**YOLO:**

Descriptions:

When it comes to deep learning-based object detection, there are three primary object detectors you’ll encounter:

1> R-CNN and their variants, including the original R-CNN, Fast R- CNN, and Faster R-CNN

2> Single Shot Detector (SSDs)

3> YOLO

R-CNNs are one of the first deep learning-based object detectors and are an example of a two-stage detector.

While R-CNNs tend to be very accurate, the biggest problem with the R-CNN family of networks is their speed — they were incredibly slow, obtaining only 5 FPS on a GPU.

To help increase the speed of deep learning-based object detectors, both Single Shot Detectors (SSDs) and YOLO use a one-stage detector strategy.

These algorithms treat object detection as a regression problem, taking a given input image and simultaneously learning bounding box coordinates and corresponding class label probabilities.

In general, single-stage detectors tend to be less accurate than two-stage detectors but are significantly faster.

YOLO is a great example of a single stage detector.

The YOLOv3 object detector pre-trained (on the COCO dataset) model files. These were trained by the Darknet team.

1> faster Rcnn:

As we know, the Faster R-CNN/Mask R-CNN architectures leverage a Region Proposal Network (RPN) to generate regions of an image that potentially contain an object.

Each of these regions is ranked based on their “objectness score” (i.e., how likely it is that a given region could potentially contain an object)

and then the top N most confident objectness regions are kept.

frozen\_inference\_graph.pb : The Mask R-CNN model weights. The weights are pre-trained on the COCO dataset.

mask\_rcnn\_inception\_v2\_coco\_2018\_01\_28.pbtxt : The Mask R-CNN model configuration. If you’d like to build + train your own model on your own annotated data,

object\_detection\_classes\_coco.txt : All 90 classes are listed in this text file, one per line. Open it in a text editor to see what objects our model can recognize.

2> yolo:

YOLO use a one-stage detector strategy.

These algorithms treat object detection as a regression problem, taking a given input image and simultaneously learning bounding box coordinates and corresponding class label probabilities.

In general, single-stage detectors tend to be less accurate than two-stage detectors but are significantly faster.

YOLO is a great example of a single stage detector.

The YOLOv3 object detector pre-trained (on the COCO dataset) model files. These were trained by the Darknet team.

To load YOLO from disk take advantage of OpenCV’s DNN function called cv2.dnn.readNetFromDarknet . This function requires both a configPath and weightsPath

**YoloV5:**

* **step -1 : Downloading the Vehicles-OpenImages Dataset :**

"""

data.yaml: It has the data-related configurations like the train and valid data directory path, the total number of classes in the dataset, and the name of each class

train: Training images along with training labels

valid: Validation images with annotations

test: Test images and labels. Assessing your model’s performance becomes easy if test data with labels are available.

"""

# Download the vehicles-open image dataset

!mkdir vehicles\_open\_image

%cd vehicles\_open\_image

!curl -L "https://public.roboflow.com/ds/2Tb6yXY8l8?key=Eg82WpxUEr" > vehicles.zip

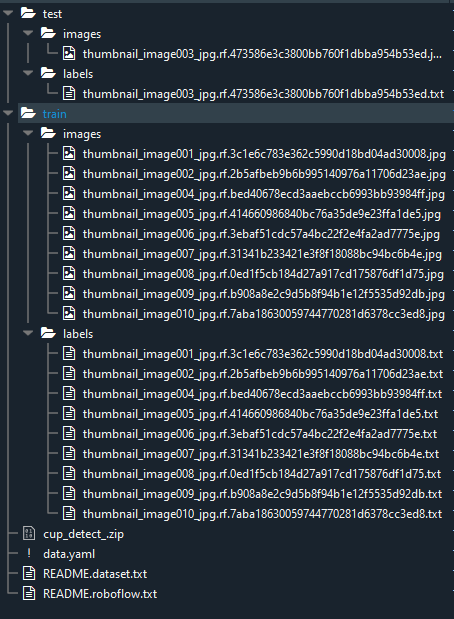
!unzip vehicles.zip

!rm vehicles.zip

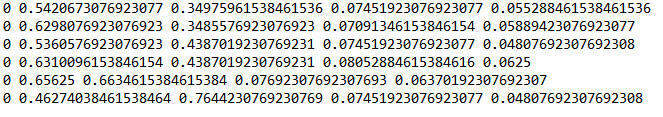
**Note: dataset should be in yolov5 pytorch format**

