**Object Detection in image and video – How to use YOLO**

**1. Background:**

* **Computer Vision:** Computer Vision is consists of various aspects such as image recognition, object detection, image generation, image super-resolution and many more. In these tasks, object detection is widely used for face detection, vehicle detection, pedestrian counting, web images, security systems and self-driving cars. As of now, there are lots of highly accurate object detection-algorithms in recent years such as R-CNN, Fast-RCNN, Faster-RCNN, RetinaNet and fast yet highly accurate ones like SSD and YOLO. Using these methods and algorithms, we can choose deep learning frameworks such as TensorFlow, OpenCV, Caffe etc. We can detect each and every object in image by the area object in highlighted rectangular boxes and identify each and every object and assign its tag to the object. This also includes the accuracy of each method for identifying objects.
* **Object Detection:** detect multiple objects within an image, with bounding boxes. Object detection is more complex than image classification as there might be different number of objects for each image. In image classification, there is only one object to be identified and with bounding box. Object detection might have various outputs for each input image.
* **Algorithms:** CNN/R-CNN/Fast-RCNN/Faster-RCNN/SSD/YOLO
  + CNN: it emerged from the study of the brain’s visual cortex, and they have been used in image recognition since the 1980s. In the last few years, thanks to the increase in computational power, CNN have managed to achieve superhuman performance on some complex visual tasks. CNN architecture includes a series of convolution layers + ReLU, and pooling layers, followed by a number of fully connected layers. there are some case studies on CNN such as LeNet-5, AlexNet, VGG, etc. The LetNet-5 architecture can successfully classify this kind of 32\*32 pixel greyscale input images but had difficulty to precess higher resolution images. AlexNet has a very similar architecture as LeNet but was deeper, with more filters per layer, and with stacked convolutional layers. VGGNet won the localization task in ILSVRC 2014. its main contribution was in showing that the depth of the network is a critical component for good performance. A remarkable thing about the VGG-16 net is that it really simplified the neural network architectures.
  + R-CNN: in practice, how to detect object on an image? an intuitive idea is to use sliding window with CNNs to detect the objects. A sliding window is a rectangular region of fixed width and height that ‘slides’ across an image. For each of these window, we would normally take the window region and apply an image classifier to determine if the window has an object that we want to detect. But this takes a very high computational cost. So we don’t use this brute force sliding window approach. Regional proposal network was come up with to overcome the computational issue in the sliding window approach. Rather than sliding the window across the entire image to search every position, a region proposal network will look for the probable regions in the input image and give us some candidate proposal regions. R-CNN uses selective search to general about 2000 region proposals, then, each region was run through ConvNet.
  + Fast R-CNN: although R-CNN was big progress in the object detection filed, but it also has some drawbacks. The training is super slow. Selective algorithm is a fixed algorithm. We are not able to update them through learning. While, Fast R-CNN builds on previous work to efficiently classify object proposals using deep convolutional networks. Fast R-CNN also uses an algorithm like Edge Boxes to generate region proposals. Unlike the R-CNN detector, which crops and resizes region proposals, the Fast R-CNN detector processes the entire image. Whereas an R-CNN detector must classify each region, Fast R-CNN pools CNN features corresponding to each region proposal. Fast R-CNN is more efficient than R-CNN, because in the Fast R-CNN detector, the computations for overlapping regions are shared.
  + Faster R-CNN: compared to R-CNN and Fast R-CNN, Faster R-CNN adds a region proposal network (RPN) to generate region proposals directly in the network instead of using an external algorithm like Edge Boxes. Fast R-CNN is faster than R-CNN, Faster R-CNN is even faster.
  + SSD: above algorithms are region-based algorithms. But there is another family of method which is not region-based. The typical algorithms are SSD&YOLO. Both methods came out around the same time. SSD has two components: a backbone model and SSD head. *Backbone* model usually is a pre-trained image classification network as a feature extractor. The SSD head is just one or more convolutional layers added to this backbone and the outputs are interpreted as the bounding boxes and classes of objects in the spatial location of the final layers activations.
  + YOLO: Yolo is fast, accurate and suitable for real time detecting. It predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection is single network, it can be optimized end-to-end directly on detection performance. Yolo takes an image and divides the image into an S\*S grid. Each grid cell predicts B bounding boxes. Each boundary box contains 5 elements: (x, y, w, h) and a box confidence score. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that is predicts. Each grid cell also predicts C conditional class probabilities which are the probability that the detected object belongs to a particular class.

