#==================➔

Code link (Github):
https://github.com/luv91/ObjectDetectionYolov3Yolov4

Datalink(google drive):

https://drive.google.com/drive/folders/1qH9LJvPpf3i8aIcSLOPHAS9BtKGBsWwP?usp=sharing

Question:

ML/DL Assignment
EagleView

Object Detection is one of the fundamental tasks in computer vision with applications ranging across medicine, robotics, and many others. In this task, you are given a dataset that consists of images containing people and cars. The goal of this task is to train a model that can localize and classify each instance of person and car as accurately as possible.

Dataset Details:

Number of images:	2239
Size:	750 MB (approx.)
Categories:	2 (person and car)
Annotation format:	COCO

Data directory structure:

```
data
  |_ annotations
        |_ bbox-annotations.json
  |_ images
        |_ image_000000001.jpg
        |_ image_000000002.jpg
        |_ ...
        ...
```

License: These images are taken from OpenImages. All annotations are licensed under CC BY 4.0 and the images have a CC BY 2.0 license*

Link to download the dataset: https://evp-ml-data.s3.us-east-2.amazonaws.com/ml-interview/openimages-personcar/trainval.tar.gz

Answer:

Yolov3 Explanation:

1. Yolo v3 can:
    a. Detect Multiple objects
    b. Predict classes
    c. Localize the objects present

2. Abstractly speaking, Yolov3:
    a. Applies single NN
    b. Divide images into grid cells
    c. Produces cell probabilities
    d. Predict boxes.

3. Yolov3 Architecture:
    a. 53 CNN layers called as Darknet-53
    b. 53 more layers, thus 106 layers overall
    c. Detections happen at layers: 82, 94 and 106.

4. Essential elements of Yolo v3:
    a. Residual blocks
    b. Skip connections
    c. Up-sampling
    d. CNN layers are followed by Leaky Relu and Batch Normalization
    e. There are no pooling layers and instead of pooling layers there are convolutional layers instead. These prevents loss of low level features, thus Yolov3 has ability to detect small objects
5. Input to the network:
    a. Batch of images ➔ (n, 416, 416, 3) namely images, width, height and RGB channel respectively.
    b. Width and height can be any other number also which is divisible by 32.
    c. Input to the network does not have to be resized.

6. Detections in Yolov3:
    a. Detections take place at layers no: 82, 94 and 106.
    b. The input image (say 416 by 416) is downsamples by stride of 32, 16 and 8 at layer 82, 94 and 106 respectively.
    c. The above is done to detect the objects of different sizes. For example, higher strides are responsible for detection of large objects whereas smaller strides are responsible for detecting smaller objects.
    d. Yolo v3 predicts 3 bounding boxes for every cell

7. Bounding Box attributes:
    a. 3 bounding boxes for each cell.
    b. Each BB has (5+c) attributes.
    c. 5 are (center(x,y); width; height; probability confidence score). C are the list of confidences whose size depends on number of classes to predict. (for example in the question above there are 2 classes).
    d. Giving example, at detection layer, say layer number 82(stride 32, input image 416x416) , the feature map would have dimension of: 13x13x(7+7+7) ➔ 13x13x21; for 2 classes.
    e. In total yolov3 can have 10,647 bounding boxes (13x13x3+26x26x3+52x52x3) ; where 3 is coming from 3 bounding boxes for each grid cell.
8. Grid Cells:
    a. For each grid cell, 3 bounding boxes
    b. However, only 1 cell is responsible (is the ground truth cell) for one object.
    c. The ground truth cell is basically the center cell of the object.
    d. The objectness score is 1 for this cell.

9. Prediction of Bounding Boxes
    a. Find out offset of predicted Bounding boxes to the ground truth boxes and not the absolute values.
    b. After training and forward pass, network output four coordinates (2 for center and 2 for height and width).

c. To find out center of the predicted bounding box, 2 outputs are passed through sigmoid function and cell's top left corner is added.
d. To get the absolute values, multiply by width and height of the images.
e. To find out width and height of a predicted BB, two outputs are passed through log space transform and multiplied by anhor's width and height.

10. Objectness Score/confidence:
    a. Objectness score/confidence is given by multiplication of (P(object)➔predicted probability that BB contains object)x(IoU).
    b. Objectness score is between 0 and 1.
    c. If P(object) equal to 0, means that no BB is associated with that cell
    d. As the objectness score of the cells around the center cell is not 0 and near to 1 (linke nearby cells can have value of 0.9 for a particular object).. generally there can be more than 1 predicted bounding box associated with a particular object.
    e. Therefore Non maximal suppression is applied, where the bounding box with highest confidence score is selected. And other bounding boxes which are finding out same object are removed by calculating the IoU with the bounding box with highest confidence. It is called as NMS, because suppression of bounding boxes that do not have the maximum confidence are removed.

#==========➔

Dataset Preparation:

1. The annotation file was in .json (coco) format and that was converted into .txts(yolo format).
2. Every .jpg (image) file had associated .txt file.
3. Each .txt file had the information which was in the format (<object-class>, <x_center>,<y_center>, <width> and <height>).