**Annote the custom dataset on yoloV5 pytorch fomat as shown below:**

cup\_detect\_.zip



**In labels folder .txt file look like below:**



* **step-2 Configuration Setup :**

#Next, we will edit the data.yaml file to have the path and absolute path for train and valid images.

# Create configuration

import yaml

config = {'path': '/content/vehicles\_open\_image',

'train': '/content/vehicles\_open\_image/train',

'val': '/content/vehicles\_open\_image/valid',

'nc': 5,

'names': ['Ambulance', 'Bus', 'Car', 'Motorcycle', 'Truck']}

with open("data.yaml", "w") as file:

yaml.dump(config, file, default\_flow\_style=False)

* **step - 4: clone repo :**

%cd ..

!git clone https://github.com/ultralytics/yolov5.git #clone repo

%cd yolov5/

!pip install -r requirements.txt #install dependencies

* **step 3: YOLOv5 Training Hyperparameters and Model Configuration :**

lr0: 0.01 # initial learning rate (SGD=1E-2, Adam=1E-3)

lrf: 0.01 # final OneCycleLR learning rate (lr0 \* lrf)

momentum: 0.937 # SGD momentum/Adam beta1

weight\_decay: 0.0005 # optimizer weight decay 5e-4

warmup\_epochs: 3.0 # warmup epochs (fractions ok)

warmup\_momentum: 0.8 # warmup initial momentum

warmup\_bias\_lr: 0.1 # warmup initial bias lr

box: 0.05 # box loss gain

cls: 0.5 # cls loss gain

cls\_pw: 1.0 # cls BCELoss positive\_weight

obj: 1.0 # obj loss gain (scale with pixels)

obj\_pw: 1.0 # obj BCELoss positive\_weight

iou\_t: 0.20 # IoU training threshold

anchor\_t: 4.0 # anchor-multiple threshold

# anchors: 3 # anchors per output layer (0 to ignore)

fl\_gamma: 0.0 # focal loss gamma (efficientDet default gamma=1.5)

hsv\_h: 0.015 # image HSV-Hue augmentation (fraction)

hsv\_s: 0.7 # image HSV-Saturation augmentation (fraction)

hsv\_v: 0.4 # image HSV-Value augmentation (fraction)

degrees: 0.0 # image rotation (+/- deg)

translate: 0.1 # image translation (+/- fraction)

scale: 0.5 # image scale (+/- gain)

shear: 0.0 # image shear (+/- deg)

perspective: 0.0 # image perspective (+/- fraction), range 0-0.001

flipud: 0.0 # image flip up-down (probability)

fliplr: 0.5 # image flip left-right (probability)

mosaic: 1.0 # image mosaic (probability)

mixup: 0.0 # image mixup (probability)

copy\_paste: 0.0 # segment copy-paste (probability)

* **step -4 : train the yolov5 model**

SIZE = 640

BATCH\_SIZE = 32

EPOCHS = 20

MODEL = "yolov5s"

WORKERS = 1

PROJECT = "vehicles\_open\_image\_pyimagesearch"

RUN\_NAME = f"{MODEL}\_size{SIZE}\_epochs{EPOCHS}\_batch{BATCH\_SIZE}\_small"