**2. Requirement:**

**2.1** **Technical Dependences:**

* Ubuntu, Python, Deep learning

**2.2 Datasets:**

* **COCO dataset**: COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features: object segmentation, recognition in context, superpixel stuff segmentation, 330k images, 1.5 million object instance, 80 object categories, 91 stuff categories, 5 captions per image and 250000 people with keypoints.
* **PASCAL VOC:** this dataset includes 20 classes. The train/val data has 11,530 images containing 27,450 ROI annotated objects and 6,929 segmentations.
* **CelebA:** This dataset contains color face images with 40 attribute annotations for each image. The dataset can be used for different computer vision tasks including face detection, face attribute recognition and landmark or facial part localization.

**3. Process: image recognition**

The detector code has been stored in AWS server. To click Home, and then open test folder that includes all image detectors as follows:

* darknet: it is the original folder downloaded from github as YOLO is open source.
* darknet­\_test: image detector with heatmap on original image
* darknet\_video: video detector
* darknet\_image\_gender: gender detector
* darknet\_test\_heatmap: image detector with heatmap on layout of store
* darknet\_logo: logo detector

**3.1 darknet**

* Open source on github: [https://pjreddie.com/darknet/install/](https://link.jianshu.com/?t=https%3A%2F%2Fpjreddie.com%2Fdarknet%2Finstall%2F)
* YOLO is the detector, and Darknet is the structure of YOLO.
* Execute below steps to install Darknet and pre-trained weights(yolov3.weights). please note that yolov3.weights should be under Darknet folder rather than cfg folder.
  + $ git clone <https://github.com/pjreddie/darknet.git>
  + $ cd darknet
  + $ make
  + $ ./darknet
  + $ wget <https://pjreddie.com/media/files/yolov3.weights>
* Once finishing above steps to configure Darknet in system, we can do a test on an image(i.e. dog.jpg) to check results.
  + $ ./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg
* Reference：
  + How to call main program starting from darknet. The location of main function: /darknet/examples/darknet.c. For details, please see the link: <https://blog.csdn.net/qq_41398808/article/details/103514677>

https://blog.csdn.net/Houchaoqun\_XMU/article/details/78710643?utm\_source=blogxgwz3

**3.2 darknet\_test**

* Copy darknet and change name to darknet\_test
* Only detect person
  + Update code in ***cfg/coco.data***

classes= 1

train = /home/pjreddie/data/coco/trainvalno5k.txt

valid = coco\_testdev

#valid = data/coco\_val\_5k.list

names = data/coco.names

backup = /home/pjreddie/backup/

eval=coco

* + Update code in ***examples/detector.c***

void test\_detector(char \*datacfg, char \*cfgfile, char weightfile, char filename, float thresh)

draw\_detections(im, l.wl.hl.n, thresh, boxes, probs, names, alphabet, 1);

void run\_detector(int argc, char \*\*argv)

int classes = option\_find\_int(options, “classes”, 1)