4. <x_center> ,<y_center> are the co-ordinates of the center of bounding box.
5. <width> and <height> are float values relative to the dimensions of the image.
6. Train and test split was created. I chose to keep train/test split of 80-20 percent. Two text files were generated, train.txt and test.txt. These text files had path to each image in train and test set.
7. Two more files called as obj.names and obj.data were created. Obj.names had names of the classes. Obj.data had information about the training, validation, names and backup folders.

#==========➔

Training of Yolov3 and Yolov4:

Training was done using respective darknet models for Yolo v3 and yolo v4.

A. In google colab, options to use GPU, cuDNN, OPENCV all were set to 1.
B. Changes in the configuration files yolov3.cfg & yolov4.cfg are as follows:
    1. Width and height for yolo ➔ 416 x 416
    2. Batch ➔ 64
    3. Subdivisions ➔ 16
    4. Maximum no. of batches considered ➔ 6000
        i. Minimum number of batches can be (Minimum No. of classes x 2000)
        ii. 6000 was considered as it is above 4000(minimum), but due to GPU limitations only 6000 were considered.
    5. Steps ➔ 80% and 90% of max_batches.
    6. Filters ➔ (number of classes+5)*3
        i. Number of classes = 2. No. of filters are 21 for this problem.
    7. Classes: No of classes ➔ 2.

C. To train, the following was instantiated:

1. ./darknet detector train data/obj.data cfg/yolo-obg.cfg yolo-weights. (this was modified according to the used names of cfg files and weights).

#==========================➔

Testing on an image:

Once weights are available, testing on an image can be either done through opencv or through darknet.

To test an image through darknet I have run the following command.

```
!./darknet detector test cfg/obj.data cfg/yolov3.cfg yolov3.weights ../ima
ge_000002232.jpg
```

#==========================➔

Evaluation Metric:

Name	Class_id	Average Precision	Mean Average precision
Yolo v3			
Person	0	47.58 %	**55.52%**
Car	1	63.45 %	
Yolo v4			
Person	0	63.54%	**67.50%**
Car	1	71.45 %	

#==========================➔

Proof of running yolov3 and Yolov4 are attached below in terms of running the code:

1. Results from Yolo v3

```
 (next mAP calculation at 6040 iterations)
 Last accuracy mAP@0.50 = 55.64 %, best = 55.79 %
 6000: 1.177478, 1.682504 avg loss, 0.000010 rate, 9.662286 seconds, 384000 images, 0.189682 hours left
Resizing to initial size: 416 x 416  try to allocate additional workspace_size = 52.43 MB
 CUDA allocate done!

 calculation mAP (mean average precision)...
 Detection layer: 82 - type = 28
 Detection layer: 94 - type = 28
 Detection layer: 106 - type = 28
440
 detections_count = 9935, unique_truth_count = 3035
class_id = 0, name = person, ap = 47.58%         (TP = 846, FP = 428)
class_id = 1, name = car, ap = 63.45%            (TP = 626, FP = 178)

 for conf_thresh = 0.25, precision = 0.71, recall = 0.49, F1-score = 0.58
 for conf_thresh = 0.25, TP = 1472, FP = 606, FN = 1563, average IoU = 52.60 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.555164, or 55.52 %
Total Detection Time: 13 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset

 mean_average_precision (mAP@0.50) = 0.555164
Saving weights to /mydrive/yolov3/backup//yolov3_custom2_6000.weights
Saving weights to /mydrive/yolov3/backup//yolov3_custom2_last.weights
Saving weights to /mydrive/yolov3/backup//yolov3_custom2_final.weights
If you want to train from the beginning, then use flag in the end of training command: -clear
```