!python train.py --img {SIZE}\

--batch {BATCH\_SIZE}\

--epochs {EPOCHS}\

--data ../vehicles\_open\_image/data.yaml\

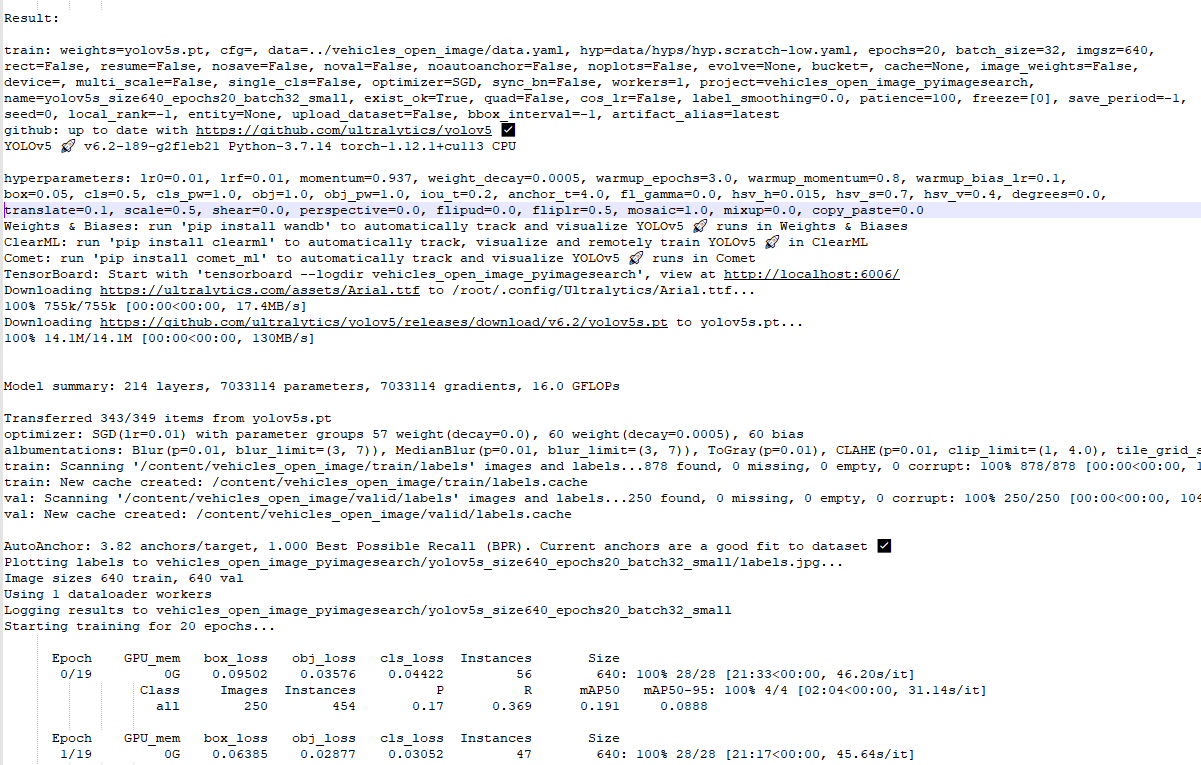
--weights {MODEL}.pt\

--workers {WORKERS}\

--project {PROJECT}\

--name {RUN\_NAME}\

--exist-ok



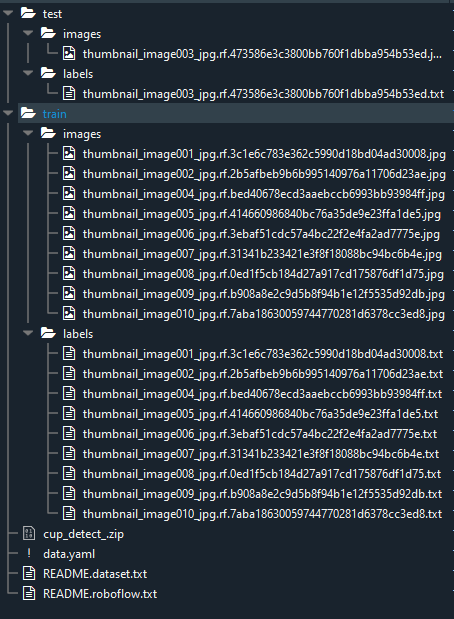
**YoloV7:**

**Step -1 : Annote the custom dataset on yoloV7 pytorch fomat as shown below:**

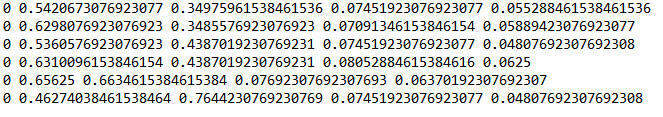
**Cmd: !mkdir cup\_data**

1. **By manually using labelimg tools:**

cup\_detect\_.zip



**In labels folder .txt file look like below:**



1. **Optional :**

**Annotation optional using roboflow** :

Annote the dataset using roboflow and download it

"""