* + Make clean
  + Make
  + Do a test: ./darknet detect cfg/yolo.cfg yolov3.weights data/person.jpg
* Python API
* Yolo has python API to detect image instead of using C++ to run code. We can use python API to add information.
* Python API is located in python folder named as darknet.py
* We need to add Internet configuration first and then use it directly. ***darknet.py*** is based on libdarknet.so and keep them in the same folder before running python code.
* Run python ***darknet.py*** in terminal to do a test
* Count person
* We want to have the number of detected persons as one of result
* See updates in function ***def detect(net, meta, image, thresh=.5, ..)***in darknet.py
* Heatmap on image
* Add new function ***def apply\_heatmap(img, data)*** in darknet.py
* Show detected image and heatmap
* Update function ***def showPicResult(image)*** in darknet.py
* Update main function

**3.3 darknet\_video**

* We use python API for video detection. Through changing code in both some basic C++ files and ***darknet.py***, and then run ***darknet.py*** as normal
* Run video to update code as below:
  + Update code in ***src/image.c***

#ifdef NUMPY

image ndarray\_to\_image(unsigned char\* src, long\* shape, long\* strides)

{

int h = shape[0];

int w = shape[1];

int c = shape[2];

int step\_h = strides[0];

int step\_w = strides[1];

int step\_c = strides[2];

image im = make\_image(w, h, c);

int i, j, k;

int index1, index2 = 0;

for(i = 0; i < h; ++i){

for(k= 0; k < c; ++k){

for(j = 0; j < w; ++j){

index1 = k\*w\*h + i\*w + j;

index2 = step\_h\*i + step\_w\*j + step\_c\*k;

//fprintf(stderr, "w=%d h=%d c=%d step\_w=%d step\_h=%d step\_c=%d \n", w, h, c, step\_w, step\_h, step\_c);

//fprintf(stderr, "im.data[%d]=%u data[%d]=%f \n", index1, src[index2], index2, src[index2]/255.);

im.data[index1] = src[index2]/255.;

}

}

}

rgbgr\_image(im);

return im;

}

#endif

* + Update code in ***src/image.h***

# Position – near 19 line

#ifdef NUMPY

image ndarray\_to\_image(unsigned char\* src, long\* shape, long\* strides);

#endif

* + Update code in ***makefile***

# Position = After 47 line

ifeq ($(NUMPY), 1)

COMMON+= -DNUMPY -I/usr/include/python2.7/ -I/usr/lib/python2.7/dist-packages/numpy/core/include/numpy/

CFLAGS+= -DNUMPY

Endif

# change below info at the beginning of the code

GPU=1

CUDNN=1

OPENCV=1

OPENMP=0

NUMPY=1

DEBUG=0

* + Make clean
  + Make
  + Update code in ***darknet.py***

# find the corresponding position to insert below code

ndarray\_image = lib.ndarray\_to\_image

ndarray\_image.argtypes = [POINTER(c\_ubyte), POINTER(c\_long), POINTER(c\_long)]

ndarray\_image.restype = IMAGE

*# define new function as below*

def nparray\_to\_image(img):

data = img.ctypes.data\_as(POINTER(c\_ubyte))

image = ndarray\_image(data, img.ctypes.shape, img.ctypes.strides)

*return image*

*# update main function*

if \_\_name\_\_ == "\_\_main\_\_":

net = load\_net(b"cfg/yolov3.cfg", b"yolov3.weights", 0)

meta = load\_meta(b"cfg/coco.data")

vid = cv2.VideoCapture(' 1pondo.avi ')

while True:

return\_value,arr=vid.read()

im=nparray\_to\_image(arr)

r = detect(net, meta, im)

* Heatmap on video
* Follow the above steps, we can run video using darknet.py. If we want to draw heatmap on image in real-time, some changes have been added to ***darknet.py*** in function ***draw\_detections(arr, boxes, thinkness=1)*** and ***main fuction***

**3.4 darknet\_image\_gender**

* This detector can detect gender on image. Yolo has 80 categories but it doesn’t include gender category. So we need to train model to get our own weights instead of using yolov3.weights.
* Follow below steps:
* Copy darknet folder
* Update Makefile file:

