize=(2,2), clipLimit=15.0)

for i in os.listdir(source):
```

Resources

- https://github.com/AlexeyAB/darknet

- https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/
- https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/