# REPLACE with your custom code snippet generated above

!pip install roboflow

from roboflow import Roboflow

rf = Roboflow(api\_key="YOUR API KEY")

project = rf.workspace("YOUR-WORKSPACE").project("YOUR-PROJECT")

dataset = project.version(1).download("yolov7")

"""

1. **Download annotted data from below link :**

<https://drive.google.com/file/d/1hAm7_g9CL4nbwI29zcyl9KjMU92Ta_Gf/view?usp=sharing>

Step -2 : # Download YOLOv7 repository and install requirements

!git clone https://github.com/WongKinYiu/yolov7

%cd yolov7

!pip install -r requirements.txt

Step -3 #upload custom annotted data and unzip it

!unzip /content/cup\_detect\_.zip ./content/cup\_data

Step-4 Configuration Setup:

#Next, we will edit the data.yaml file to have the path and absolute path for train and valid images.

# Create configuration

import yaml

config = {'path': '/content/cup\_data',

         'train': '/content/cup\_data/train',

         'val': '/content/cup\_data/test',

         'nc': 1,

         'names': ['cup']}

with open("/content/cup\_data/data.yaml", "w") as file:

   yaml.dump(config, file, default\_flow\_style=False)

Step -5 # download COCO starting checkpoint

%cd /content/yolov7

!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7\_training.pt

**Step 6 : Train the model :**

# run this cell to begin training

# --device 1 ----- to with cuda gpu

%cd /content/yolov7

!python train.py --batch 16 --epochs 55 --data /content/cup\_data/data.yaml --weights 'yolov7\_training.pt'

**Result:**

/content/yolov7

YOLOR 🚀 v0.1-115-g072f76c torch 1.12.1+cu113 CPU

Namespace(adam=False, artifact\_alias='latest', batch\_size=16, bbox\_interval=-1, bucket='', cache\_images=False, cfg='',

data='/content/cup\_data/data.yaml', device='', entity=None, epochs=55, evolve=False, exist\_ok=False, freeze=[0], global\_rank=-1,

hyp='data/hyp.scratch.p5.yaml', image\_weights=False, img\_size=[640, 640], label\_smoothing=0.0, linear\_lr=False, local\_rank=-1,

multi\_scale=False, name='exp', noautoanchor=False, nosave=False, notest=False, project='runs/train', quad=False, rect=False,

resume=False, save\_dir='runs/train/exp', save\_period=-1, single\_cls=False, sync\_bn=False, total\_batch\_size=16, upload\_dataset=False,

v5\_metric=False, weights='yolov7\_training.pt', workers=8, world\_size=1)

tensorboard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/

hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937, weight\_decay=0.0005, warmup\_epochs=3.0, warmup\_momentum=0.8, warmup\_bias\_lr=0.1,

box=0.05, cls=0.3, cls\_pw=1.0, obj=0.7, obj\_pw=1.0, iou\_t=0.2, anchor\_t=4.0, fl\_gamma=0.0, hsv\_h=0.015, hsv\_s=0.7, hsv\_v=0.4, degrees=0.0,

translate=0.2, scale=0.9, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.15, copy\_paste=0.0, paste\_in=0.15, loss\_ota=1

wandb: Install Weights & Biases for YOLOR logging with 'pip install wandb' (recommended)

Overriding model.yaml nc=80 with nc=1

105 [102, 103, 104] 1 34156 models.yolo.IDetect [1, [[12, 16, 19, 36, 40, 28], [36, 75, 76, 55, 72, 146], [142, 110, 192, 243, 459, 401]], [256, 512, 1024]]