GPU=1

CUDNN=1

OPENCV=0

OPENMP=0

DEBUG=0

CC=gcc

NVCC=/usr/local/cuda-10.2/bin/nvcc

AR=ar

ARFLAGS=rcs

OPTS=-Ofast

LDFLAGS= -lm -pthread

COMMON= -Iinclude/ -Isrc/

CFLAGS=-Wall -Wno-unused-result -Wno-unknown-pragmas -Wfatal-errors -fPIC

...

ifeq ($(GPU), 1)

COMMON+= -DGPU -I/usr/local/cuda-10.2/include/

CFLAGS+= -DGPU

LDFLAGS+= -L//usr/local/cuda-10.2/lib64 -lcuda -lcudart -lcublas -lcurand

Endif

* Make clean
* Make
* Prepare training dataset
* We use 2 datasets: CeleA and VOCGender that have been put into data folder.
* All training datasets should have below structures:

——Annotations: xml

——ImageSets

———Layout

———Main: test.txt, train.txt, val.txt, traincal.txt

———Segmentation

——JPEGImages: all images

* Update code in ***cfg/train\_voc\_50\_celeb\_50.data***

classes= 2 #2 classes

train = /home/ubuntu/test/darknet\_image\_gender/data/VOCGender/2007\_train.txt

train2 = /home/ubuntu/test/darknet\_image\_gender/data/CelebA/train.txt

names = /home/ubuntu/test/darknet\_image\_gender/data/gender.names

# the trained weights have been stored in backup folder

backup = /home/ubuntu/test/darknet\_image\_gender/backup/gender\_voc\_50\_celeb\_50

* Update code in ***data/gender.names***

Man

Woman

* Update code in ***cfg/yolov3.cfg***

[net]

# Testing ### test model

# batch=1

# subdivisions=1

# Training ### training model

batch=64

subdivisions=16

width=416

height=416

channels=3

momentum=0.9

decay=0.0005

angle=0

saturation = 1.5

exposure = 1.5

hue=.1

learning\_rate=0.001 ### learning rate

burn\_in=1000

max\_batches = 500200 ### iterate

policy=steps

steps=400000,450000 ### learning step

scales=.1,.1

[convolutional]

batch\_normalize=1 ### BN

filters=32

size=3

stride=1

pad=1

activation=leaky

......

[convolutional]

size=1

stride=1

pad=1

filters=21 #3\*(2+4+1) while 2: categories, 4: box coordinate, 1: objectness score, 3: boxes scale

activation=linear

[yolo]

mask = 6,7,8

anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326

classes=2 #category

num=9

jitter=.3

ignore\_thresh = .5

truth\_thresh = 1

random=0

......

[convolutional]

size=1

stride=1

pad=1

filters=21 #3\*(2+4+1)

activation=linear

[yolo]

mask = 3,4,5

anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326

classes=2 #category

num=9

jitter=.3

ignore\_thresh = .5

truth\_thresh = 1

random=0

......

[convolutional]

size=1

stride=1

pad=1

filters=21 #3\*(2+4+1)

activation=linear

[yolo]

mask = 0,1,2

anchors = 10,13, 16,30, 33,23, 30,61, 62,45, 59,119, 116,90, 156,198, 373,326

classes=2 #category

num=9

jitter=.3

ignore\_thresh = .5

truth\_thresh = 1

random=0

# bbox of coco dataset has been classified to 9 classes through k-means clustering, and every 3 classes belong to one scale. Different scale identifies different size of things.