2. Results from Yolo v4.

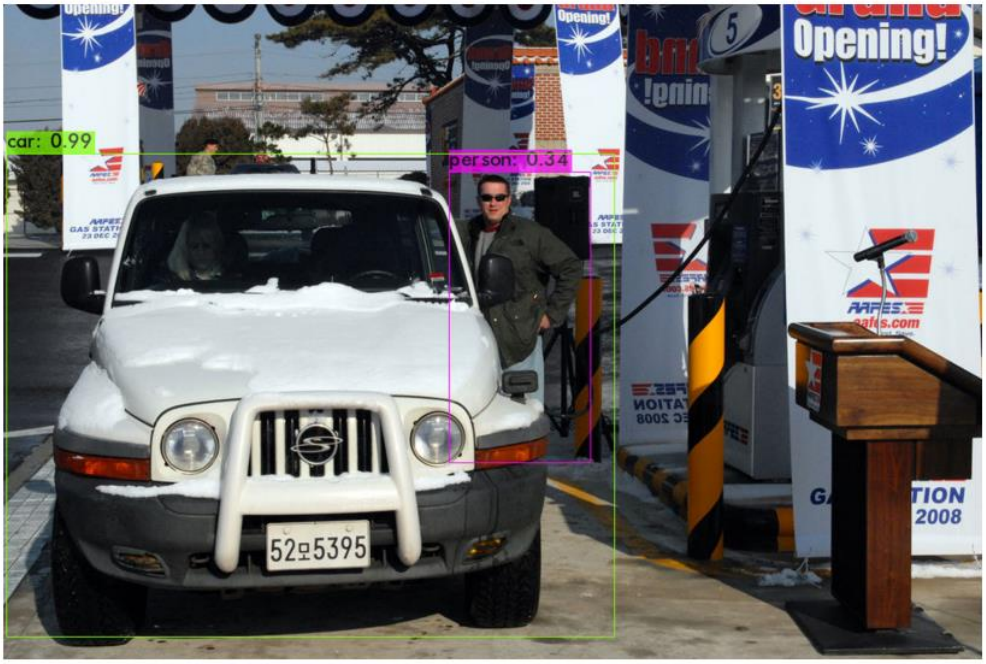
```
[yolo] params: iou loss: ciou (4), iou_norm: 0.07, obj_norm: 1.00, cls_norm: 1.00, delta_norm: 1.00, scale_x_y: 1.05
nms_kind: greedynms (1), beta = 0.600000
Total BFLOPS 59.570
avg_outputs = 489910
 Allocate additional workspace_size = 52.43 MB
Loading weights from /mydrive/yolov4/backup/yolov4-obj_last.weights...
 seen 64, trained: 384 K-images (6 Kilo-batches_64)
Done! Loaded 162 layers from weights-file

 calculation mAP (mean average precision)...
 Detection layer: 139 - type = 28
 Detection layer: 150 - type = 28
 Detection layer: 161 - type = 28
440
 detections_count = 10787, unique_truth_count = 3035
class_id = 0, name = person, ap = 63.54%         (TP = 1229, FP = 567)
class_id = 1, name = car, ap = 71.45%            (TP = 756, FP = 238)

 for conf_thresh = 0.25, precision = 0.71, recall = 0.65, F1-score = 0.68
 for conf_thresh = 0.25, TP = 1985, FP = 805, FN = 1050, average IoU = 54.95 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.674971, or 67.50 %
Total Detection Time: 10 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

#-==========➔

Comparison on Test image:

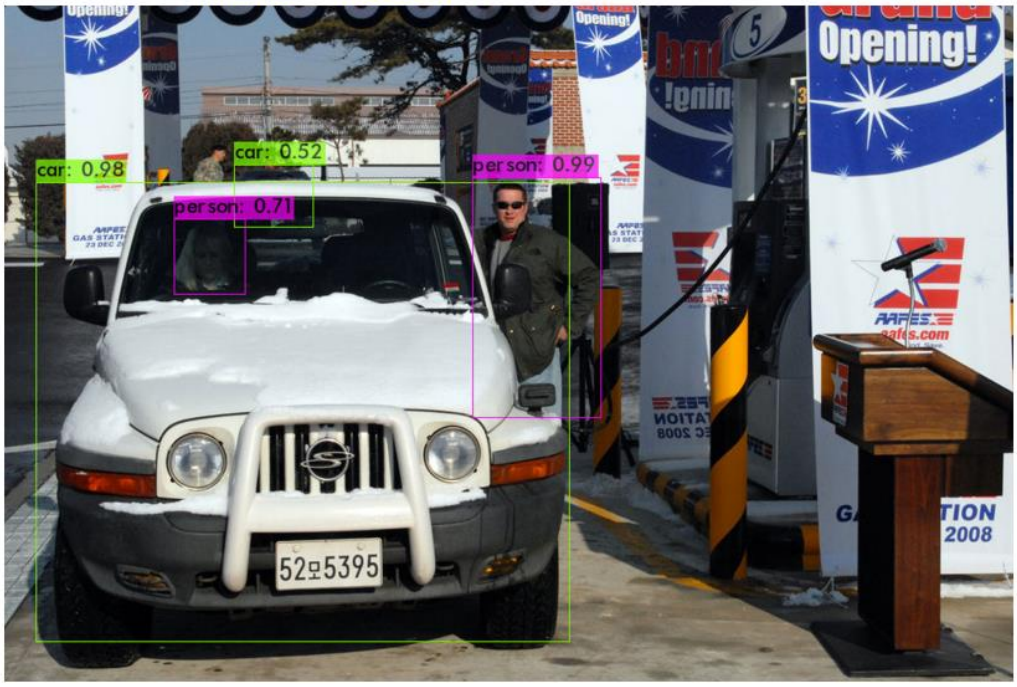
1. Predictions from Yolov3:

```
(predictions:2021): Gtk-WARNING **: 18:51:57.313: cannot open display:
```



2. Predictions from Yolov4:

```
(predictions:2677): Gtk-WARNING **: 18:55:57.952: cannot open display:
```



Conclusion:

From mean average precision score and predictions on image, it can be said that Yolov4 does a much better job of object detection in comparison to Yolov3.

Other techniques which can increase the mAP score for Yolov3 could be:

- ⇨ Increase in datasize, by first augmenting the data and then creating bounding boxes for them. (can be done through softwares)
- ⇨ Variations in training and test data can be tried out.