/usr/local/lib/python3.7/dist-packages/torch/functional.py:478: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at ../aten/src/ATen/native/TensorShape.cpp:2894.)

return \_VF.meshgrid(tensors, \*\*kwargs) # type: ignore[attr-defined]

Model Summary: 415 layers, 37196556 parameters, 37196556 gradients, 105.1 GFLOPS

Transferred 557/566 items from yolov7\_training.pt

Scaled weight\_decay = 0.0005

Optimizer groups: 95 .bias, 95 conv.weight, 98 other

train: Scanning '/content/cup\_data/train/labels' images and labels... 9 found, 0 missing, 0 empty, 0 corrupted: 100% 9/9 [00:00<00:00, 376.11it/s]

train: New cache created: /content/cup\_data/train/labels.cache

val: Scanning '/content/cup\_data/test/labels' images and labels... 1 found, 0 missing, 0 empty, 0 corrupted: 100% 1/1 [00:00<00:00, 265.65it/s]

val: New cache created: /content/cup\_data/test/labels.cache

autoanchor: Analyzing anchors... anchors/target = 6.00, Best Possible Recall (BPR) = 1.0000

Image sizes 640 train, 640 test

Using 2 dataloader workers

Logging results to runs/train/exp

Starting training for 55 epochs...

**Epoch gpu\_mem box obj cls total labels img\_size**

**0% 0/1 [00:00<?, ?it/s]/usr/local/lib/python3.7/dist-packages/torch/amp/autocast\_mode.py:198: UserWarning: User provided device\_type of 'cuda', but CUDA is not available. Disabling**

**warnings.warn('User provided device\_type of \'cuda\', but CUDA is not available. Disabling')**

**YoloV4:**

* **Step 1: download custom dataset annotted :**

**Using below kaggle API :**

!pip install opendatasets

import opendatasets as od

import pandas

od.download("https://www.kaggle.com/datasets/techzizou/labeled-mask-dataset-yolo-darknet")

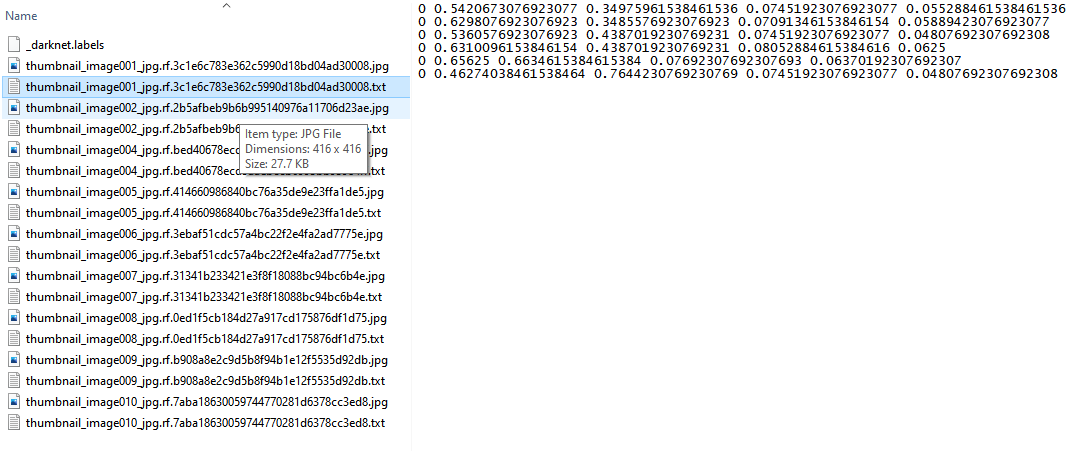
**Or**

**Direct Download using below link :**

<https://www.kaggle.com/datasets/techzizou/labeled-mask-dataset-yolo-darknet>

Note : Custom dataset should be annnotted on **yoloV4 darknet format**.