* Download pre-trained model(transformation learning)

wget <https://pjreddie.com/media/files/darknet53.conv.74>

* Train model

./darknet detector train cfg/ train\_voc\_50\_celeb\_50.data cfg/yolov3.cfg darknet53.conv.74

* Stop taining
* Avg loss is below 0.06
* Training weights can be found in backup folder. Below 1000 batches, it stores weights result every 100 batches, but above 1000 batches, every 10000 batches will store one weight result.
* i.e. 929 batch looks like below, and average loss is 92.059631:

929: 810.616150, 92.059631 avg, 0.000745 rate, 0.420621 seconds, 4645 images

**3.5 darknet\_test\_heatmap**

* In order to create a heatmap on layout of store, we have to draw a layout of store first and find 3 points in image which is photographed by camera in store and corresponding 3 points in layout to build a transformation function between image and layout.
* Once getting transformation matrix between image and layout, we can use it to map other points directly. Please note that if the position of camera has been changed, we have to find new transformation matrix accordingly through finding 3 new points on image and corresponding ones in layout.
* How to improve the accuracy of transformation matrix: for selecting 3 points in image, try to find correct corresponding ones in layout.
* For example, we take 3 points in image and find corresponding ones in layout:

In image: (72, 264), (151, 354), (208, 202)

In layout: (221, 553), (297, 260), (34, 586)

# build function as below

from math import cos, sin

def func(x):

c, a, m, n = x[0], x[1], x[2], x[3]

return [

(1 + c)\*(72\*cos(a) + 264\*sin(a)) + m - 221,

(1 + c)\*(-72\*sin(a) + 264\*cos(a)) + n - 553,

(1 + c)\*(151\*cos(a) + 354\*sin(a)) + m - 297,

(1 + c)\*(-151\*sin(a) + 354\*cos(a)) + n - 260,

(1 + c)\*(208\*cos(a) + 202\*sin(a)) + m - 34,

(1 + c)\*(-208\*sin(a) + 202\*cos(a)) + n - 586]

r = fsolve(func, [1, 1, 1, 1])

root1 = leastsq(func, [0, 0, 0, 0])

* Add transformation matrix code in ***darknet.py***

**3.6 darknet\_logo**

* Change and add code in kinds of files referring to steps in section ‘***darknet\_image\_gender***’. The only one different part is to prepare dataset.
* The dataset used in this detector to train model is the json format, which should be changed to the format that yolo needs.
* Dataset structure that yolo uses:
* Dataset folder is in the route: ***darknet\_logo*** -> ***data*** -> ***LOGO***
* Structure of dataset as below:

— LOGO2020

——Annotations: xml (it is not necessary for yolo)

——Labels

———test2019: test.txt

———train2019: train.txt

———val2019: val.txt

——JPEGImages: all images

— train.txt: it is used in ***logo.data***

— text.txt: it is used in ***logo.data***

* Well trained: average loss is below 0.06. if the avg loss is not able to decrease down to 0.06, there are some parameters that we can tune to resolve it such as learning rate (in logo.cfg file), improving noise of pictures (in logo.cfg file), without using transformation learning (darknet53.conv.74), etc.
* Dataset downloaded from the link: https://rpc-dataset.github.io/#3-our-rpc-dataset

**4. Optimization**

**4.1 Heatmap on layout(darknet\_test\_heatmap):**

* In this detector, we use transformation matrix from 2-D to 2-D. The quality of the result is based on the points we choose on 2 images (image from camera and layout).
* Try to consider transforming original image from 2-D to 3-D and then use 3-D to create layout directly.

**4.1 Gender detector(darknet\_image\_gender):**

* We have got good test result on image that photographed from the front and back. But in the supermarket, the camera is installed on the top so the picture is shot from top. For these pictures, the detector doesn’t have good performance. How to optimize it? One popular way is to take similar pictures in supermarket, totally 200 to 300 pictures, and then use them to train model based on existing trained weights. Label pictures using the below tool named Labelimg

https://blog.csdn.net/xiaomu\_347/article/details/83744828

**4.2 Object detector(darknet\_logo):**

* Try to get lower avg loss: feature engineering on dataset and tune parameters
* Try to use this detector on shelf detection. If the result is not good, another dataset should be considered or researching other algorithms.

**4.3 others**

* YOLO can also be used to do classification, NLP, etc.
* YOLO is faster but not good for small objects detection.