Train folder look like below:



* **Step -2 : Create & upload the following files which we need for training a custom detector**

a. Labeled Custom Dataset

b. Custom cfg file

c. obj.data and obj.names files

d. process.py file (to create train.txt and test.txt files for training)

Note: working with 2 classes i.e. with\_mask and without\_mask

All above file will get from below repo:

Cmd:

!git clone https://github.com/techzizou/yolov4-custom\_Training.git

!cp -r /content/yolov4-custom\_Training/yolov4 ./

%cd /content/yolov4

Create your obj.data and obj.names files and upload to your drive

**obj.data**

classes = 2

train = data/train.txt

valid = data/test.txt

names = data/obj.names

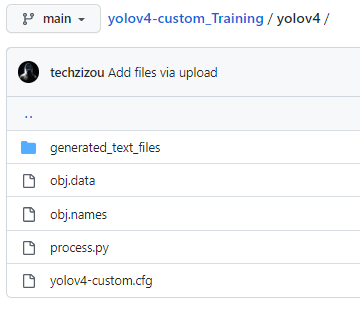
backup = /mydrive/yolov4/training

**obj.names**

with\_mask

without\_mask

Yolov4 folder look like below”



* **Step- 3 : Clone darknet git repository :**

!git clone https://github.com/AlexeyAB/darknet

# change makefile to have GPU and OPENCV enabled

# also set CUDNN, CUDNN\_HALF and LIBSO to 1

%cd darknet/

!sed -i 's/OPENCV=0/OPENCV=1/' Makefile

!sed -i 's/GPU=0/GPU=1/' Makefile

!sed -i 's/CUDNN=0/CUDNN=1/' Makefile

!sed -i 's/CUDNN\_HALF=0/CUDNN\_HALF=1/' Makefile

!sed -i 's/LIBSO=0/LIBSO=1/' Makefile

# build darknet

!make

Step 4: **Copy all the files from the yolov4 folder to the darknet directory :**

# Clean the data and cfg folders first except the labels folder in data which is required

%cd data/

!find -maxdepth 1 -type f -exec rm -rf {} \;

%cd ..

%rm -rf cfg/

%mkdir cfg

# Unzip the obj.zip dataset and its contents so that they are now in /darknet/data/ folder

#!unzip /mydrive/yolov4/obj.zip -d data/

!cp -r /content/labeled-mask-dataset-yolo-darknet/obj data

# Copy the yolov4-custom.cfg file so that it is now in /darknet/cfg/ folder

!cp /content/yolov4/yolov4-custom.cfg cfg

# verify if your custom file is in cfg folder

!ls cfg/

# Copy the obj.names and obj.data files so that they are now in /darknet/data/ folder

!cp /content/yolov4/obj.names data

!cp /content/yolov4/obj.data  data

# Copy the process.py file to the current darknet directory

!cp /content/yolov4/process.py .

**Step 5: Run the process.py python script to create the train.txt & test.txt files inside the data folder :**

# run process.py ( this creates the train.txt and test.txt files in our darknet/data folder )

!python process.py

# list the contents of data folder to check if the train.txt and test.txt files have been created

!ls data/

**Step 6: Download the pre-trained yolov4 weights:**

# Download the yolov4 pre-trained weights file

!wget https://github.com/AlexeyAB/darknet/releases/download/darknet\_yolo\_v3\_optimal/yolov4.conv.137

**Step 7: Train your custom detector:**

**For best results, you should stop the training when the average loss is less than 0.05 if possible or at least below 0.3, else train the model until the average loss does not show any significant change for a while.**

# train your custom detector! (uncomment %%capture below if you run into memory issues or your Colab is crashing)

# %%capture

!./darknet detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont\_show -map

**Result:**

**CUDA-version: 11020 (11020), cuDNN: 8.1.1, CUDNN\_HALF=1, GPU count: 1**

**CUDNN\_HALF=1**

**OpenCV version: 3.2.0**

**Prepare additional network for mAP calculation...**

**0 : compute\_capability = 750, cudnn\_half = 1, GPU: Tesla T4**

**net.optimized\_memory = 0**

**mini\_batch = 1, batch = 16, time\_steps = 1, train = 0**

**layer filters size/strd(dil) input output**

**0 Create CUDA-stream - 0**

**Create cudnn-handle 0**

**161 yolo**

**[yolo] params: iou loss: ciou (4), iou\_norm: 0.07, obj\_norm: 1.00, cls\_norm: 1.00, delta\_norm: 1.00, scale\_x\_y: 1.05**

**nms\_kind: greedynms (1), beta = 0.600000**

**Total BFLOPS 59.570**

**avg\_outputs = 489910**

**Allocate additional workspace\_size = 52.44 MB**

**yolov4-custom**

**0 : compute\_capability = 750, cudnn\_half = 1, GPU: Tesla T4**

**net.optimized\_memory = 0**

**mini\_batch = 4, batch = 64, time\_steps = 1, train = 1**

**161 yolo**

**[yolo] params: iou loss: ciou (4), iou\_norm: 0.07, obj\_norm: 1.00, cls\_norm: 1.00, delta\_norm: 1.00, scale\_x\_y: 1.05**

**nms\_kind: greedynms (1), beta = 0.600000**

**Total BFLOPS 59.570**

**avg\_outputs = 489910**

**Allocate additional workspace\_size = 149.82 MB**

**Loading weights from yolov4.conv.137...**

**seen 64, trained: 0 K-images (0 Kilo-batches\_64)**

**Done! Loaded 137 layers from weights-file**

**Learning Rate: 0.001, Momentum: 0.949, Decay: 0.0005**

**Detection layer: 139 - type = 28**

**Detection layer: 150 - type = 28**

**Detection layer: 161 - type = 28**

**Resizing, random\_coef = 1.40**

**608 x 608**

**Create 6 permanent cpu-threads**

**try to allocate additional workspace\_size = 189.86 MB**

**CUDA allocate done!**

**Loaded: 0.000040 seconds**

**v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 139 Avg (IOU: 0.000000), count: 1, class\_loss = 5586.581543, iou\_loss = 0.000000, total\_loss = 5586.581543**

**v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.182250), count: 3, class\_loss = 1048.471436, iou\_loss = 0.048096, total\_loss = 1048.519531**

**v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.501089), count: 12, class\_loss = 267.065186, iou\_loss = 0.343079, total\_loss = 267.408264**

**total\_bbox = 15, rewritten\_bbox = 0.000000 %**

**YoloV3:**

from google.colab import drive

drive.mount('/content/gdrive')

!ln -s /content/gdrive/My\ Drive/ /mydrive

!ls /mydrive

**Step-1 :**

1. **Clone the daknet**

!git clone https://github.com/AlexeyAB/darknet

**Compile Darknet using Nvidia GPU:**

# change makefile to have GPU and OPENCV enabled

%cd darknet

!sed -i 's/OPENCV=0/OPENCV=1/' Makefile

!sed -i 's/GPU=0/GPU=1/' Makefile

!sed -i 's/CUDNN=0/CUDNN=1/' Makefile

!make

**Step -2:**

**Configure Darknet network for training YOLO V3**

!cp cfg/yolov3.cfg cfg/yolov3\_training.cfg

!sed -i 's/batch=1/batch=64/' cfg/yolov3\_training.cfg

!sed -i 's/subdivisions=1/subdivisions=16/' cfg/yolov3\_training.cfg

!sed -i 's/max\_batches = 500200/max\_batches = 4000/' cfg/yolov3\_training.cfg

!sed -i '610 s@classes=80@classes=1@' cfg/yolov3\_training.cfg

!sed -i '696 s@classes=80@classes=1@' cfg/yolov3\_training.cfg

!sed -i '783 s@classes=80@classes=1@' cfg/yolov3\_training.cfg

!sed -i '603 s@filters=255@filters=18@' cfg/yolov3\_training.cfg

!sed -i '689 s@filters=255@filters=18@' cfg/yolov3\_training.cfg

!sed -i '776 s@filters=255@filters=18@' cfg/yolov3\_training.cfg

!mkdir "/content/yolov3"

!echo "pot" > data/obj.names

!echo -e 'classes= 1\ntrain  = data/train.txt\nvalid  = data/test.txt\nnames = data/obj.names\nbackup = /content/yolov3' > data/obj.data

!mkdir data/obj

!cp cfg/yolov3\_training.cfg /content/yolov3/yolov3\_testing.cfg

!cp data/obj.names /content/yolov3/classes.txt

# Download weights darknet model 53

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

**Step -3: Download the dataset:**

**Download dataset from below link:**

**Images.zip**

**<https://drive.google.com/file/d/1sWcniczmtLWWWNOsgWm0bW8tTrn1ZxQW/view?usp=sharing>**

**Extract Images**

**The images need to be inside a zip archive called "images.zip" and they need to be inside the folder "yolov3"**

!unzip /content/images.zip -d data/obj

**Step -4 : start the training:**

import glob

images\_list = glob.glob("data/obj/\*.jpg")

print(images\_list)

with open("data/train.txt", "w") as f:

    f.write("\n".join(images\_list))

# Start the training

!./darknet detector train data/obj.data cfg/yolov3\_training.cfg darknet53.conv.74 -dont\_show

**Result:**

**Streaming output truncated to the last 5000 lines.** v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.597882), count: 16, class\_loss = 2.245492, iou\_loss = 2.679028, total\_loss = 4.924519 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.615584), count: 5, class\_loss = 1.259185, iou\_loss = 0.848613, total\_loss = 2.107799 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class\_loss = 0.000663, iou\_loss = 0.000000, total\_loss = 0.000663 total\_bbox = 147335, rewritten\_bbox = 0.000000 % v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.591621), count: 9, class\_loss = 1.370835, iou\_loss = 1.420687, total\_loss = 2.791522 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.587303), count: 10, class\_loss = 1.668527, iou\_loss = 1.777594, total\_loss = 3.446121 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.401879), count: 2, class\_loss = 0.552142, iou\_loss = 1.091768, total\_loss = 1.643910 total\_bbox = 147356, rewritten\_bbox = 0.000000 % v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.554410), count: 16, class\_loss = 1.372218, iou\_loss = 3.534926, total\_loss = 4.907145 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.583716), count: 3, class\_loss = 0.744673, iou\_loss = 0.597081, total\_loss = 1.341754 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class\_loss = 0.000660, iou\_loss = 0.000000, total\_loss = 0.000660 total\_bbox = 147375, rewritten\_bbox = 0.000000 % v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.524180), count: 12, class\_loss = 1.405161, iou\_loss = 3.051949, total\_loss = 4.457109 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.651696), count: 8, class\_loss = 0.717272, iou\_loss = 1.182696, total\_loss = 1.899968 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.000000), count: 1, class\_loss = 0.000695, iou\_loss = 0.000000, total\_loss = 0.000695 total\_bbox = 147395, rewritten\_bbox = 0.000000 % v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.693964), count: 10, class\_loss = 1.011873, iou\_loss = 0.856601, total\_loss = 1.868474 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.538796), count: 8, class\_loss = 1.944561, iou\_loss = 2.039993, total\_loss = 3.984554 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.438767), count: 2, class\_loss = 0.585304, iou\_loss = 0.613031, total\_loss = 1.198335 total\_bbox = 147415, rewritten\_bbox = 0.000000 % v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.702691), count: 8, class\_loss = 1.315074, iou\_loss = 0.724434, total\_loss = 2.039508 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.488883), count: 9, class\_loss = 1.940190, iou\_loss = 3.179153, total\_loss = 5.119343 v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.353075), count: 1, class\_loss = 0.267574, iou\_loss = 0.286506, total\_loss = 0.554080 total\_bbox = 147433, rewritten\_bbox = 0.000000 % 454: 2.309885, 2.641740 avg loss, 0.000042 rate, 5.609965 seconds, 29056 images, 5.856388 hours left Loaded: 0.000089 seconds